



デベロッパーガイド

Amazon Simple Notification Service



Amazon Simple Notification Service: 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 が所有しない商標はすべてそれぞれの所有者に所属します。所有者は必ずしも Amazon との提携や関連があるわけではありません。また、Amazon のサポートを受けているとはかぎりません。

Table of Contents

「Amazon SNS とは」	1
仕組み	1
Amazon SNS へアクセスする	2
一般的な Amazon SNS シナリオ	3
アプリケーション統合	3
アプリケーションアラート	4
ユーザー通知	4
モバイルプッシュ通知	5
Amazon SNS の料金	5
Amazon SNS の特徴と機能	5
一般的な共通共有サービス	7
AWS SDKs の使用	10
トピックの作成とメッセージの発行	12
セットアップする	12
アカウントと IAM ユーザーの作成	12
次の手順	14
ステップ 1: トピックの作成	15
AWS マネジメントコンソール	15
AWS SDKs	18
ステップ 2: トピックに対するサブスクリプションを作成する	35
エンドポイントを Amazon SNS トピックにサブスクライブするには	36
ステップ 3: メッセージを発行する	37
AWS マネジメントコンソール	37
AWS SDKs	39
容量の大きなメッセージペイロード	66
メッセージ属性	77
メッセージのバッチ処理	82
ステップ 4: サブスクリプションとトピックを削除する	86
AWS マネジメントコンソール	86
AWS SDKs	87
次の手順	98
FIFO トピックを使用したメッセージの順序付けと重複排除	99
高スループット FIFO トピック	99
ユースケース	99

パーティションとデータ分散	100
パーティションとデータ分散	101
高スループットを有効にする	101
Amazon SQS キューの高スループットモードを有効にする	102
FIFO トピックのユースケース	102
メッセージの順序の詳細	103
メッセージのグループ化	107
メッセージグループ ID 別のデータ配布によりパフォーマンスを改善	108
メッセージ配信	109
メッセージのフィルター処理	110
メッセージ重複排除	111
メッセージセキュリティ	114
メッセージの耐久性	115
メッセージのアーカイブとリプレイ	117
メッセージのアーカイブとリプレイとは	117
トピック所有者の場合	118
トピックサブスクライバーの場合	123
コードの例	127
FIFO の例 (AWS SDKs)	127
FIFO の例 (CloudFormation)	140
メッセージのフィルター処理	145
サブスクリプションフィルターポリシーの範囲	145
サブスクリプションフィルターポリシー	146
Amazon SNS サンプルフィルターポリシー	147
フィルターポリシーの制約	150
AND/OR ロジック	154
キーの一致	158
数値の一致	160
文字列値の一致	163
サブスクリプションフィルターポリシーを適用する	171
AWS マネジメントコンソール	172
AWS CLI	173
AWS SDKs	174
Amazon SNS API	179
AWS CloudFormation	179
サブスクリプションフィルターポリシーを削除する	180

の使用 AWS マネジメントコンソール	180
の使用 AWS CLI	180
Amazon SNS API の使用	181
メッセージデータ保護	182
メッセージデータ保護とは	182
メッセージデータ保護を使用する理由	182
データ保護ポリシー	183
データ保護ポリシーとは	183
データ保護ポリシーの構成の概要	184
IAM プリンシパルを決定する方法	187
データ保護ポリシーオペレーション	187
データ保護ポリシーの例	196
データ保護ポリシーの作成	202
データ保護ポリシーを削除する	211
データ識別子	212
マネージドデータ識別子	212
カスタムデータ識別子	252
メッセージ配信	255
raw メッセージの配信	255
を使用した raw メッセージ配信の有効化 AWS マネジメントコンソール	256
メッセージ形式の例	256
Amazon SQS サブスクリプションのメッセージ属性と raw メッセージ配信	257
アカウント間での配信	257
キューの所有者がサブスクリプションを作成する	258
キュー作成サブスクリプションを所有していないユーザー	259
サブスクリプション解除リクエストでサブスクリプションに認証を要求させるにはどうすればよいですか。	262
リージョン間での配信	263
オプトインリージョン	263
メッセージ配信ステータス	266
配信ステータスのログ記録に関する前提条件	267
を使用した配信ステータスのログ記録の設定 AWS マネジメントコンソール	268
AWS SDKs をを使用した配信ステータスのログ記録の設定	269
AWS トピック属性を設定するための SDK の例	272
を使用した配信ステータスのログ記録の設定 CloudFormation	280
メッセージ配信の再試行	281

配信プロトコルとポリシー	282
配信ポリシーの段階	283
HTTP/S 配信ポリシーの作成	283
デッドレターキュー	290
メッセージ配信が失敗する理由	291
デッドレターキューのしくみ	292
メッセージがデッドレターキューに移動する仕組み	292
メッセージをデッドレターキューから移動する方法	292
デッドレターキューのモニタリングとログ記録方法	293
デッドレターキューを設定する	293
メッセージのアーカイブと分析	299
リソースの管理と最適化	300
Tagging	300
コスト割り当てのタグ付け	300
アクセス制御のタグ付け	301
リソース検索およびフィルタリングのタグ付け	302
タグを設定する	303
Amazon SNS イベントのソースと送信先	310
イベントソース	310
分析	310
アプリケーション統合	311
請求情報とコスト管理	312
ビジネスアプリケーション	313
コンピューティング	313
コンテナ	314
カスタマーエンゲージメント	315
データベース	316
デベロッパーツール	317
フロントエンドウェブおよびモバイル	318
ゲーム開発	319
IoT	320
機械学習	320
管理とガバナンス	322
メディア	324
移行と転送	324
ネットワーキングとコンテンツ配信	325

セキュリティ、アイデンティティ、コンプライアンス	326
サーバーレス	328
Storage	328
追加のイベントソース	330
イベントの送信先	331
A2A 送信先	331
A2Pの送信先	332
Application-to-Application (A2A) メッセージング	335
Firehose 配信ストリームへのファンアウト	335
前提条件	336
トピックへ配信ストリームをサブスクライブする	338
複数の配信ストリーム送信先にわたるメッセージを管理する	339
メッセージのアーカイブと分析のユースケース例	352
Lambda 関数へのファンアウト	364
前提条件	365
関数をトピックへサブスクライブする	365
Amazon SQS キューへのファンアウト	366
トピックへキューをサブスクライブする	367
を使用して Amazon SNS から Amazon SQS へのメッセージングを自動化する AWS	
CloudFormation	375
通知の HTTPS エンドポイントへのファンアウト	382
トピックにエンドポイントをサブスクライブする	384
メッセージ署名を検証する	393
メッセージ形式を解析する	399
Event Fork Pipelines への AWS ファンアウトイベント	409
AWS Event Fork Pipelines の仕組み	410
Event AWS Fork Pipelines のデプロイ	414
Event Fork Pipelines サンプルアプリケーションをデプロイしてテストする	415
トピックにイベントパイプラインをサブスクライブする	424
EventBridge スケジューラの使用	433
実行ロールの設定	434
新しいスケジュールを作成する	434
関連リソース	439
Application-to-Person メッセージング	440
モバイルテキストメッセージング	440
Amazon SNS が SMS メッセージを配信する方法	442

開始方法	442
送信元アイデンティティ	448
設定	450
SMS メッセージングのベストプラクティス	531
モバイルプッシュ通知を送信する	546
Amazon SNS ユーザー通知の仕組み	547
Amazon SNS でプッシュ通知を設定する	548
モバイルアプリのセットアップ	549
Amazon SNS を使用したモバイルプッシュ通知	568
モバイルアプリケーションの属性	582
モバイルアプリケーションのイベント	586
モバイルプッシュ API アクション	589
モバイルプッシュ API の一般的なエラー	591
モバイルプッシュの TTL	603
サポート対象のリージョン	605
モバイルプッシュ通知のベストプラクティス	606
E メールサブスクリプションの設定と管理	607
AWS マネジメントコンソール	607
AWS SDKs	609
コードの例	642
基本	643
Hello Amazon SNS	644
アクション	656
シナリオ	842
DynamoDB テーブルにデータを送信するアプリケーションを構築する	843
Amazon SNS アプリケーションの構築	844
プッシュ通知のプラットフォームエンドポイントを作成します	845
サーバーレスアプリケーションを作成して写真の管理	848
Amazon Textract エクスプローラーアプリケーションを作成する	852
FIFO トピックを作成して発行する	854
動画内の人物や物体を検出する	866
トピックへの SMS メッセージの発行	867
大きなメッセージを発行する	874
SMS テキストメッセージを発行する	877
メッセージをキューに発行する	886
API Gateway を使用して Lambda 関数を呼び出す	1023

スケジュールされたイベントを使用した Lambda 関数の呼び出し	1025
サーバーレスサンプル	1027
Amazon SNS トリガーから Lambda 関数を呼び出す	1028
セキュリティ	1038
データ暗号化	1039
サーバー側の暗号化を使用したデータの保護	1039
キー管理	1042
サーバー側の暗号化によるトピック暗号化のセットアップ	1049
暗号化された Amazon SQS キューサブスクリプションによるトピック暗号化の設定	1051
VPC エンドポイントによるトラフィックの保護	1056
VPC エンドポイントの作成	1057
VPC ポリシーを作成する	1059
VPC からメッセージを発行する	1060
接続にデュアルスタックエンドポイントを使用する	1073
メッセージデータ保護セキュリティ	1073
ID とアクセス管理	1073
オーディエンス	1074
アイデンティティを使用した認証	1074
ポリシーを使用したアクセスの管理	1075
アクセスコントロール	1077
アクセスコントロールのユースケース	1077
アクセスポリシーの主な概念	1078
アーキテクチャの概要	1081
アクセスポリシー言語を使用する	1083
評価論理	1084
Amazon SNS アクセスコントロールのケース例	1089
Amazon SNS と IAM が連携する仕組み	1101
AWS マネージドポリシー	1101
ポリシーアクション	1108
ポリシーリソース	1108
ポリシー条件キー	1109
ACL	1109
ABAC	1110
一時認証情報	1110
プリンシパルアクセス権限	1110
サービスロール	1111

サービスリンクロール	1111
アイデンティティベースのポリシーの例	1111
アイデンティティベースのポリシー	1115
リソースベースのポリシー	1115
アイデンティティベースのポリシーを使用する	1116
カスタム IAM ポリシーの管理	1124
一時的な認証情報を使用する	1125
API アクセス許可のリファレンス	1126
ログ記録とモニタリング	1130
CloudTrail ログ	1131
CloudWatch を使用したトピックのモニタリング	1141
コンプライアンス検証	1157
耐障害性	1157
インフラストラクチャセキュリティ	1158
セキュリティのベストプラクティス	1159
予防的ベストプラクティス	1159
AWS X-Ray を使用したトピックのトラブルシューティング	1165
アクティブトレース	1165
アクセス許可	1166
アクティブトレースの有効化	1166
AWS SDK を使用して Amazon SNS トピックでアクティブトレースを有効にする	1167
AWS CLI を使用して Amazon SNS トピックでアクティブなトレースを有効にする	1167
を使用して Amazon SNS トピックでアクティブトレースを有効にする CloudFormation ..	1168
アクティブトレースが有効になっていることを確認する	1168
テスト	1169
Amazon SNS のドキュメント履歴	1171
.....	mclxxx

Amazon SNS とは

Amazon Simple Notification Service (Amazon SNS) は、パブリッシャー (プロデューサー) からサブスクライバー (コンシューマー) へのメッセージ配信を提供する完全マネージド型サービスです。発行者は、論理アクセスポイントおよび通信チャネルであるトピックにメッセージを送信することで、受信者と非同期的に通信します。

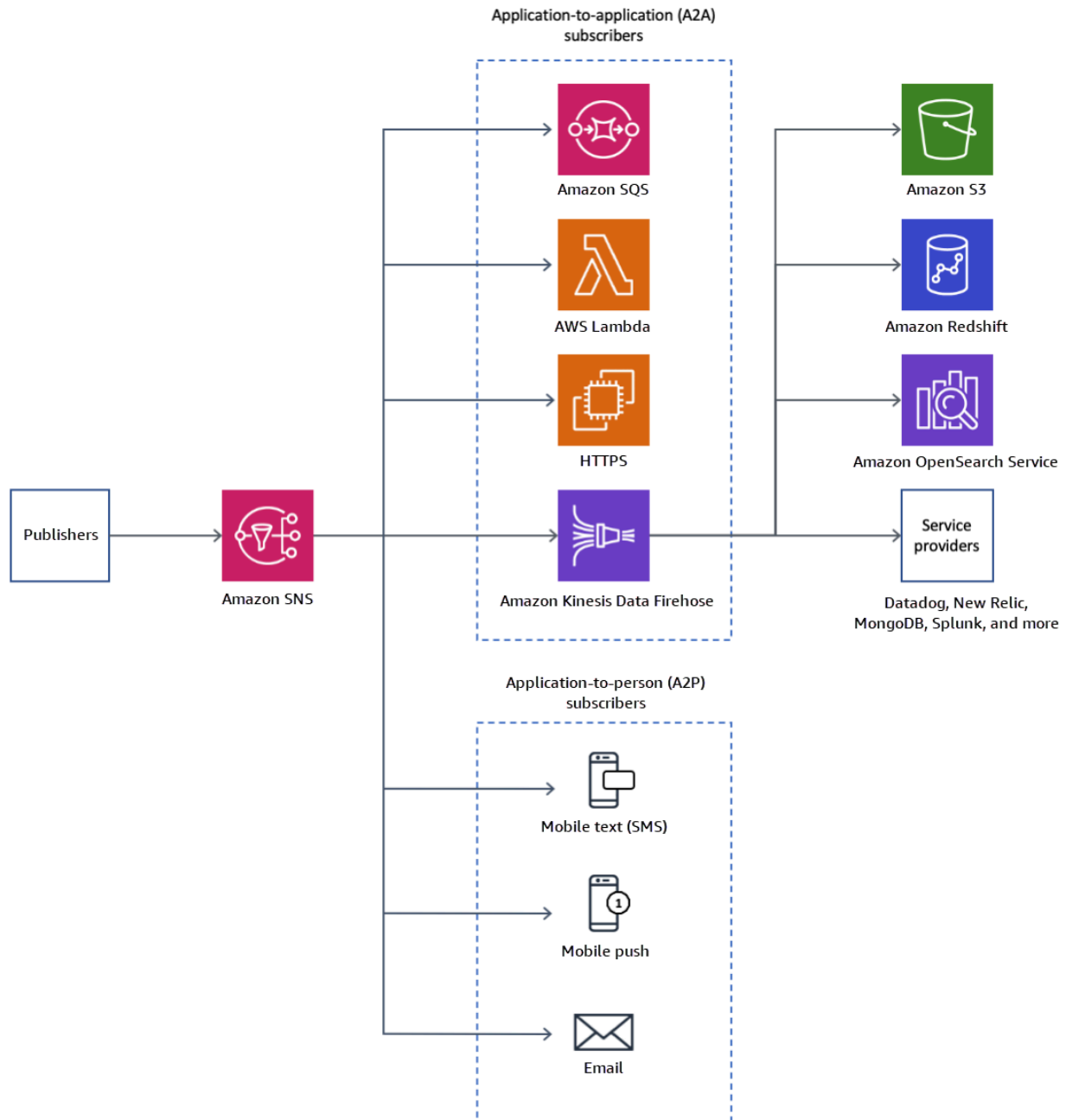
仕組み

SNS では、パブリッシャーは通信チャネルとして機能するトピックにメッセージを送信します。トピックは論理アクセスポイントとして機能し、メッセージがさまざまなプラットフォームにある複数のサブスクライバーに配信されるようにします。

SNS トピックのサブスクライバーは、次のようなユースケースに応じて、さまざまなエンドポイントを経由してメッセージを受信できます。

- Amazon SQS
- Lambda
- HTTP/HTTPS エンドポイント
- E メール
- モバイルプッシュ通知
- モバイルテキストメッセージ (SMS)
- Amazon Data Firehose
- サービスプロバイダー (Datadog、MongoDB、Splunk など)

SNS は Application-to-Application (A2A) メッセージングと Application-to-Person (A2P) メッセージングの両方をサポートしているため、さまざまなアプリケーション間でメッセージを送信したり、携帯電話やE メールアドレスへ直接メッセージを送信したりすることができます。



Amazon SNS へアクセスする

任意のインタラクション方法に応じて、コンソール AWS CLI、または AWS SDKs を使用して Amazon SNS にアクセスして管理できます。コンソールは基本的なタスク用のグラフィカルイン

ターフェイスを提供しますが、AWS CLI および SDKsはより複雑なユースケース用の高度な設定および自動化機能を提供します。

- [Amazon SNS コンソール](#)は、トピックとサブスクリプションの作成、メッセージの送信と受信、イベントとログのモニタリングを行うための便利なユーザーインターフェイスを提供します。
- AWS Command Line Interface (AWS CLI) を使用すると、高度な設定と自動化のユースケースのために Amazon SNS API に直接アクセスできます。詳細については、「[AWS CLIでの Amazon SNS の使用](#)」を参照してください。
- AWS は、さまざまな言語で SDKsを提供します。詳細については、「[SDK とツールキット](#)」を参照してください。

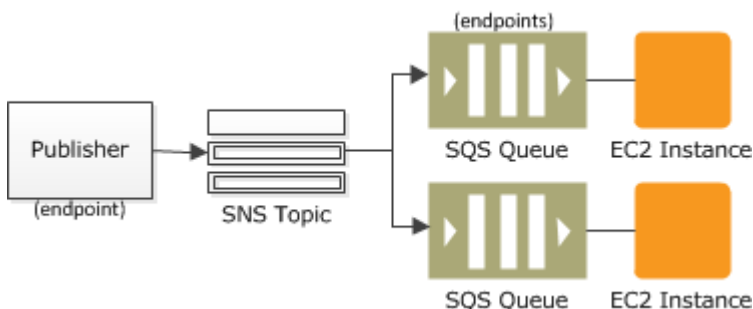
一般的な Amazon SNS シナリオ

これらの一般的な Amazon SNS シナリオを使用して、スケーラブルなイベント駆動型アーキテクチャを実装し、アプリケーションとユーザー間の信頼性の高いリアルタイムの通信を確保します。

アプリケーション統合

ファンアウトシナリオでは、SNS トピックに発行されたメッセージがレプリケートされ、Firehose 配信ストリーム、Amazon SQS キュー、HTTP(S) エンドポイント、Lambda 関数などの複数のエンドポイントにプッシュされます。これで並列非同期処理が可能になります。

例えば、ある製品が注文されるたびにトピックに SNS メッセージを送信するアプリケーションを開発できます。そうすると、その SNS トピックにサブスクライブされている SQS キューは、新しい注文に関する同一の通知を受け取ります。SQS キューの 1 つに添付された Amazon Elastic Compute Cloud (Amazon EC2) サーバーインスタンスは、注文の処理や実行を処理できます。また、受信したすべての注文を分析するために、別の Amazon EC2 サーバーインスタンスをデータウェアハウスにアタッチすることもできます。



ファンアウトは、本番稼働用環境に送信されるデータを、テスト環境にレプリケートすることにも使用できます。前の例から発展させて、新しい受注用に同じ SNS トピックに別の SQS キューをサブ

スクライブすることもできます。次に、この新しい SQS キューをテスト環境に添付することで、本番稼働用環境から受け取ったデータを使用してアプリケーションの改善とテストを継続することができます。

Important

本番稼働用データをテスト環境に送信する前に、データのプライバシーとセキュリティを考慮してください。

詳細については、以下のリソースを参照してください。

- [Firehose 配信ストリームへのファンアウト](#)
- [Amazon SNS 通知の Lambda 関数へのファンアウトと自動処理](#)
- [Amazon SNS 通知の Amazon SQS キューへのファンアウトと非同期処理](#)
- [Amazon SNS 通知の HTTPS エンドポイントへのファンアウト](#)
- [Amazon SNS と AWS コンピューティング、ストレージ、データベース、ネットワーキングサービスを使用したイベント駆動型コンピューティング](#)

アプリケーションアラート

アプリケーションおよびシステムアラートは、事前定義されたしきい値によってトリガーされた送信される通知です。Amazon SNS は、SMS および E メールを使用して、指定されたユーザーにこれらの通知を送信できます。例えば、Amazon EC2 Auto Scaling グループに対する特定の変更、Amazon S3 バケットにアップロードされた新しいファイル、Amazon CloudWatch でメトリクスしきい値に違反したなどのイベントが発生した場合、ただちに通知を受信することができます。詳細については、『Amazon CloudWatch ユーザーガイド』の「[Amazon SNS 通知の設定](#)」を参照してください。

ユーザー通知

Amazon SNS では、プッシュ E メールメッセージとテキストメッセージ (SMS メッセージ) を個人またはグループに送信できます。例えば、e コマースの注文確認をユーザー通知として送信できます。Amazon SNS を使用した SMS メッセージの送信の詳細については、「[Amazon SNS を使用したモバイルテキストメッセージ](#)」を参照してください。

モバイルプッシュ通知

モバイルプッシュ通知を使用すると、モバイルアプリケーションに直接メッセージを送信できます。例えば、Amazon SNS を使用して、アプリケーションに更新通知を送信できます。通知メッセージには、更新をダウンロードおよびインストールするためのリンクを含めることができます。Amazon SNS を使用したプッシュ通知メッセージの送信の詳細については、「[Amazon SNS でモバイルプッシュ通知を送信する](#)」を参照してください。

Amazon SNS の料金

Amazon SNS には前払いのコストはかかりません。発行するメッセージの数、配信する通知の数、およびトピックとサブスクリプションを管理するための追加の API コールに基づいてお支払いいただきます。配信料金は、エンドポイントのタイプによって異なります。Amazon SNS 無料利用枠を使用すれば、無料で始められます。詳細については、「[世界各国の SMS 料金](#)」を参照してください。

Amazon SNS の特徴と機能

Amazon SNS は、アプリケーションとユーザー間のメッセージングを強化するように設計された包括的な機能セットを提供します。これらの機能により、シームレスな通信、安全なメッセージ配信、堅牢なメッセージ管理が可能になり、幅広いメッセージングのユースケースで高可用性、耐久性、柔軟性を確保できます。

Application-to-Application (A2A) メッセージング

[Application-to-Application メッセージング](#)は、配信ストリーム、Lambda 関数、Amazon SQS キュー、HTTP/S エンドポイント、AWS Event Fork Pipelines などのサブスクライバーをサポートしています。これにより、イベント駆動型アーキテクチャで効率的なメッセージ配信が可能になります。

Application-to-Person の通知

[Application-to-Person 通知](#)は、モバイルアプリケーション、携帯電話番号、E メールアドレスなどのサブスクライバーへのユーザー通知を提供します。

標準および FIFO トピック

[FIFO トピック](#)により、厳密なメッセージの順序付け、メッセージのグループ化、重複排除が確保され、FIFO キューおよび標準キューがメッセージ処理をサブスクライブできます。[標準ト](#)

[ピック](#)は、メッセージの順序付けと重複の可能性が重要でない場合に使用され、より広範なユースケースのすべての配信プロトコルをサポートします。

メッセージの耐久性

Amazon SNS では、メッセージの耐久性を実現するために連携するいくつかの戦略を使用しています。

- 公開されたメッセージは、地理的に分離された複数のサーバーおよびデータセンターの間で保存されます。
- サブスクライブされたエンドポイントが利用できない場合、Amazon SNS は[配信の再試行ポリシー](#)を実行します。
- 配信再試行ポリシーが終了する前に配信されなかったメッセージを保持するには、[デッドレターキュー](#)を作成できます。

メッセージのアーカイブ、リプレイ、および分析

Amazon SNS により、[SNS トピックへの Firehose 配信ストリーム](#)のサブスクライブを含め、複数の方法でメッセージをアーカイブできます。これにより、Amazon Simple Storage Service (Amazon S3) バケット、Amazon Redshift テーブルなどの分析エンドポイントに通知を送信することができます。また、Amazon SNS FIFO トピックは、トピック所有者がトピック内でメッセージを保存 (またはアーカイブ) できる、ノーコードのインプレースメッセージアーカイブとしてメッセージのアーカイブとリプレイをサポートしています。トピックサブスクライバーは、アーカイブされたメッセージをサブスクライブされたエンドポイントに対してリトリート (またはリプレイ) できます。詳細については、「[FIFO トピックの Amazon SNS メッセージのアーカイブとリプレイ](#)」を参照してください。

メッセージ属性

[Amazon SNS メッセージ属性](#) により、メッセージに関する任意のメタデータを提供できます。

メッセージのフィルター処理

デフォルトでは、各受信者は、トピックに対して発行されたすべてのメッセージを受信します。メッセージのサブセットを受信する場合、受信者はトピックのサブスクリプションにフィルターポリシーを割り当てる必要があります。サブスクライバーはフィルターポリシーの範囲を定義して、ペイロードベースまたは属性ベースのフィルタリングを有効にすることもできます。フィルターポリシー範囲のデフォルト値は MessageAttributes です。受信メッセージ属性がフィルターポリシー属性と一致すると、メッセージはサブスクライブされたエンドポイントに配信されます。それ以外の場合、メッセージは除外されます。フィルターポリシーの範囲が MessageBody の場合、フィルターポリシー属性がペイロードと照合されます。詳細については、「[メッセージのフィルター処理](#)」を参照してください。

メッセージセキュリティ

サーバー側の暗号化は、[が提供する暗号化キーを使用して、Amazon SNS トピックに保存されているメッセージの内容を保護します AWS KMS](#)。詳細については、「[the section called “サーバー側の暗号化を使用したデータの保護”](#)」を参照してください。また、Amazon SNS と仮想プライベートクラウド (VPC) の間にプライベート接続を確立することもできます。詳細については、「[the section called “VPC エンドポイントによるトラフィックの保護”](#)」を参照してください。

AWS Amazon SNS で一般的に使用される サービス

Amazon SNS を複数の AWS クラウド サービスに統合して、メッセージ処理の強化、アクセスコントロールの改善、イベント駆動型処理の有効化、リソースの自動化を行います。この統合により、パフォーマンスが最適化され、セキュリティが強化され、運用が効率化されます。

Amazon CloudWatch

Amazon CloudWatch は Amazon SNS のモニタリングとオプザバビリティを提供し、メッセージ配信の追跡、異常の検出、問題のトラブルシューティングに役立ちます。CloudWatch を使用すると、次のことが可能になります。

- Amazon SNS メトリクスのモニタリング。こうしたメトリクスには、トピックやサブスクリプション全体における発行、配信、失敗したメッセージの数などがあります。
- CloudWatch アラームのセットアップ。Amazon SNS メトリクスが事前定義されたしきい値 (配信失敗の増加やスロットリングなど) を超えた場合に、自動アクションをトリガーします。
- CloudWatch Logs を使用した Amazon SNS 配信ステータスの取得。HTTP/ S、Lambda、Amazon SQS の各エンドポイントに送信されるメッセージの配信ステータスを取得して、デバッグや監査に利用できます。

詳細については、「[CloudWatch を使用した Amazon SNS のモニタリング](#)」を参照してください。

Amazon SQS

Amazon SQS は、分散ソフトウェアコンポーネント間で安全かつ堅牢でスケーラブルな通信を可能にする、完全マネージド型のメッセージキューイングサービスです。メッセージのバッファリング、信頼性の高い配信の確保、およびメッセージ損失によるシステム障害の防止を行うことで、アプリケーションアーキテクチャのデカップリングに役立ちます。Amazon SQS は、以下の方法で Amazon SNS と統合されます。

- [デッドレターキュー](#) – Amazon SNS は、トラブルシューティングや再処理のために、配信不能メッセージを Amazon SQS デッドレターキューにルーティングできます。
- [トピックのサブスクリプション](#) – Amazon SQS キューを Amazon SNS トピックにサブスクライブして、Amazon SNS が Amazon SQS を使用して複数のコンシューマーにメッセージをファンアウトできるようにします。
- [FIFO キューのサポート](#) – Amazon SQS FIFO キューは Amazon SNS FIFO トピックにサブスクライブできるため、厳密なメッセージの順序付けと 1 回限りの処理を確実に行うことができます。[標準 Amazon SQS キュー](#)も Amazon SNS トピックにサブスクライブできますが、順序付けられたメッセージの配信や重複排除は保証されません。

CloudFormation

CloudFormation は、Infrastructure as Code (IaC) を使用して、Amazon SNS トピックやサブスクリプションなどの AWS リソースのプロビジョニングと管理を自動化します。を使用すると CloudFormation、次のことができます。

- Amazon SNS トピック、サブスクリプション、およびアクセス許可の定義。バージョン管理された再利用可能なテンプレートを使用します。
- 複数の AWS アカウント およびリージョンに Amazon SNS リソースを一貫してデプロイします。
- Amazon SNS 設定の更新または変更。変更セットを使用します。手動で行う必要はありません。

詳細については、「[AWS CloudFormation ユーザーガイド](#)」を参照してください。

AWS CloudTrail

CloudTrail は Amazon SNS の API アクティビティを可視化し、Amazon SNS トピック、サブスクリプション、メッセージへのアクセスのモニタリングや監査に役立ちます。CloudTrail を使用すると、次のことが可能になります。

- Amazon SNS に対して行われた API コールの追跡。トピック、サブスクリプション、アクセス許可にアクセスまたは変更を加えたユーザーが対象となります。
- 不正なアクティビティや予期しないアクティビティの検出。セキュリティおよびコンプライアンスを目的としてログを分析し、アクティビティの検出を行います。
- Amazon CloudWatch または AWS Security Hub CSPMとの統合。この統合により、異常な Amazon SNS アクションに基づいてアラートを作成できます。

詳細については、「[を使用した AWS SNS API コールのログ記録 AWS CloudTrail](#)」を参照してください。

AWS Lambda

AWS Lambda は、イベントに応じてコードを自動的に実行するサーバーレスコンピューティングサービスであり、サーバーのプロビジョニングや管理が不要になります。これにより、可用性の高いコンピューティング環境で自動的にスケールして実行するイベント駆動型アプリケーションを構築できます。

Amazon SNS は、Lambda 関数を Amazon SNS トピックにサブスクライブできるようにすることで、Lambda と統合します。Amazon SNS トピックがメッセージを受け取ると、Lambda 関数がトリガーされ、リアルタイム処理、オートメーション、およびアプリケーションロジックの実行が可能になります。この統合は、一般的に以下の用途で使用されます。

- [イベント駆動型処理](#) – Amazon SNS メッセージに応答して関数を自動的にトリガーします。
- [データ変換](#) – Amazon SNS メッセージを、他のサービスに転送する前に変更またはフィルタリングします。
- 自動ワークフロー – アプリケーションアラート、システムモニタリング、またはイベントオーケストレーションの通知を処理します。

AWS Identity and Access Management (IAM)

IAM は AWS リソースの安全なアクセスコントロールを提供し、Amazon SNS トピックにアクセスできるユーザー、実行できるアクション、および条件を管理できます。IAM を使用すると、次のことが可能になります。

- ユーザーとサービスの認証。ユーザーやサービスが Amazon SNS トピックとやり取りする前に認証を行います。
- きめ細かいアクセス許可の定義。ユーザーやロールが発行、サブスクライブ、管理できる Amazon SNS トピックを指定できます。
- アイデンティティベースのポリシーを使用して、特定の IP アドレスや条件へのアクセスを制限するなど AWS アカウント、セキュリティのベストプラクティスを適用します。

詳細については、「[Amazon SNS でのアイデンティティベースのポリシーを使用する](#)」を参照してください。

AWS Key Management Service (AWS KMS)

AWS KMS は、メッセージの機密性のためにサーバー側の暗号化 (SSE) を有効にすることで、Amazon SNS のセキュリティを強化します。を使用すると AWS KMS、次のことができます。

- 管理またはカスタマー管理 AWS の暗号化キー (CMK) を使用して、保管中の Amazon SNS メッセージを暗号化します。CMKs

- Amazon SNS トピックへのアクセスの制御。きめ細かなキーポリシーを定義して、トピックを発行またはサブスクライブできるユーザーを制限します。
- セキュリティや規制要件へのコンプライアンスの確保。AWS CloudTrailを使用してキーの使用状況を監査します。

詳細については、「[Amazon SNS 暗号化キーとコストの管理](#)」を参照してください。

AWS X-Ray

X-Ray は Amazon SNS のトレースを提供し、イベント駆動型アーキテクチャを経由したメッセージのフローの分析とデバッグに役立ちます。X-Ray を使用すると、次のことが可能になります。

- Lambda AWS のサービス、Amazon SNS、HTTP/S エンドポイントなど、複数の にまたがる Amazon SNS メッセージ配信をトレースします。Amazon SQS
- レイテンシーのボトルネックの特定。メッセージの発行、配信、処理にかかる時間を視覚化します。
- エラーと再試行の検出。Amazon SNS メッセージフローでエラーや再試行を検出し、失敗した配信や処理時間の遅延をトラブルシューティングします。

詳細については、「[Amazon SNS でのアクティブトレース](#)」を参照してください。

AWS SDK での Amazon SNS 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

SDK ドキュメント	コード例
AWS SDK for Kotlin	AWS SDK for Kotlin 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

Amazon SNS 固有の例については、「[SDK を使用した Amazon SNS のコード例 AWS SDKs](#)」を参照してください。

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。

Amazon SNS トピックの作成とメッセージの発行

このトピックでは、Amazon SNS リソースを管理するための基本的な手順を説明します。特にトピック、サブスクリプション、メッセージの発行に焦点を当てています。まず、Amazon SNS に必要なアクセス許可を設定し、Amazon SNS リソースを作成および管理するための適切なアクセス許可があることを確認します。次に、新しい Amazon SNS トピックを作成します。これは、メッセージの管理とサブスクライバーへの配信のための中央ハブとして機能します。トピックを作成したら、このトピックのサブスクリプションを作成し、特定のエンドポイントがそのトピックに発行されたメッセージを受信できるようにします。

トピックとサブスクリプションが設定されたら、そのトピックにメッセージを発行します。Amazon SNS からサブスクリプションが作成されたすべてのエンドポイントにメッセージが効率的に配信されることを確認します。最後に、サブスクリプションとトピックの両方を削除し、管理した Amazon SNS リソースのライフサイクルの完了を確認します。このアプローチにより、Amazon SNS の基本的な操作を明確に理解し、Amazon SNS コンソールを使用してメッセージングワークフローを管理するために必要な実践的なスキルを身に付けることができます。

Amazon SNS のアクセスをセットアップする

Amazon SNS を初めて使用する前に、以下のステップを完了する必要があります。

AWS アカウント と IAM ユーザーを作成する

AWS サービスにアクセスするには、まず [AWS アカウント](#) を作成する必要があります。AWS アカウント、アクティビティと使用状況レポートを表示し、認証とアクセスを管理できます。

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 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 AWS アカウント ルートユーザーを使用しないように、ルートユーザーを保護し AWS IAM アイデンティティセンター、 を有効にして管理ユーザーを作成します。

AWS アカウント ルートユーザーを保護する

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 アイデンティティセンター ユーザーガイド」の「[AWS IAM アイデンティティセンターの有効化](#)」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的チュートリアルについては、「AWS IAM アイデンティティセンター 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 アイデンティティセンター 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 アイデンティティセンター ユーザーガイド」の「[グループの追加](#)」を参照してください。

次の手順

Amazon SNS を使用する準備ができたので、次に進みます。

1. [Amazon SNS トピックを作成する](#)
2. [Amazon SNS トピックのサブスクリプションの作成](#)
3. [Amazon SNS メッセージの発行](#)
4. [Amazon SNS トピックおよびサブスクリプションを削除する](#)

Amazon SNS トピックを作成する

Amazon SNS トピックは、通信チャネルとして機能する論理アクセスポイントです。トピックでは、複数のエンドポイント (Amazon SQS、AWS Lambda、HTTP/S、E メールアドレスなど) をグループ化できます。

メッセージプロデューサーシステム (e コマースウェブサイトなど) で当システムのメッセージを必要とする他の複数のサービス (チェックアウトシステムやフルフィルメントシステムなど) にメッセージをブロードキャストするには、プロデューサーシステムのトピックを作成できます。

Amazon SNS で最も一般的な最初のタスクは、トピックの作成です。このページでは AWS マネジメントコンソール、AWS SDK for Java、および AWS SDK for .NET を使用してトピックを作成する方法を示します。

作成時に、トピックタイプ (標準または FIFO) を選択し、トピックに名前を付けます。トピックを作成したら、トピックのタイプや名前を変更することはできません。その他の設定項目はすべて、トピックの作成時にオプションであり、後で編集できます。

Important

個人を特定できる情報 (PII) などの機密情報や秘匿性の高い情報はトピック名に追加しないでください。トピック名は、CloudWatch Logs を含む他の Amazon Web Services のサービスからアクセスできます。トピック名は、プライベートデータや機密データとして使用することを意図していません。

を使用してトピックを作成するには AWS マネジメントコンソール

Amazon SNS でトピックを作成すると、メッセージ配信の基盤が確立され、メッセージを発行して複数のサブスクライバーにファンアウトできるようになります。このステップは、トピックのタイプ、暗号化設定、アクセスポリシーを設定し、トピックが組織のセキュリティ、コンプライアンス、運用要件を満たすようにするために不可欠です。

1. [Amazon SNS コンソール](#)にサインインします。
2. 次のいずれかを行います。

- AWS アカウント 以前に トピックを作成したことがない場合は、ホームページの Amazon SNS の説明をお読みください。

- AWS アカウント 以前に トピックが作成されている場合は、ナビゲーションパネルでトピックを選択します。
- 3. [トピック] ページで、[トピックの作成] を選択します。
- 4. [サブスクリプションの作成] ページで [詳細] セクションで、以下を実行します。
 - a. [タイプ] で、トピックタイプ (標準またはFIFO) を選択します。
 - b. トピックの名前を入力します。[FIFO トピック](#)で、名前の末尾に .fifo を追加します。
 - c. (オプション) トピックの表示名を入力します。

 Important

E メールエンドポイントをサブスクライブする場合、Amazon SNS トピックの表示名と送信 E メールアドレス (例えば、no-reply@sns.amazonaws.com) の合計文字数が 320 UTF-8 文字を超えてはなりません。Amazon SNS トピックの表示名を設定する前に、サードパーティーのエンコードツールを使用して送信アドレスの長さを確認できます。

- d. (オプション) FIFO トピックで、[コンテンツベースのメッセージ重複排除] を選択して、デフォルトのメッセージの重複排除を有効にします。詳細については、「[FIFO トピックの Amazon SNS メッセージ重複排除](#)」を参照してください。
- 5. (オプション) [暗号化] セクションを展開し、以下の操作を実行します。詳細については、「[サーバー側の暗号化を使用した Amazon SNS データの保護](#)」を参照してください。
 - a. [暗号化の有効化] を選択します。
 - b. AWS KMS キーを指定します。詳細については、「[重要な用語](#)」を参照してください。

KMS タイプごとに、[Description] (説明)、[Account] (アカウント)、および [KMS ARN] が表示されます。

 Important

KMS の所有者ではない場合、または kms:ListAliases および kms:DescribeKey の許可がないアカウントでログインした場合、Amazon SNS コンソールで KMS に関する情報を表示できません。

これらの許可を付与するように、KMS の所有者へ依頼してください。詳細については、「AWS Key Management Service デベロッパーガイド」の「[AWS KMS API アクセス権限: アクションとリソースのリファレンス](#)」を参照してください。

- Amazon SNS (デフォルト) エイ alias/aws/sns の AWS マネージド KMS はデフォルトで選択されています。

 Note

以下に留意してください。

- AWS マネジメントコンソール を使用してトピックの Amazon SNS の AWS マネージド KMS を初めて指定すると、 は Amazon SNS の AWS マネージド KMS AWS KMS を作成します。
 - または、SSE が有効になっているトピックで Publishアクションを初めて使用すると、 は Amazon SNS の AWS マネージド KMS AWS KMS を作成します。
- AWS アカウントからカスタム KMS を使用するには、KMS キーフィールドを選択し、リストからカスタム KMS を選択します。

 Note

カスタム KMS の作成手順については、AWS Key Management Service デベロッパーガイドの「[キーの作成](#)」を参照してください。

- アカウントまたは別の AWS アカウントからカスタム KMS ARN を使用するには AWS 、KMS キーフィールドに入力します。
6. (オプション) デフォルトでは、トピックの所有者のみがトピックを発行またはサブスクライブできます。追加のアクセス許可を設定するには、[アクセスポリシー] セクションを展開します。詳細については、「[Amazon SNS での Identity and Access Management](#)」および「[Amazon SNS アクセスコントロールのケース例](#)」を参照してください。

Note

コンソールを使用してトピックを作成すると、デフォルトのポリシーでは `aws:SourceOwner` 条件キーが使用されます。このキーは `aws:SourceAccount` に類似しています。

7. (オプション) 失敗したメッセージ配信試行を Amazon SNS で再試行する方法を設定するには、[配信再試行ポリシー (HTTP/S)] セクションを展開します。詳細については、「[Amazon SNS メッセージ配信の再試行](#)」を参照してください。
8. (オプション) CloudWatch へのメッセージの配信を Amazon SNS でログに記録する方法を設定するには、[配信ステータスのログ記録] セクションを展開します。詳細については、「[Amazon SNS メッセージ配信ステータス](#)」を参照してください。
9. (オプション) トピックにメタデータタグを追加するには、[タグ] セクションを展開し、[キー] と [値] (オプション) に入力し、[タグの追加] を選択します。詳細については、「[Amazon SNS トピックのタグ付け](#)」を参照してください。
10. [トピックの作成] を選択します。

トピックが作成され、**[MyTopic]** ページが表示されます。

トピックの名前、ARN、(オプション) 表示名、トピック所有者の AWS アカウント ID が詳細セクションに表示されます。

11. トピック ARN をクリップボードにコピーします。例:

```
arn:aws:sns:us-east-2:123456789012:MyTopic
```

AWS SDK を使用してトピックを作成するには

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、「AWS SDK とツールのリファレンスガイド」の「[共有設定ファイルと認証情報ファイル](#)」を参照してください。

次のサンプルコードは、`CreateTopic` を使用する方法を説明しています。

.NET

SDK for .NET

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。

トピックを作成して、個別の名前を付けます。

```
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use Amazon Simple Notification Service
/// (Amazon SNS) to add a new Amazon SNS topic.
/// </summary>
public class CreateSNSTopic
{
    public static async Task Main()
    {
        string topicName = "ExampleSNSTopic";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var topicArn = await CreateSNSTopicAsync(client, topicName);
        Console.WriteLine($"New topic ARN: {topicArn}");
    }

    /// <summary>
    /// Creates a new SNS topic using the supplied topic name.
    /// </summary>
    /// <param name="client">The initialized SNS client object used to
    /// create the new topic.</param>
    /// <param name="topicName">A string representing the topic name.</param>
    /// <returns>The Amazon Resource Name (ARN) of the created topic.</
returns>
```

```
public static async Task<string>
CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)
{
    var request = new CreateTopicRequest
    {
        Name = topicName,
    };

    var response = await client.CreateTopicAsync(request);

    return response.TopicArn;
}
}
```

名前と特定の FIFO および重複除外属性を使用して新しいトピックを作成します。

```
/// <summary>
/// Create a new topic with a name and specific FIFO and de-duplication
attributes.
/// </summary>
/// <param name="topicName">The name for the topic.</param>
/// <param name="useFifoTopic">True to use a FIFO topic.</param>
/// <param name="useContentBasedDeduplication">True to use content-based de-
duplication.</param>
/// <returns>The ARN of the new topic.</returns>
public async Task<string> CreateTopicWithName(string topicName, bool
useFifoTopic, bool useContentBasedDeduplication)
{
    var createTopicRequest = new CreateTopicRequest()
    {
        Name = topicName,
    };

    if (useFifoTopic)
    {
        // Update the name if it is not correct for a FIFO topic.
        if (!topicName.EndsWith(".fifo"))
        {
            createTopicRequest.Name = topicName + ".fifo";
        }
    }
}
```

```
// Add the attributes from the method parameters.
createTopicRequest.Attributes = new Dictionary<string, string>
{
    { "FifoTopic", "true" }
};
if (useContentBasedDeduplication)
{
    createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
}

var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
return createResponse.TopicArn;
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[CreateTopic](#)」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 Amazon Simple Notification Service (Amazon SNS) topic.
/*!
    \param topicName: An Amazon SNS topic name.
    \param topicARNResult: String to return the Amazon Resource Name (ARN) for the
    topic.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::createTopic(const Aws::String &topicName,
                             Aws::String &topicARNResult,
```

```
const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::CreateTopicRequest request;
    request.SetName(topicName);

    const Aws::SNS::Model::CreateTopicOutcome outcome =
snsClient.CreateTopic(request);

    if (outcome.IsSuccess()) {
        topicARNResult = outcome.GetResult().GetTopicArn();
        std::cout << "Successfully created an Amazon SNS topic " << topicName
                    << " with topic ARN '" << topicARNResult
                    << "'." << std::endl;
    }
    else {
        std::cerr << "Error creating topic " << topicName << ":" <<
                    outcome.GetError().GetMessage() << std::endl;
        topicARNResult.clear();
    }

    return outcome.IsSuccess();
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「[CreateTopic](#)」を参照してください。

CLI

AWS CLI

SNS トピックを作成するには

次の create-topic の例では、my-topic という名前の SNS トピックを作成します。

```
aws sns create-topic \
    --name my-topic
```

出力:

```
{
  "ResponseMetadata": {
    "RequestId": "1469e8d7-1642-564e-b85d-a19b4b341f83"
  },
  "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic"
}
```

詳細については、[AWS「コマンドラインインターフェイスユーザーガイド」のAmazon SQS および Amazon SNS でのコマンドラインインターフェイスの使用](#)を参照してください。

AWS

- APIの詳細については、「AWS CLI コマンドリファレンス」の「[CreateTopic](#)」を参照してください。

Go

SDK for Go V2

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 (
    "context"
    "encoding/json"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/sns"
    "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type SnsActions struct {
    SnsClient *sns.Client
}
```

```
// CreateTopic creates an Amazon SNS topic with the specified name. You can
// optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
// based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
    isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
    var topicArn string
    topicAttributes := map[string]string{}
    if isFifoTopic {
        topicAttributes["FifoTopic"] = "true"
    }
    if contentBasedDeduplication {
        topicAttributes["ContentBasedDeduplication"] = "true"
    }
    topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
        Name:      aws.String(topicName),
        Attributes: topicAttributes,
    })
    if err != nil {
        log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
    } else {
        topicArn = *topic.TopicArn
    }

    return topicArn, err
}
```

- APIの詳細については、「AWS SDK for Go API リファレンス」の「[CreateTopic](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class CreateTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicName>

            Where:
                topicName - The name of the topic to create (for example,
mytopic).

            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
    }
}
```

```
String topicName = args[0];
System.out.println("Creating a topic with name: " + topicName);
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();

String arnVal = createSNSTopic(snsClient, topicName);
System.out.println("The topic ARN is" + arnVal);
snsClient.close();
}

public static String createSNSTopic(SnsClient snsClient, String topicName) {
    CreateTopicResponse result;
    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
            .name(topicName)
            .build();

        result = snsClient.createTopic(request);
        return result.topicArn();

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
}
```

- APIの詳細については、「AWS SDK for Java 2.x API リファレンス」の「[CreateTopic](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
  return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[CreateTopic](#)」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 createSNSTopic(topicName: String): String {
    val request =
        CreateTopicRequest {
            name = topicName
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn.toString()
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の「[CreateTopic](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
```

```
use Aws\Sns\SnsClient;

/**
 * Create a Simple Notification Service topics in your AWS account at the
 * requested region.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topicname = 'myTopic';

try {
    $result = $SnSClient->createTopic([
        'Name' => $topicname,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for PHP API リファレンス」の「[CreateTopic](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def create_topic(self, name):
        """
        Creates a notification topic.

        :param name: The name of the topic to create.
        :return: The newly created topic.
        """
        try:
            topic = self.sns_resource.create_topic(Name=name)
            logger.info("Created topic %s with ARN %s.", name, topic.arn)
        except ClientError:
            logger.exception("Couldn't create topic %s.", name)
            raise
        else:
            return topic
```

```
class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
```

```

    """
    Initialize the SnsWrapper.

    :param sns_client: A Boto3 Amazon SNS client.
    """
    self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def create_topic(
        self,
        topic_name: str,
        is_fifo: bool = False,
        content_based_deduplication: bool = False
    ) -> str:
        """
        Create an SNS topic.

        :param topic_name: The name of the topic to create.
        :param is_fifo: Whether to create a FIFO topic.
        :param content_based_deduplication: Whether to use content-based
        deduplication for FIFO topics.
        :return: The ARN of the created topic.
        :raises ClientError: If the topic creation fails.
        """
        try:
            # Add .fifo suffix for FIFO topics
            if is_fifo and not topic_name.endswith('.fifo'):
                topic_name += '.fifo'

            attributes = {}
            if is_fifo:
                attributes['FifoTopic'] = 'true'
                if content_based_deduplication:
                    attributes['ContentBasedDeduplication'] = 'true'

```

```
        response = self.sns_client.create_topic(
            Name=topic_name,
            Attributes=attributes
        )

        topic_arn = response['TopicArn']
        logger.info(f"Created topic: {topic_name} with ARN: {topic_arn}")
        return topic_arn

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error creating topic {topic_name}: {error_code} - {e}")
        raise
```

- APIの詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[CreateTopic](#)」を参照してください。

Ruby

SDK for Ruby

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。

```
# This class demonstrates how to create an Amazon Simple Notification Service
# (SNS) topic.
class SNSTopicCreator
  # Initializes an SNS client.
  #
  # Utilizes the default AWS configuration for region and credentials.
  def initialize
    @sns_client = Aws::SNS::Client.new
  end
end
```

```
# Attempts to create an SNS topic with the specified name.
#
# @param topic_name [String] The name of the SNS topic to create.
# @return [Boolean] true if the topic was successfully created, false
otherwise.
def create_topic(topic_name)
  @sns_client.create_topic(name: topic_name)
  puts "The topic '#{topic_name}' was successfully created."
  true
rescue Aws::SNS::Errors::ServiceError => e
  # Handles SNS service errors gracefully.
  puts "Error while creating the topic named '#{topic_name}': #{e.message}"
  false
end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_name = 'YourTopicName' # Replace with your topic name
  sns_topic_creator = SNSTopicCreator.new

  puts "Creating the topic '#{topic_name}'..."
  unless sns_topic_creator.create_topic(topic_name)
    puts 'The topic was not created. Stopping program.'
    exit 1
  end
end
end
```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for Ruby API リファレンス」の「[CreateTopic](#)」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 make_topic(client: &Client, topic_name: &str) -> Result<(), Error> {
    let resp = client.create_topic().name(topic_name).send().await?;

    println!(
        "Created topic with ARN: {}",
        resp.topic_arn().unwrap_or_default()
    );

    Ok(())
}

```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「[CreateTopic](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```

TRY.
    oo_result = lo_sns->createtopic( iv_name = iv_topic_name ). " oo_result
is returned for testing purposes. "
    MESSAGE 'SNS topic created' TYPE 'I'.
    CATCH /aws1/cx_snstopiclimitexcdex.
    MESSAGE 'Unable to create more topics. You have reached the maximum
number of topics allowed.' TYPE 'E'.
ENDTRY.

```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「[CreateTopic](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

let output = try await snsClient.createTopic(
    input: CreateTopicInput(name: name)
)

guard let arn = output.topicArn else {
    print("No topic ARN returned by Amazon SNS.")
    return
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[CreateTopic](#)」を参照してください。

Amazon SNS トピックのサブスクリプションの作成

[トピック](#)に対して発行されたメッセージを受信するには、そのトピックへの[エンドポイント](#)をサブスクライブする必要があります。エンドポイントをトピックにサブスクライブしてサブスクリプションが確認されると、エンドポイントでは、関連付けられたトピックに発行されたメッセージの受信を開始できます。

Note

他の AWS アカウント の HTTP(S) エンドポイント、E メールアドレス、および AWS リソースは、メッセージを受信する前にサブスクリプションを確認する必要があります。

エンドポイントを Amazon SNS トピックにサブスクライブするには

エンドポイントを Amazon SNS トピックにサブスクライブすると、指定されたエンドポイントへメッセージを配信して、メッセージがトピックに発行されたときに適切なシステムまたはユーザーが通知を受信できるようになります。これは、コンシューマー (アプリケーション、E メール受信者、その他のサービスなど) にトピックをリンクするために不可欠であり、システム間でのシームレスな通信を実現します。

1. [Amazon SNS コンソール](#)にサインインします。
2. 左のナビゲーションペインで、[サブスクリプション] を選択します。
3. [サブスクリプション] ページで [サブスクリプションの作成] を選択します。
4. [サブスクリプションの作成] ページの [詳細] セクションで、以下を実行します。
 - a. [トピック ARN] は、トピックの Amazon リソースネーム (ARN) を選択します。この値は、Amazon SNS トピックの作成時に生成された AWS ARN です。たとえば、`arn:aws:sns:us-east-2:123456789012:your_topic`。
 - b. [プロトコル] でエンドポイントタイプを選択します。使用可能なエンドポイントタイプは次のとおりです。

- [HTTP/HTTPS](#)
- [Email/Email-JSON](#)
-
- [Amazon SQS](#)

Note

[SNS FIFO トピック](#)にサブスクライブするには、このオプションを選択します。

- [AWS Lambda](#)
 - [プラットフォームアプリケーションエンドポイント](#)
 - [SMS](#)
- c. [エンドポイント] に、エンドポイント値 (E メールアドレスや Amazon SQS キューの ARN など) を入力します。
 - d. Firehose エンドポイントのみ: [サブスクリプションロール ARN] で、Firehose 配信ストリームに書き込むために作成した IAM ロールの ARN を指定します。詳細については、

「[Firehose 配信ストリームを Amazon SNS トピックにサブスクライブするための前提条件](#)」を参照してください。

- e. (オプション) Firehose、Amazon SQS、HTTP/S エンドポイントは、raw メッセージ配信を有効にすることもできます。詳細については、「[Amazon SNS raw メッセージの配信](#)」を参照してください。
- f. (オプション) フィルターポリシーを設定するには、[サブスクリプションのフィルターポリシー] セクションを展開します。詳細については、「[Amazon SNS サブスクリプションフィルターポリシー](#)」を参照してください。
- g. (オプション) ペイロードベースのフィルタリングを有効にするには、Filter Policy Scope を MessageBody に設定します。詳細については、「[Amazon SNS サブスクリプションフィルターポリシーの範囲](#)」を参照してください。
- h. (オプション) サブスクリプションのデッドレターキューを設定するには、Redrive ポリシー (デッドレターキュー) を展開します。詳細については、「[Amazon SNS デッドレターキュー](#)」を参照してください。
- i. [サブスクリプションの作成] を選択します。

コンソールがサブスクリプションを作成し、サブスクリプションの [詳細] ページを開きます。

Amazon SNS メッセージの発行

[Amazon SNS トピックを作成](#)して、このトピックに[エンドポイントをサブスクライブ](#)したら、トピックにメッセージを発行できます。メッセージが公開されると、Amazon SNS はサブスクライブされた[エンドポイント](#)にメッセージを配信しようとします。

を使用して Amazon SNS トピックにメッセージを発行するには AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. 左のナビゲーションペインで、[トピック] を選択します。
3. [トピック] ページで、トピックを選択し、[メッセージの発行] を選択します。

コンソールが [トピックにメッセージを発行] ページを開きます。

4. [メッセージ詳細] セクションで、次の操作を行います。
 - a. (オプション) メッセージの件名を入力します。

- b. [FIFO トピック](#)は、メッセージグループ ID を入力します。FIFO トピックにはメッセージグループ ID が必要です。同じメッセージグループ内のメッセージは、発行された順序で配信されます。
 - c. (オプション) 標準トピックの場合は、メッセージグループ ID を入力します。この ID はすべての Amazon SQS 標準サブスクリプションに転送され、他のエンドポイントタイプで使用されたり、他のエンドポイントタイプに送信されたりすることはありません。
 - d. FIFO トピックは、メッセージ重複排除 ID を入力します。トピックの[コンテンツベースのメッセージ重複除外] 設定を有効にしていれば、この ID は任意です。
 - e. (オプション) [モバイルプッシュ通知](#)に、有効期限 (TTL) 値を秒単位で入力します。この属性で、Apple Push Notification Service (APNs) や Firebase Cloud Messaging (FCM) などのプッシュ通知サービスによってエンドポイントにメッセージが配信される時間を指定できます。
5. [メッセージ本文] セクションで、以下のいずれかの操作を行います。
- a. [すべての配信プロトコルに同一のペイロード] を選択し、メッセージを入力します。
 - b. [各配信プロトコルのカスタムペイロード] を選択し、JSON オブジェクトを入力し、各プロトコルに送信するメッセージを定義します。
- 詳細については、「[プラットフォーム固有のペイロードを使用した Amazon SNS 通知の発行](#)」を参照してください。
6. [メッセージの属性] セクションで、属性を追加します。Amazon SNS は、追加された属性とサブスクリプション属性 FilterPolicy をマッチングし、サブスクライブされたエンドポイントが、発行されたメッセージに関心があるかどうかを判断します。
- a. タイプでは、String.Array などの属性タイプを選択します。

 Note

属性のタイプが String.Array である場合は、配列を角括弧 ([]) で囲みます。配列内で、値の文字列を二重引用符で囲みます。数字またはキーワード true、false、null を引用符で囲む必要はありません。

- b. customer_interests などの属性の名前を入力します。
- c. ["soccer", "rugby", "hockey"] などの属性の値を入力します。

属性のタイプが、String、String.Array、または Number である場合、フィルターポリシー (存在する場合) の範囲が明示的に MessageBody に設定されていなければ、Amazon SNS はサブスクリプションにメッセージを送信する前に、サブスクリプションの [フィルターポリシー](#) に照らしてメッセージ属性を評価します。

詳細については、「[Amazon SNS メッセージ属性](#)」を参照してください。

7. [メッセージの発行] を選択します。

トピックにメッセージが発行され、コンソールがトピックの [詳細] ページを開きます。

AWS SDK を使用してトピックにメッセージを発行するには

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、「AWS SDK とツールのリファレンスガイド」の「[共有設定ファイルと認証情報ファイル](#)」を参照してください。

次のサンプルコードは、Publish を使用する方法を説明しています。

.NET

SDK for .NET

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。

トピックへのメッセージの発行

```
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example publishes a message to an Amazon Simple Notification
/// Service (Amazon SNS) topic.
/// </summary>
```

```
public class PublishToSNSTopic
{
    public static async Task Main()
    {
        string topicArn = "arn:aws:sns:us-
east-2:000000000000:ExampleSNSTopic";
        string messageText = "This is an example message to publish to the
ExampleSNSTopic.";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await PublishToTopicAsync(client, topicArn, messageText);
    }

    /// <summary>
    /// Publishes a message to an Amazon SNS topic.
    /// </summary>
    /// <param name="client">The initialized client object used to publish
    /// to the Amazon SNS topic.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="messageText">The text of the message.</param>
    public static async Task PublishToTopicAsync(
        IAmazonSimpleNotificationService client,
        string topicArn,
        string messageText)
    {
        var request = new PublishRequest
        {
            TopicArn = topicArn,
            Message = messageText,
        };

        var response = await client.PublishAsync(request);

        Console.WriteLine($"Successfully published message ID:
{response.MessageId}");
    }
}
```

グループ、重複、属性のオプションを指定してトピックにメッセージを発行します。

```
/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                             "\r\nAll messages within the same group will be
received in the order " +
                             "they were published.");

            Console.WriteLine();
            var messageId = GetUserResponse("Enter a message group ID
for this message:", "1");

            if (!_useContentBasedDeduplication)
            {
                Console.WriteLine("Because you are not using content-based
deduplication, " +
                                 "you must enter a deduplication ID.");

                Console.WriteLine("Enter a deduplication ID for this
message.");
                deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
            }

            if (GetYesNoResponse("Add an attribute to this message?"))
            {
```

```
        Console.WriteLine("Enter a number for an attribute.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }
    }

    var messageID = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
        messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

keepSendingMessages = GetYesNoResponse("Send another message?",
false);
}
}
```

ユーザーの選択を発行アクションに適用します。

```
/// <summary>
/// Publish a message to a topic with an attribute and optional deduplication
and group IDs.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="message">The message to publish.</param>
/// <param name="attributeName">The optional attribute for the message.</
param>
/// <param name="attributeValue">The optional attribute value for the
message.</param>
/// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
/// <param name="groupId">The optional group ID for the message.</param>
```



```
/// <returns>The ID of the message published.</returns>
public async Task<string> PublishToTopicWithAttribute(
    string topicArn,
    string message,
    string? attributeName = null,
    string? attributeValue = null,
    string? deduplicationId = null,
    string? groupId = null)
{
    var publishRequest = new PublishRequest()
    {
        TopicArn = topicArn,
        Message = message,
        MessageDeduplicationId = deduplicationId,
        MessageGroupId = groupId
    };

    if (attributeValue != null)
    {
        // Add the string attribute if it exists.
        publishRequest.MessageAttributes =
            new Dictionary<string, MessageAttributeValue>
            {
                { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
            };
    }

    var publishResponse = await
        _amazonSNSClient.PublishAsync(publishRequest);
    return publishResponse.MessageId;
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[発行](#)」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。

```
//! Send a message to an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
    \param message: The message to publish.
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::publishToTopic(const Aws::String &message,
                                const Aws::String &topicARN,
                                const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::PublishRequest request;
    request.SetMessage(message);
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Message published successfully with id '"
                    << outcome.GetResult().GetMessageId() << "'." << std::endl;
    }
    else {
        std::cerr << "Error while publishing message "
                    << outcome.GetError().GetMessage()
                    << std::endl;
    }

    return outcome.IsSuccess();
}
```

属性を指定してメッセージを発行します。

```
static const Aws::String TONE_ATTRIBUTE("tone");
static const Aws::Vector<Aws::String> TONES = {"cheerful", "funny",
"serious",
"sincere"};

Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::SNS::SNSClient snsClient(clientConfiguration);

Aws::SNS::Model::PublishRequest request;
request.SetTopicArn(topicARN);
Aws::String message = askQuestion("Enter a message text to publish. ");
request.SetMessage(message);

if (filteringMessages && askYesNoQuestion(
    "Add an attribute to this message? (y/n) ")) {
    for (size_t i = 0; i < TONES.size(); ++i) {
        std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
    }
    int selection = askQuestionForIntRange(
        "Enter a number for an attribute. ",
        1, static_cast<int>(TONES.size()));
    Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
    messageAttributeValue.SetDataType("String");
    messageAttributeValue.SetStringValue(TONES[selection - 1]);
    request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
}

Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

if (outcome.IsSuccess()) {
    std::cout << "Your message was successfully published." << std::endl;
}
else {
    std::cerr << "Error with TopicsAndQueues::Publish. "
        << outcome.GetError().GetMessage()
        << std::endl;

    cleanUp(topicARN,
        queueURLS,
```

```
        subscriptionARNS,  
        snsClient,  
        sqsClient);  
  
    return false;  
}
```

- API の詳細については、AWS SDK for C++ API リファレンスの「[Publish](#)」を参照してください。

CLI

AWS CLI

例 1: トピックにメッセージを発行するには

次の publish の例では、指定した Amazon SNS トピックに指定した通知を公開します。メッセージはテキストファイルから取得されたもので、改行を含めることができます。

```
aws sns publish \  
  --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic" \  
  --message file://message.txt
```

message.txt の内容:

```
Hello World  
Second Line
```

出力:

```
{  
  "MessageId": "123a45b6-7890-12c3-45d6-111122223333"  
}
```

例 2: 電話番号に SMS メッセージを公開するには

次の publish の例では、Hello world! メッセージを電話番号 +1-555-555-0100 に公開します。

```
aws sns publish \  
  --message "Hello world!" \  
  --phone-number +1-555-555-0100
```

出力:

```
{  
  "MessageId": "123a45b6-7890-12c3-45d6-333322221111"  
}
```

- APIの詳細については、「AWS CLI コマンドリファレンス」の「[Publish](#)」を参照してください。

Go

SDK for Go V2

 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 (  
  "context"  
  "encoding/json"  
  "log"  
  
  "github.com/aws/aws-sdk-go-v2/aws"  
  "github.com/aws/aws-sdk-go-v2/service/sns"  
  "github.com/aws/aws-sdk-go-v2/service/sns/types"  
)  
  
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)  
// actions  
// used in the examples.  
type SnsActions struct {  
  SnsClient *sns.Client  
}
```

```
// Publish publishes a message to an Amazon SNS topic. The message is then sent
// to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
// group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
// value
// filter attribute can be specified so that the message can be filtered
// according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
string, groupId string, dedupId string, filterKey string, filterValue string)
error {
    publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
aws.String(message)}
    if groupId != "" {
        publishInput.MessageGroupId = aws.String(groupId)
    }
    if dedupId != "" {
        publishInput.MessageDeduplicationId = aws.String(dedupId)
    }
    if filterKey != "" && filterValue != "" {
        publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
            filterKey: {DataType: aws.String("String"), StringValue:
aws.String(filterValue)},
        }
    }
    _, err := actor.SnsClient.Publish(ctx, &publishInput)
    if err != nil {
        log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
err)
    }
    return err
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[Publish](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class PublishTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <message> <topicArn>

            Where:
                message - The message text to send.
                topicArn - The ARN of the topic to publish.
            "";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String message = args[0];
```

```
String topicArn = args[1];
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();
pubTopic(snsClient, message, topicArn);
snsClient.close();
}

public static void pubTopic(SnsClient snsClient, String message, String
topicArn) {
    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .topicArn(topicArn)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out
            .println(result.messageId() + " Message sent. Status is " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[Publish](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
plain string or an object
 *
 * if you are using the `json`
`MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  // }
```

```
// MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'  
// }  
return response;  
};
```

グループ、重複、属性のオプションを指定してトピックにメッセージを発行します。

```
async publishMessages() {  
  const message = await this.prompter.input({  
    message: MESSAGES.publishMessagePrompt,  
  });  
  
  let groupId;  
  let deduplicationId;  
  let choices;  
  
  if (this.isFifo) {  
    await this.logger.log(MESSAGES.groupIdNotice);  
    groupId = await this.prompter.input({  
      message: MESSAGES.groupIdPrompt,  
    });  
  
    if (this.autoDedup === false) {  
      await this.logger.log(MESSAGES.deduplicationIdNotice);  
      deduplicationId = await this.prompter.input({  
        message: MESSAGES.deduplicationIdPrompt,  
      });  
    }  
  
    choices = await this.prompter.checkbox({  
      message: MESSAGES.messageAttributesPrompt,  
      choices: toneChoices,  
    });  
  }  
  
  await this.snsClient.send(  
    new PublishCommand({  
      TopicArn: this.topicArn,  
      Message: message,  
      ...(groupId  
        ? {  
          MessageGroupId: groupId,  

```

```
        }
        : {}),
    ...(deduplicationId
    ? {
        MessageDeduplicationId: deduplicationId,
    }
    : {}),
    ...(choices
    ? {
        MessageAttributes: {
            tone: {
                DataType: "String.Array",
                StringValue: JSON.stringify(choices),
            },
        },
    }
    : {}),
    )),
);

const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
});

if (publishAnother) {
    await this.publishMessages();
}
}
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[Publish](#)」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 pubTopic(
    topicArnVal: String,
    messageVal: String,
) {
    val request =
        PublishRequest {
            message = messageVal
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[Publish](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Sends a message to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->publish([
        'Message' => $message,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[Publish](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: この例では、インラインで宣言された単一の MessageAttribute を使用してメッセージを発行しています。

```
Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute
@{'City'=[Amazon.SimpleNotificationService.Model.MessageAttributeValue]@{DataType='String';
StringValue = 'AnyCity'}}}
```

例 2: この例では、事前に宣言された複数の MessageAttribute を使用してメッセージを発行しています。

```
$cityAttributeValue = New-Object
    Amazon.SimpleNotificationService.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object
    Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute $messageAttributes
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[Publish](#)」を参照してください。

Tools for PowerShell V5

例 1: この例では、インラインで宣言された単一の MessageAttribute を使用してメッセージを発行しています。

```
Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute
```

```
@{'City'=[Amazon.SimpleNotificationService.Model.MessageAttributeValue]@{'DataType='String',
StringValue = 'AnyCity'}}}
```

例 2: この例では、事前に宣言された複数の MessageAttribute を使用してメッセージを発行しています。

```
$cityAttributeValue = New-Object
    Amazon.SimpleNotificationService.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object
    Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
    Message "Hello" -MessageAttribute $messageAttributes
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V5)」の「[Publish](#)」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。

サブスクリプションが属性に基づいてフィルター処理できるように、属性を含むメッセージを発行します。

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
```

```

def __init__(self, sns_resource):
    """
    :param sns_resource: A Boto3 Amazon SNS resource.
    """
    self.sns_resource = sns_resource

    @staticmethod
    def publish_message(topic, message, attributes):
        """
        Publishes a message, with attributes, to a topic. Subscriptions can be
        filtered
        based on message attributes so that a subscription receives messages only
        when specified attributes are present.

        :param topic: The topic to publish to.
        :param message: The message to publish.
        :param attributes: The key-value attributes to attach to the message.
        Values
                           must be either `str` or `bytes`.
        :return: The ID of the message.
        """
        try:
            att_dict = {}
            for key, value in attributes.items():
                if isinstance(value, str):
                    att_dict[key] = {"DataType": "String", "StringValue": value}
                elif isinstance(value, bytes):
                    att_dict[key] = {"DataType": "Binary", "BinaryValue": value}
            response = topic.publish(Message=message, MessageAttributes=att_dict)
            message_id = response["MessageId"]
            logger.info(
                "Published message with attributes %s to topic %s.",
                attributes,
                topic.arn,
            )
        except ClientError:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise
        else:
            return message_id

```


サブスクライバーのプロトコルに基づいて異なる形式のメッセージを発行します。

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def publish_multi_message(
        topic, subject, default_message, sms_message, email_message
    ):
        """
        Publishes a multi-format message to a topic. A multi-format message takes
        different forms based on the protocol of the subscriber. For example,
        an SMS subscriber might receive a short version of the message
        while an email subscriber could receive a longer version.

        :param topic: The topic to publish to.
        :param subject: The subject of the message.
        :param default_message: The default version of the message. This version
        is
                                sent to subscribers that have protocols that are
        not
                                otherwise specified in the structured message.
        :param sms_message: The version of the message sent to SMS subscribers.
        :param email_message: The version of the message sent to email
        subscribers.
        :return: The ID of the message.
        """
        try:
            message = {
                "default": default_message,
                "sms": sms_message,
                "email": email_message,
            }
            response = topic.publish(
```

```

        Message=json.dumps(message), Subject=subject,
        MessageStructure="json"
    )
    message_id = response["MessageId"]
    logger.info("Published multi-format message to topic %s.", topic.arn)
except ClientError:
    logger.exception("Couldn't publish message to topic %s.", topic.arn)
    raise
else:
    return message_id

```

```

class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def publish_message(
        self,
        topic_arn: str,
        message: str,
        tone_attribute: Optional[str] = None,
        deduplication_id: Optional[str] = None,
        message_group_id: Optional[str] = None
    )

```

```
) -> str:
    """
    Publish a message to an SNS topic.

    :param topic_arn: The ARN of the SNS topic.
    :param message: The message content to publish.
    :param tone_attribute: Optional tone attribute for message filtering.
    :param deduplication_id: Optional deduplication ID for FIFO topics.
    :param message_group_id: Optional message group ID for FIFO topics.
    :return: The message ID of the published message.
    :raises ClientError: If the message publication fails.
    """
    try:
        publish_args = {
            'TopicArn': topic_arn,
            'Message': message
        }

        # Add message attributes if tone is specified
        if tone_attribute:
            publish_args['MessageAttributes'] = {
                'tone': {
                    'DataType': 'String',
                    'StringValue': tone_attribute
                }
            }

        # Add FIFO-specific parameters
        if message_group_id:
            publish_args['MessageGroupId'] = message_group_id

        if deduplication_id:
            publish_args['MessageDeduplicationId'] = deduplication_id

        response = self.sns_client.publish(**publish_args)

        message_id = response['MessageId']
        logger.info(f"Published message to topic {topic_arn} with ID: {message_id}")
        return message_id

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
```

```
        logger.error(f"Error publishing message to topic: {error_code} - {e}")
        raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[Publish](#)」を参照してください。

Ruby

SDK for Ruby

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。

```
# Service class for sending messages using Amazon Simple Notification Service (SNS)
class SnsMessageSender
  # Initializes the SnsMessageSender with an SNS client
  #
  # @param sns_client [Aws::SNS::Client] The SNS client
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Sends a message to a specified SNS topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param message [String] The message to send
  # @return [Boolean] true if message was successfully sent, false otherwise
  def send_message(topic_arn, message)
    @sns_client.publish(topic_arn: topic_arn, message: message)
    @logger.info("Message sent successfully to #{topic_arn}.")
    true
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Error while sending the message: #{e.message}")
  end
end
```

```
        false
      end
    end
  end

  # Example usage:
  if $PROGRAM_NAME == __FILE__
    topic_arn = 'SNS_TOPIC_ARN' # Should be replaced with a real topic ARN
    message = 'MESSAGE'         # Should be replaced with the actual message
    content

    sns_client = Aws::SNS::Client.new
    message_sender = SnsMessageSender.new(sns_client)

    @logger.info('Sending message.')
    unless message_sender.send_message(topic_arn, message)
      @logger.error('Message sending failed. Stopping program.')
      exit 1
    end
  end
end
```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for Ruby API リファレンスの「[発行](#)」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 subscribe_and_publish(
  client: &Client,
  topic_arn: &str,
  email_address: &str,
) -> Result<(), Error> {
  println!("Receiving on topic with ARN: `{}`", topic_arn);
```

```
let rsp = client
    .subscribe()
    .topic_arn(topic_arn)
    .protocol("email")
    .endpoint(email_address)
    .send()
    .await?;

println!("Added a subscription: {:?}", rsp);

let rsp = client
    .publish()
    .topic_arn(topic_arn)
    .message("hello sns!")
    .send()
    .await?;

println!("Published message: {:?}", rsp);

Ok(())
}
```

- APIの詳細については、AWS SDK for Rust API リファレンスの「[発行](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.
    oo_result = lo_sns->publish(
testing purposes. " " oo_result is returned for
    iv_topicarn = iv_topic_arn
```

```

        iv_message = iv_message ).
    MESSAGE 'Message published to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.

```

属性を含むメッセージをトピックに発行します。

```

TRY.
    oo_result = lo_sns->publish(
testing purposes. " oo_result is returned for
        iv_topicarn = iv_topic_arn
        iv_message = iv_message
        it_messageattributes = it_msg_attrs ).
    MESSAGE 'Message with attributes published to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.

```

マルチフォーマットメッセージをトピックに発行します。

```

" Build JSON message structure for multi-format message
DATA(lv_json_message) = |\{ "default": "{ iv_default_message }", "sms":
"{ iv_sms_message }", "email": "{ iv_email_message }" \}|.

TRY.
    oo_result = lo_sns->publish(
testing purposes. " oo_result is returned for
        iv_topicarn = iv_topic_arn
        iv_message = lv_json_message
        iv_subject = iv_subject
        iv_messagestructure = 'json' ).
    MESSAGE 'Multi-format message published to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.

```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの「[Publish](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

let output = try await snsClient.publish(
    input: PublishInput(
        message: message,
        topicArn: arn
    )
)

guard let messageId = output.messageId else {
    print("No message ID received from Amazon SNS.")
    return
}

print("Published message with ID \(messageId)")
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[Publish](#)」を参照してください。

Amazon SNS および Amazon S3 を使用した容量の大きなメッセージを発行する

容量の大きな Amazon SNS メッセージを発行するには、[Java 用 Amazon SNS 拡張クライアントライブラリ](#)または [Python 用 Amazon SNS 拡張クライアントライブラリ](#)を使用できます。このライブラリは、現在の最大容量 256 KB を超える、最大 2 GB までのメッセージに便利です。両方のライブラリは、実際のペイロードを Amazon S3 バケットに保存し、保存された Amazon S3 オブジェク

トの参照をトピックに発行します。サブスクライブした Amazon SQS キューは、[Java 用 Amazon SQS 拡張クライアントライブラリ](#)を使用し、Amazon S3 からペイロードを逆参照して取得します。Lambda などの他のエンドポイントは、[Payload Offloading Java Common Library for AWS](#) を使用し、ペイロードを逆参照して取得します。

Note

Amazon SNS 拡張クライアントライブラリは、標準トピックと FIFO トピックの両方と互換性があります。

Java 用の Amazon SNS 拡張クライアントライブラリ

前提条件

[Java 用 Amazon SNS 拡張クライアントライブラリ](#)を使用するための前提条件は以下のとおりです。

- AWS SDK。このページの例では、Java SDK AWS を使用しています。SDK をインストールしてセットアップするには、「AWS SDK for Java デベロッパーガイド」の [AWS 「SDK for Java のセットアップ」](#) を参照してください。
- 適切な認証情報 AWS アカウント を持つ。を作成するには AWS アカウント、[AWS ホームページ](#)に移動し、AWS アカウントの作成を選択します。手順に従います。

認証情報の詳細については、「AWS SDK for Java デベロッパーガイド」の [「開発用の AWS 認証情報とリージョンの設定」](#) を参照してください。

- Java 8 以上
- Java 用 Amazon SNS 拡張クライアントライブラリ ([Maven](#) から利用可能)

メッセージストレージの設定

Amazon SNS 拡張クライアントライブラリは、メッセージの保存と取得 AWS に のペイロードオフロード Java 共通ライブラリを使用します。以下の Amazon S3 [メッセージストレージオプション](#)を設定できます。

- カスタムメッセージサイズのしきい値 – このサイズを超えるペイロードと属性を持つメッセージは、自動的に Amazon S3 に保存されます。
- **alwaysThroughS3** フラグ – この値を true に設定すると、すべてのメッセージペイロードが強制的に Amazon S3 に保存されます。例:

```
SNSExtendedClientConfiguration snsExtendedClientConfiguration = new
SNSExtendedClientConfiguration().withPayloadSupportEnabled(s3Client,
    BUCKET_NAME).withAlwaysThroughS3(true);
```

- カスタム KMS キー – Amazon S3 バケットのサーバー側の暗号化に使用するキー。
- バケット名 – メッセージペイロードを保存するための Amazon S3 バケットの名前。

例: Amazon S3 に保存されたペイロードを使用して Amazon SNS にメッセージを発行する

次のコード例は、以下の操作方法を示しています。

- サンプルのトピックとキューを作成します。
- トピックからメッセージを受信するためにキューをサブスクライブします。
- テストメッセージを発行します。

メッセージペイロードは Amazon S3 に保存され、そのペイロードへのリファレンスが発行されます。メッセージの受信には、Amazon SQS 拡張クライアントが使用されます。

SDK for Java 1.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。

容量の大きなメッセージを発行するには、Amazon SNS Extended Client Library for Java を使用します。送信するメッセージは、実際のメッセージコンテンツを含む Amazon S3 オブジェクトをリファレンスします。

```
import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
```

```
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.PublishRequest;
import com.amazonaws.services.sns.model.SetSubscriptionAttributesRequest;
import com.amazonaws.services.sns.util.Topics;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
import software.amazon.sns.AmazonSNSExtendedClient;
import software.amazon.sns.SNSExtendedClientConfiguration;

public class Example {

    public static void main(String[] args) {
        final String BUCKET_NAME = "extended-client-bucket";
        final String TOPIC_NAME = "extended-client-topic";
        final String QUEUE_NAME = "extended-client-queue";
        final Regions region = Regions.DEFAULT_REGION;

        // Message threshold controls the maximum message size that will be
allowed to
        // be published
        // through SNS using the extended client. Payload of messages
exceeding this
        // value will be stored in
        // S3. The default value of this parameter is 256 KB which is the
maximum

        // message size in SNS (and SQS).
        final int EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD = 32;

        // Initialize SNS, SQS and S3 clients
        final AmazonSNS snsClient =
AmazonSNSClientBuilder.standard().withRegion(region).build();
        final AmazonSQS sqsClient =
AmazonSQSClientBuilder.standard().withRegion(region).build();
        final AmazonS3 s3Client =
AmazonS3ClientBuilder.standard().withRegion(region).build();

        // Create bucket, topic, queue and subscription
        s3Client.createBucket(BUCKET_NAME);
        final String topicArn = snsClient.createTopic(
            new
CreateTopicRequest().withName(TOPIC_NAME)).getTopicArn();
        final String queueUrl = sqsClient.createQueue(
```

```

        new
CreateQueueRequest().withQueueName(QUEUE_NAME)).getQueueUrl();
        final String subscriptionArn = Topics.subscribeQueue(
            snsClient, sqsClient, topicArn, queueUrl);

        // To read message content stored in S3 transparently through SQS
extended
        // client,
        // set the RawMessageDelivery subscription attribute to TRUE
        final SetSubscriptionAttributesRequest subscriptionAttributesRequest
= new SetSubscriptionAttributesRequest();
        subscriptionAttributesRequest.setSubscriptionArn(subscriptionArn);

subscriptionAttributesRequest.setAttributeName("RawMessageDelivery");
subscriptionAttributesRequest.setAttributeValue("TRUE");
snsClient.setSubscriptionAttributes(subscriptionAttributesRequest);

        // Initialize SNS extended client
        // PayloadSizeThreshold triggers message content storage in S3 when
the
        // threshold is exceeded
        // To store all messages content in S3, use AlwaysThroughS3 flag
        final SNSExtendedClientConfiguration snsExtendedClientConfiguration
= new SNSExtendedClientConfiguration()
            .withPayloadSupportEnabled(s3Client, BUCKET_NAME)

            .withPayloadSizeThreshold(EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD);
        final AmazonSNSExtendedClient snsExtendedClient = new
AmazonSNSExtendedClient(snsClient,
            snsExtendedClientConfiguration);

        // Publish message via SNS with storage in S3
        final String message = "This message is stored in S3 as it exceeds
the threshold of 32 bytes set above.";
        snsExtendedClient.publish(topicArn, message);

        // Initialize SQS extended client
        final ExtendedClientConfiguration sqsExtendedClientConfiguration =
new ExtendedClientConfiguration()
            .withPayloadSupportEnabled(s3Client, BUCKET_NAME);
        final AmazonSQSEntendedClient sqsExtendedClient = new
AmazonSQSEntendedClient(sqsClient,
            sqsExtendedClientConfiguration);

```

```
        // Read the message from the queue
        final ReceiveMessageResult result =
            sqsExtendedClient.receiveMessage(queueUrl);
        System.out.println("Received message is " +
            result.getMessages().get(0).getBody());
    }
}
```

その他のエンドポイントプロトコル

Amazon SNS と Amazon SQS ライブラリの両方で、[Payload Offloading Java Common Library for AWS](#) を使用して、Amazon S3 でメッセージペイロードを保存および取得できます。Java が有効なエンドポイント (Java で実装されている HTTPS エンドポイントなど) は、同じライブラリを使用してメッセージコンテンツを逆リファレンスできます。

のペイロードオフロード Java 共通ライブラリを使用できないエンドポイントでも、Amazon S3 に保存されているペイロードを使用してメッセージを発行 AWS できます。以下は、上記のコード例で発行された Amazon S3 リファレンスの例です。

```
[
    "software.amazon.payloadoffloading.PayloadS3Pointer",
    {
        "s3BucketName": "extended-client-bucket",
        "s3Key": "xxxx-xxxxxx-xxxxxx-xxxxxx"
    }
]
```

Python 用の Amazon SNS 拡張クライアントライブラリ

前提条件

[Python 用 Amazon SNS 拡張クライアントライブラリ](#)を使用するための前提条件は以下のとおりです。

- AWS SDK。このページの例では、Python SDK Boto3 AWS を使用しています。SDK をインストールして設定するには、[AWS SDK for Python](#) のドキュメントを参照してください。
- 適切な認証情報 AWS アカウント を持つ。を作成するには AWS アカウント、[AWS ホームページ](#)に移動し、AWS アカウントの作成を選択します。手順に従います。

認証情報については、「AWS SDK for Python デベロッパーガイド」の「[認証情報](#)」を参照してください。

- Python 3.x (または以降) および pip。
- Python 用 Amazon SNS 拡張クライアントライブラリ ([PyPI](#) から利用可能)

メッセージストレージの設定

Amazon S3 のメッセージ保存オプションを設定するには、Boto3 Amazon SNS の [Client](#)、[Topic](#)、[PlatformEndpoint](#) の各オブジェクトで以下の属性を使用できます。

- **large_payload_support** – サイズの大きいメッセージを保存する Amazon S3 バケットの名前。
- **use_legacy_attribute** – True の場合、すべての発行済みメッセージは、現在の予約済みメッセージ属性 (ExtendedPayloadSize) ではなく、レガシーの予約済みメッセージ属性 (SQSLargePayloadSize) を使用します。
- **message_size_threshold** – メッセージを大きなメッセージバケットに保存するためのしきい値。0 未満または 262144 を超える値を指定することはできません。デフォルトは 262144 です。
- **always_through_s3** – True の場合、すべてのメッセージは Amazon S3 に保存されます。デフォルトは False です。
- **s3_client** – Amazon S3 にオブジェクトを保存するために使用する Boto3 Amazon S3 の client オブジェクト。Amazon S3 クライアント (Amazon S3 のカスタム設定や認証情報など) を制御する場合に使用します。以前に設定していない場合は、最初の使用時にデフォルトで `boto3.client("s3")` に設定されます。

例: Amazon S3 にペイロードが保存されている Amazon SNS にメッセージを発行する

次のコード例は、以下の操作方法を示しています。

- Amazon SNS トピックと Amazon SQS キューを作成します。
- ポリシーを Amazon SQS キューにアタッチして、Amazon SNS トピックからメッセージを受信します。
- トピックからメッセージを受信するためにキューをサブスクライブします。
- Amazon SNS 拡張クライアント、トピックリソース、および PlatformEndpoint リソースを使用してテストメッセージを発行します。

- メッセージペイロードは Amazon S3 に保存され、その参照が発行されます。
- キューから発行したメッセージと、Amazon S3 から取得した元のメッセージを印刷します。

容量の大きなメッセージを発行するには、Java 用 Amazon SNS 拡張クライアントライブラリを使用します。送信するメッセージは、実際のメッセージコンテンツが含まれている Amazon S3 オブジェクトを参照します。

```
import boto3
from sns_extended_client import SNSExtendedClientSession
from json import loads

s3_extended_payload_bucket = "extended-client-bucket-store" # S3 bucket with the
given bucket name is a resource which is created and accessible with the given AWS
credentials
TOPIC_NAME = "---TOPIC-NAME---"
QUEUE_NAME = "---QUEUE-NAME---"

def allow_sns_to_write_to_sqs(topicarn, queuearn):
    policy_document = """{{
        "Version": "2012-10-17",
        "Statement": [
            {{
                "Sid": "MyPolicy",
                "Effect": "Allow",
                "Principal" : {{ "AWS" : "*" }},
                "Action": "SQS:SendMessage",
                "Resource": "{}",
                "Condition": {{
                    "ArnEquals": {{
                        "aws:SourceArn": "{}"
                    }}
                }}
            }}
        ]
    }}""".format(queuearn, topicarn)

    return policy_document

def get_msg_from_s3(body, sns_extended_client):
    """Handy Helper to fetch message from S3"""
    json_msg = loads(body)
    s3_object = sns_extended_client.s3_client.get_object(
```

```

        Bucket=json_msg[1].get("s3BucketName"), Key=json_msg[1].get("s3Key")
    )
    msg = s3_object.get("Body").read().decode()
    return msg

def fetch_and_print_from_sqs(sqs, queue_url,sns_extended_client):
    sqs_msg = sqs.receive_message(
        QueueUrl=queue_url,
        AttributeNames=['All'],
        MessageAttributeNames=['All'],
        VisibilityTimeout=0,
        WaitTimeSeconds=0,
        MaxNumberOfMessages=1
    ).get("Messages")[0]

    message_body = sqs_msg.get("Body")
    print("Published Message: {}".format(message_body))
    print("Message Stored in S3 Bucket is:
    {}\n".format(get_msg_from_s3(message_body,sns_extended_client)))

    # Delete the Processed Message
    sqs.delete_message(
        QueueUrl=queue_url,
        ReceiptHandle=sqs_msg['ReceiptHandle']
    )

sns_extended_client = boto3.client("sns", region_name="us-east-1")
create_topic_response = sns_extended_client.create_topic(Name=TOPIC_NAME)
sns_topic_arn = create_topic_response.get("TopicArn")

# create and subscribe an sqs queue to the sns client
sqs = boto3.client("sqs",region_name="us-east-1")
demo_queue_url = sqs.create_queue(QueueName=QUEUE_NAME).get("QueueUrl")
sqs_queue_arn = sqs.get_queue_attributes(
    QueueUrl=demo_queue_url, AttributeNames=["QueueArn"]
)["Attributes"].get("QueueArn")

# Adding policy to SQS queue such that SNS topic can send msg to SQS queue
policy_json = allow_sns_to_write_to_sqs(sns_topic_arn, sqs_queue_arn)
response = sqs.set_queue_attributes(
    QueueUrl = demo_queue_url,
    Attributes = {

```



```
        'Policy' : policy_json
    }
)

# Set the RawMessageDelivery subscription attribute to TRUE if you want to use
# SQSExtendedClient to help with retrieving msg from S3
sns_extended_client.subscribe(TopicArn=sns_topic_arn, Protocol="sqs",
    Endpoint=sqs_queue_arn
    , Attributes={"RawMessageDelivery":"true"})

sns_extended_client.large_payload_support = s3_extended_payload_bucket

# Change default s3_client attribute of sns_extended_client to use 'us-east-1' region
sns_extended_client.s3_client = boto3.client("s3", region_name="us-east-1")

# Below is the example that all the messages will be sent to the S3 bucket
sns_extended_client.always_through_s3 = True
sns_extended_client.publish(
    TopicArn=sns_topic_arn, Message="This message should be published to S3"
)
print("\n\nPublished using SNS extended client:")
fetch_and_print_from_sqs(sqs, demo_queue_url,sns_extended_client) # Prints message
    stored in s3

# Below is the example that all the messages larger than 32 bytes will be sent to the
    S3 bucket
print("\nUsing decreased message size threshold:")

sns_extended_client.always_through_s3 = False
sns_extended_client.message_size_threshold = 32
sns_extended_client.publish(
    TopicArn=sns_topic_arn,
    Message="This message should be published to S3 as it exceeds the limit of the 32
    bytes",
)

fetch_and_print_from_sqs(sqs, demo_queue_url,sns_extended_client) # Prints message
    stored in s3

# Below is the example to publish message using the SNS.Topic resource
sns_extended_client_resource = SNSExtendedClientSession().resource(
```

```
    "sns", region_name="us-east-1"
)

topic = sns_extended_client_resource.Topic(sns_topic_arn)
topic.large_payload_support = s3_extended_payload_bucket

# Change default s3_client attribute of topic to use 'us-east-1' region
topic.s3_client = boto3.client("s3", region_name="us-east-1")

topic.always_through_s3 = True
# Can Set custom S3 Keys to be used to store objects in S3
topic.publish(
    Message="This message should be published to S3 using the topic resource",
    MessageAttributes={
        "S3Key": {
            "DataType": "String",
            "StringValue": "347c11c4-a22c-42e4-a6a2-9b5af5b76587",
        }
    },
)
print("\nPublished using Topic Resource:")
fetch_and_print_from_sqs(sqs, demo_queue_url, topic)

# Below is the example to publish message using the SNS.PlatformEndpoint resource
sns_extended_client_resource = SNSExtendedClientSession().resource(
    "sns", region_name="us-east-1"
)

platform_endpoint = sns_extended_client_resource.PlatformEndpoint(sns_topic_arn)
platform_endpoint.large_payload_support = s3_extended_payload_bucket

# Change default s3_client attribute of platform_endpoint to use 'us-east-1' region
platform_endpoint.s3_client = boto3.client("s3", region_name="us-east-1")

platform_endpoint.always_through_s3 = True
# Can Set custom S3 Keys to be used to store objects in S3
platform_endpoint.publish(
    Message="This message should be published to S3 using the PlatformEndpoint
resource",
    MessageAttributes={
        "S3Key": {
            "DataType": "String",
            "StringValue": "247c11c4-a22c-42e4-a6a2-9b5af5b76587",
        }
    },
)
```

```
    },
)
print("\nPublished using PlatformEndpoint Resource:")
fetch_and_print_from_sqs(sqs, demo_queue_url, platform_endpoint)
```

出力

```
Published using SNS extended client:
Published Message: ["software.amazon.payloadoffloading.PayloadS3Pointer",
{"s3BucketName": "extended-client-bucket-store", "s3Key": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"}]
Message Stored in S3 Bucket is: This message should be published to S3

Using decreased message size threshold:
Published Message: ["software.amazon.payloadoffloading.PayloadS3Pointer",
{"s3BucketName": "extended-client-bucket-store", "s3Key": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"}]
Message Stored in S3 Bucket is: This message should be published to S3 as it exceeds
the limit of the 32 bytes

Published using Topic Resource:
Published Message: ["software.amazon.payloadoffloading.PayloadS3Pointer",
{"s3BucketName": "extended-client-bucket-store", "s3Key": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"}]
Message Stored in S3 Bucket is: This message should be published to S3 using the topic
resource

Published using PlatformEndpoint Resource:
Published Message: ["software.amazon.payloadoffloading.PayloadS3Pointer",
{"s3BucketName": "extended-client-bucket-store", "s3Key": "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx"}]
Message Stored in S3 Bucket is: This message should be published to S3 using the
PlatformEndpoint resource
```

Amazon SNS メッセージ属性

Amazon SNS では、メッセージに関する構造化メタデータ項目 (タイムスタンプ、地理空間データ、署名、識別子など) を指定できるメッセージ属性の配信をサポートしています。SQS サブスクリプションの場合、[raw メッセージの配信](#)を有効にすると、最大 10 個のメッセージ属性を送信できます。10 個を超えるメッセージ属性を送信するには、Raw メッセージの配信を無効にする必要があります。raw メッセージ配信が有効になっている Amazon SQS サブスクリプション宛てのメッセージ属性が 10 個を超えるメッセージは、クライアント側のエラーとして破棄されます。

メッセージ属性はオプションであり、メッセージ本文とは別個のものですが、同時に送信されます。受信者は、この情報を使用して、メッセージ本文を最初に処理する必要なしでメッセージを処理する方法を決定できます。

AWS マネジメントコンソール または [を使用して属性を含むメッセージを送信する方法](#)については AWS SDK for Java、[を使用して Amazon SNS トピックにメッセージを発行するには AWS マネジメントコンソール](#)「」チュートリアルを参照してください。

Note

メッセージ属性は、メッセージ構造が JSON ではなく String である場合にのみ送信されます。

また、モバイルエンドポイント用のプッシュ通知メッセージを構築するためにメッセージ属性を使用することもできます。このシナリオでは、メッセージ属性はプッシュ通知メッセージの構築のみに使用されます。属性は、Amazon SQS エンドポイントにメッセージ属性とともにメッセージを送信する場合とは異なり、エンドポイントには配信されません。

メッセージ属性を使用して、サブスクリプションフィルターポリシーでメッセージをフィルター処理可能にすることもできます。フィルターポリシーは、トピックのサブスクリプションに適用もできます。フィルターポリシーの範囲を `MessageAttributes` (デフォルト) に設定して、フィルターポリシーを適用すると、サブスクリプションは、ポリシーが受け入れる属性を持つメッセージのみを受信します。詳細については、「[Amazon SNS メッセージフィルター処理](#)」を参照してください。

Note

メッセージ属性をフィルタリングに使用する場合、値は有効な JSON 文字列でなければなりません。これにより、メッセージ属性フィルタリングが有効になっているサブスクリプションにメッセージが確実に配信されます。

メッセージ属性の項目および検証

各メッセージ属性は、次の項目で構成されています。

- 名前 - メッセージ属性名には、A-Z、a-z、0-9、下線 (`_`)、ハイフン (`-`)、ピリオド (`.`) を使用できます。名前の先頭と末尾をピリオドにすることはできず、ピリオドを連続して使用することはでき

ません。名前では大文字と小文字が区別され、メッセージのすべての属性名間で一意にする必要があります。名前の長さは最大 256 文字です。名前の先頭を「AWS.」や「Amazon.」(または、大文字と小文字が異なるあらゆる変化形) にすることはできません。これらのプレフィックスは Amazon Web Services で使用するために取り置かれています。

- 型 - サポートされるメッセージ属性のデータ型

は、String、String.Array、Number、Binary です。データ型のコンテンツには、メッセージ本文と同じ制限があります。詳細については「[メッセージ属性のデータ型と検証](#)」セクションを参照してください。

- 値 - ユーザー指定のメッセージ属性値。文字列データ型の場合、値属性はメッセージ本文と同じコンテンツの制限に従う必要があります。ただし、メッセージ属性をフィルタリングに使用する場合、Amazon SNS サブスクリプションフィルターポリシーとの互換性を確保するために、値は有効な JSON 文字列であることが必要です。詳細については、『Amazon Simple Notification Service API リファレンス』の「[公開](#)アクション」を参照してください。

名前、型、値を空または Null にすることはできません。さらに、メッセージ本文を空または Null にすることもできません。メッセージ属性のすべての部分 (名前、型、値を含む) は、メッセージサイズの制限に含まれます。制限は 256 KB です。

メッセージ属性のデータ型と検証

メッセージ属性のデータ型は、メッセージ属性が Amazon SNS によって処理される方法を特定します。例えば、型が数値の場合、Amazon SNS はその属性が数値であることを検証します。

Amazon SNS は、記載されている場合を除き、すべてのエンドポイントについて、次の論理データ型をサポートしています。

- 文字列 - 文字列は、UTF-8 バイナリエンコードされた Unicode です。コードの値のリストについては、http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters を参照してください。

Note

サロゲート値は、メッセージ属性ではサポートされていません。例えば、サロゲート値を使用して絵文字を表すと、次のエラーが発生します: Invalid attribute value was passed in for message attribute.

- String.Array - 複数の値を含むことができる文字列として書式設定される配列。この値は、文字列、数値、またはキーワード (true、false、null) とすることができます。数値型またはブール型の String.Array は引用符を必要としません。複数の String.Array 値はカンマで区切られます。

このデータ型は AWS Lambda サブスクリプションではサポートされていません。このデータ型を Lambda エンドポイントに指定すると、String データ型を、Amazon SNS が Lambda に配信する JSON ペイロードに格納します。

- 数値 - 数値は正または負の整数か、浮動小数点数です。数値には、整数、浮動小数点数、倍精度浮動小数点数が通常サポートするほとんどの値を包含できる十分な範囲と精度があります。数値は $-10^9 \sim 10^9$ までの値とすることができ、小数点以下 5 桁の精度を持ちます。先頭と末尾の 0 は切り捨てられます。

このデータ型は AWS Lambda サブスクリプションではサポートされていません。このデータ型を Lambda エンドポイントに指定すると、String データ型を、Amazon SNS が Lambda に配信する JSON ペイロードに格納します。

- Binary - Binary 型の属性には、圧縮データ、暗号化データ、イメージなど、任意のバイナリデータが保存されます。

モバイルプッシュ通知の予約済みメッセージ属性

次の表は、プッシュ通知メッセージを構築するために使用できるモバイルプッシュ通知サービスの予約済みメッセージ属性の一覧です。

プッシュ通知サービス	予約済みメッセージ属性
ADM	AWS.SNS.MOBILE.ADM.TTL
APNs ¹	AWS.SNS.MOBILE.APNS_MDM.TTL
	AWS.SNS.MOBILE.APNS_MDM_SANDBOX.TTL
	AWS.SNS.MOBILE.APNS_PASSBOOK.TTL
	AWS.SNS.MOBILE.APNS_PASSBOOK_SANDBOX.TTL
	AWS.SNS.MOBILE.APNS_SANDBOX.TTL
	AWS.SNS.MOBILE.APNS_VOIP.TTL
	AWS.SNS.MOBILE.APNS_VOIP_SANDBOX.TTL
	AWS.SNS.MOBILE.APNS_COLLAPSE_ID

プッシュ通知サービス	予約済みメッセージ属性
	<code>AWS.SNS.MOBILE.APNS.PRIORITY</code>
	<code>AWS.SNS.MOBILE.APNS.PUSH_TYPE</code>
	<code>AWS.SNS.MOBILE.APNS.TOPIC</code>
	<code>AWS.SNS.MOBILE.APNS.TTL</code>
Baidu	<code>AWS.SNS.MOBILE.BAIDU.DeployStatus</code>
	<code>AWS.SNS.MOBILE.BAIDU.MessageKey</code>
	<code>AWS.SNS.MOBILE.BAIDU.MessageType</code>
	<code>AWS.SNS.MOBILE.BAIDU.TTL</code>
FCM	<code>AWS.SNS.MOBILE.FCM.TTL</code>
	<code>AWS.SNS.MOBILE.GCM.TTL</code>
macOS	<code>AWS.SNS.MOBILE.MACOS_SANDBOX.TTL</code>
	<code>AWS.SNS.MOBILE.MACOS.TTL</code>
MPNS	<code>AWS.SNS.MOBILE.MPNS.NotificationClass</code>
	<code>AWS.SNS.MOBILE.MPNS.TTL</code>
	<code>AWS.SNS.MOBILE.MPNS.Type</code>
WNS	<code>AWS.SNS.MOBILE.WNS.CachePolicy</code>
	<code>AWS.SNS.MOBILE.WNS.Group</code>
	<code>AWS.SNS.MOBILE.WNS.Match</code>
	<code>AWS.SNS.MOBILE.WNS.SuppressPopup</code>
	<code>AWS.SNS.MOBILE.WNS.Tag</code>

プッシュ通知サービス	予約済みメッセージ属性
	AWS.SNS.MOBILE.WNS.TTL
	AWS.SNS.MOBILE.WNS.Type

¹ メッセージ属性が要件を満たさない場合、Apple は Amazon SNS 通知を拒否します。詳細については、Apple Developer ウェブサイトの「[APNs への通知リクエストの送信](#)」を参照してください。

Amazon SNS メッセージのバッチ処理

メッセージのバッチ処理とは何ですか。

個別の Publish API リクエストで標準トピックまたは FIFO トピックにメッセージを発行する代わりに、Amazon SNS PublishBatch API を使用して、単一の API リクエストで最大 10 のメッセージを発行します。メッセージをバッチで送信すると、分散アプリケーションの接続 ([A2A メッセージング](#)) または Amazon SNS を使用した人への通知の送信 ([A2P メッセージング](#)) に関連するコストを最大 10 分の 1 に削減できます。Amazon SNS では、操作するリージョンに基づいて、1 秒間にトピックに発行できるメッセージの数に関するクォータがあります。API クォータの詳細については、「AWS 全般のリファレンス」の「[Amazon SNS エンドポイントとクォータ](#)」のページを参照してください。

Note

1 回の PublishBatch API 呼び出しで送信するすべてのメッセージの合計サイズは 262,144 バイト (256 KiB) を超えることはできません。

PublishBatch API は、IAM ポリシーの同じ Publish API アクションを使用します。

メッセージのバッチ処理はどのように機能しますか。

Publish API を使用したメッセージの公開は、PublishBatch API を使用したメッセージの公開に似ています。主な違いは、PublishBatch API リクエストでは、各メッセージに一意のバッチ ID (最大 80 文字) を割り当てる必要があります。このようにして、Amazon SNS は、バッチ内のすべてのメッセージに対して個々の API レスポンスを返して、各メッセージが発行されたか、または障害が発生したかを確認できます。FIFO トピックに発行されるメッセージについては、一意のバッチ ID の割り当てに加えて、個々のメッセージごとに MessageDeduplicationID および MessageGroupId を含める必要があります。

例

のメッセージのバッチを標準トピックに公開する

```
// Imports
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishBatchRequest;
import software.amazon.awssdk.services.sns.model.PublishBatchRequestEntry;
import software.amazon.awssdk.services.sns.model.PublishBatchResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

// Code
private static final int MAX_BATCH_SIZE = 10;

public static void publishBatchToTopic(SnsClient snsClient, String topicArn, int
    batchSize) {
    try {
        // Validate the batch size
        if (batchSize > MAX_BATCH_SIZE) {
            throw new IllegalArgumentException("Batch size cannot exceed " +
MAX_BATCH_SIZE);
        }

        // Create the batch entries
        List<PublishBatchRequestEntry> entries = IntStream.range(0, batchSize)
            .mapToObj(i -> PublishBatchRequestEntry.builder()
                .id("id" + i)
                .message("message" + i)
                .build())
            .collect(Collectors.toList());

        // Build the batch request
        PublishBatchRequest request = PublishBatchRequest.builder()
            .topicArn(topicArn)
            .publishBatchRequestEntries(entries)
            .build();

        // Publish the batch request
        PublishBatchResponse response = snsClient.publishBatch(request);
    }
```

```
// Handle successful messages
response.successful().forEach(success -> {
    System.out.println("Successful Batch Id: " + success.id());
    System.out.println("Message Id: " + success.messageId());
});

// Handle failed messages
response.failed().forEach(failure -> {
    System.err.println("Failed Batch Id: " + failure.id());
    System.err.println("Error Code: " + failure.code());
    System.err.println("Sender Fault: " + failure.senderFault());
    System.err.println("Error Message: " + failure.message());
});

} catch (SnsException e) {
    // Log and handle exceptions
    System.err.println("SNS Exception: " + e.awsErrorDetails().errorMessage());
} catch (IllegalArgumentException e) {
    System.err.println("Validation Error: " + e.getMessage());
}
}
```

メッセージのバッチを FIFO トピックに公開する

```
// Imports
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishBatchRequest;
import software.amazon.awssdk.services.sns.model.PublishBatchRequestEntry;
import software.amazon.awssdk.services.sns.model.PublishBatchResponse;
import software.amazon.awssdk.services.sns.model.BatchResultErrorEntry;
import software.amazon.awssdk.services.sns.model.SnsException;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

// Code
private static final int MAX_BATCH_SIZE = 10;

public static void publishBatchToFifoTopic(SnsClient snsClient, String topicArn) {
    try {
        // Create the batch entries to send
        List<PublishBatchRequestEntry> entries = IntStream.range(0, MAX_BATCH_SIZE)
```

```
        .mapToObj(i -> PublishBatchRequestEntry.builder()
            .id("id" + i)
            .message("message" + i)
            .messageGroupId("groupId")
            .messageDeduplicationId("deduplicationId" + i)
            .build())
        .collect(Collectors.toList());

// Create the batch request
PublishBatchRequest request = PublishBatchRequest.builder()
    .topicArn(topicArn)
    .publishBatchRequestEntries(entries)
    .build();

// Publish the batch request
PublishBatchResponse response = snsClient.publishBatch(request);

// Handle the successfully sent messages
response.successful().forEach(success -> {
    System.out.println("Batch Id for successful message: " + success.id());
    System.out.println("Message Id for successful message: " +
success.messageId());
    System.out.println("Sequence Number for successful message: " +
success.sequenceNumber());
});

// Handle the failed messages
response.failed().forEach(failure -> {
    System.err.println("Batch Id for failed message: " + failure.id());
    System.err.println("Error Code for failed message: " + failure.code());
    System.err.println("Sender Fault for failed message: " +
failure.senderFault());
    System.err.println("Failure Message for failed message: " +
failure.message());
});

} catch (SnsException e) {
    // Handle any exceptions from the request
    System.err.println("SNS Exception: " + e.awsErrorDetails().errorMessage());
}
}
```

Amazon SNS トピックおよびサブスクリプションを削除する

トピックが削除されると、それに関連付けられたサブスクリプションは非同期的に削除されます。これらのサブスクリプションには引き続きアクセスできますが、同じ名前を使用してトピックを再作成しても、サブスクリプションはトピックに関連付けられなくなります。パブリッシャーが削除されたトピックにメッセージを発行しようとする、パブリッシャーはトピックが存在しないことを示すエラーメッセージを受け取ります。同様に、削除されたトピックをサブスクライブしようとする、エラーメッセージが表示されます。確認待ちのサブスクリプションは削除できません。Amazon SNS は 48 時間後に未確認のサブスクリプションを自動的に削除します。ただし、30 日後に削除される E メールサブスクリプションを除きます。

Important

削除された Amazon SNS トピックは復元できません。トピックを削除する前に、サブスクリプションを削除することをお勧めします。

サブスクリプションが削除されると、復元することはできません。Amazon SNS トピックに発行された通知は、削除されたサブスクリプションには配信されません。通知を受け取るには、サブスクリプションを再作成する必要があります。

を使用して Amazon SNS トピックまたはサブスクリプションを削除するには AWS マネジメントコンソール

Amazon SNS トピックまたはサブスクリプションを削除すると、リソース管理が効率的になり、不要な使用を防いで、Amazon SNS コンソールを適切に整理できます。この手順により、不要になったトピックやサブスクリプションを削除して、アイドル状態のリソースによる潜在的なコストを回避し、管理を効率化できます。

を使用してトピックを削除するには AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. 左のナビゲーションペインで、[トピック] を選択します。
3. [トピック] ページでトピックを選択し、[編集] を選択します。
4. [削除] ダイアログボックスに「delete me」と入力し、[削除] を選択します。

コンソールがトピックを削除します。

を使用してサブスクリプションを削除するには AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. 左のナビゲーションペインで、[サブスクリプション] を選択します。
3. [サブスクリプション] ページで、確認済みステータスのサブスクリプションを選択してから、[削除] を選択します。
4. [サブネットの削除] ダイアログボックスで、[削除] を選択します。

コンソールはサブスクリプションを削除します。

AWS SDK を使用してサブスクリプションとトピックを削除するには

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、「AWS SDK とツールのリファレンスガイド」の「[共有設定ファイルと認証情報ファイル](#)」を参照してください。

次のサンプルコードは、DeleteTopic を使用する方法を説明しています。

.NET

SDK for .NET

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。

トピック ARN でトピックを削除します。

```
/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
```

```

        {
            TopicArn = topicArn
        });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }

```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[DeleteTopic](#)」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 an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::deleteTopic(const Aws::String &topicARN,
                              const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::DeleteTopicRequest request;
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::DeleteTopicOutcome outcome =
        snsClient.DeleteTopic(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted the Amazon SNS topic " << topicARN <<
            std::endl;
    }
}

```

```
else {
    std::cerr << "Error deleting topic " << topicARN << ":" <<
        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「[DeleteTopic](#)」を参照してください。

CLI

AWS CLI

SNS トピックを削除するには

次の delete-topic の例では、指定した SNS トピックを削除します。

```
aws sns delete-topic \
    --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic"
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[DeleteTopic](#)」を参照してください。

Go

SDK for Go V2

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 (
```

```
"context"
"encoding/json"
"log"

"github.com/aws/aws-sdk-go-v2/aws"
"github.com/aws/aws-sdk-go-v2/service/sns"
"github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type SnsActions struct {
    SnsClient *sns.Client
}

// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
    _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
        TopicArn: aws.String(topicArn)})
    if err != nil {
        log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
    }
    return err
}
```

- APIの詳細については、AWS SDK for Go API リファレンスの「[DeleteTopic](#)」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。


```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class DeleteTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:      <topicArn>

            Where:
                topicArn - The ARN of the topic to delete.
            "";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        System.out.println("Deleting a topic with name: " + topicArn);
        deleteSNSTopic(snsClient, topicArn);
        snsClient.close();
    }

    public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
        try {
            DeleteTopicRequest request = DeleteTopicRequest.builder()
```

```
        .topicArn(topicArn)
        .build();

        DeleteTopicResponse result = snsClient.deleteTopic(request);
        System.out.println("\n\nStatus was " +
            result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[DeleteTopic](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[DeleteTopic](#)」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 deleteSNSTopic(topicArnVal: String) {
    val request =
        DeleteTopicRequest {
            topicArn = topicArnVal
        }
}
```

```
    }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was successfully deleted.")
    }
}
```

- APIの詳細については、AWS SDK for Kotlin API リファレンスの「[DeleteTopic](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Deletes an SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnsClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);
```

```
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->deleteTopic([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- API の詳細については、AWS SDK for PHP API リファレンスの「[DeleteTopic](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def delete_topic(topic):
        """
```

```
Deletes a topic. All subscriptions to the topic are also deleted.
"""
try:
    topic.delete()
    logger.info("Deleted topic %s.", topic.arn)
except ClientError:
    logger.exception("Couldn't delete topic %s.", topic.arn)
    raise
```

```
class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def delete_topic(self, topic_arn: str) -> bool:
        """
        Delete an SNS topic.

        :param topic_arn: The ARN of the topic to delete.
        :return: True if successful.
        :raises ClientError: If the topic deletion fails.
        """
        try:
            self.sns_client.delete_topic(TopicArn=topic_arn)
```

```
logger.info(f"Deleted topic: {topic_arn}")
return True

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')

    if error_code == 'NotFound':
        logger.warning(f"Topic not found: {topic_arn}")
        return True # Already deleted
    else:
        logger.error(f"Error deleting topic: {error_code} - {e}")
        raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[DeleteTopic](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.
    lo_sns->deletetopic( iv_topicarn = iv_topic_arn ).
    MESSAGE 'SNS topic deleted.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[DeleteTopic](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

_ = try await snsClient.deleteTopic(
    input: DeleteTopicInput(topicArn: arn)
)
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[DeleteTopic](#)」を参照してください。

次の手順

これでサブスクリプションを使用してトピックを作成し、トピックにメッセージを送信したので、以下を試すことができます。

- [\[AWS デベロッパーセンター\]](#) の詳細。
- [\[セキュリティ\]](#) セクションで、データのアクセスの保護について説明します。
- トピックの[サーバー側の暗号化](#)を有効にします。
- [暗号化された Amazon Simple Queue Service \(Amazon SQS\) キュー](#)がサブスクライブされ、トピックのサーバー側の暗号化を有効にします。
- [AWS Event Fork Pipelines](#) をトピックにサブスクライブします。

Amazon SNS FIFO トピックを使用したメッセージの順序付けと重複排除戦略

このトピックでは、Amazon SNS FIFO (先入れ先出し) トピックの特徴と機能、およびそれらを [Amazon SQS FIFO キュー](#) と統合する方法について説明します。これらのサービスをあわせて使用して、データ整合性を必要とするアプリケーションに不可欠な厳密なメッセージの順序付けと重複排除を実現する方法について説明します。このコンテンツでは、Amazon SNS FIFO トピックが役立つ特定のユースケースについて説明し、メッセージの順序と一意性が重要なシナリオに関するインサイトを提供します。

また、メッセージの順序付け、メッセージのグループ化、それらによるメッセージ配信への影響の技術的な詳細についても説明します。メッセージ重複排除のトピックでは、重複メッセージを防ぎ、各メッセージが 1 回のみ処理されるようにするメカニズムについて説明します。さらに、メッセージングシステムの整合性と信頼性の確保のために重要となる、メッセージフィルタリング、セキュリティ、耐久性についても説明します。このコンテンツには、メッセージアーカイブとリプレイに関する情報も含まれており、メッセージ履歴を管理するための戦略を提供します。これらの機能を独自のアプリケーションに実装するために役立つ実用的なコード例も提供されており、Amazon SNS FIFO トピックと Amazon SQS FIFO キューの統合を実際に体験できます。

Amazon SNS の高スループット FIFO トピック

Amazon SNS の高スループット FIFO トピックは、厳密なメッセージ順序を維持しながら、高いメッセージスループットを効率的に管理することで、多数のメッセージを処理するアプリケーションの信頼性とスケーラビリティを確保します。このソリューションは、高スループットおよび順序付けられたメッセージ配信の両方を必要とするシナリオに最適です。高スループット FIFO トピックを使用してメッセージのスループットを向上させるには、メッセージグループの数を増やすことを推奨します。高スループットのメッセージクォータの詳細については、「Amazon Web Services 全般のリファレンス」の「[Amazon SNS Service Quotas](#)」を参照してください。

Amazon SNS FIFO トピックに関する高スループットのユースケース

以下のユースケースでは、高スループット FIFO トピックの多様な応用例を紹介し、さまざまな業界やシナリオでの効果を示します。

- リアルタイムデータ処理: イベント処理やテレメトリデータインGESTなど、リアルタイムデータストリームを扱うアプリケーションは、高スループット FIFO トピックを利用することで、メッ

ページの継続的な流入を処理しながら、メッセージの順序を維持して正確な分析を行うことができます。

- e コマース注文処理: 顧客取引の順序を維持することが重要な e コマースプラットフォームでは、高スループット FIFO トピックを利用することで、ショッピングのピークシーズンでも注文を遅滞なく順番に配信できます。
- 金融サービス: 高頻度の取引データやトランザクションデータを処理する金融機関は、高スループット FIFO トピックを利用することで、メッセージの順序付けに関する厳格な規制要件を遵守しながら、レイテンシーを最小限に抑えて市場データやトランザクションを処理できます。
- メディアストリーミング: ストリーミングプラットフォームやメディア配信サービスは、高スループット FIFO トピックを利用することで、メディアファイルとストリーミングコンテンツの配信を管理し、コンテンツ配信の正しい順序を維持しながら、ユーザーにスムーズな再生エクスペリエンスを提供できます。

Amazon SNS FIFO トピックの高スループットを実現するパーティションとデータ分散

高スループットのトピックでは、Amazon SNS は、FIFO トピックデータを複数のパーティションに分散します。パーティションとはトピックの容量の割り当てであり、AWS リージョン内の複数のアベイラビリティゾーンにわたって、自動的にレプリケートされます。パーティションは管理しません。代わりに、Amazon SNS はユーザーに代わって、インGRESレートに基づいてパーティションを自動的に管理します。

FIFO トピックでは、Amazon SNS は次の状況でトピック内のパーティションの数を変更します。

- 現在の発行レートが、既存のパーティションがサポートできる値に近づいたり超えたりすると、トピックがリージョンのクォータに達するまで追加のパーティションが割り当てられます。クォータの詳細については、「Amazon Web Services 全般のリファレンス」の「[Amazon SNS Service Quotas](#)」を参照してください。
- 現在のパーティションの使用率が低い場合は、パーティションの数が減ることがあります。

パーティション管理は自動的にバックグラウンドで自動的に発生し、アプリケーションに対して透過的です。トピックとメッセージは常に利用可能です。

Note

通常の数倍あるボリュームを送信しているときに、トピックへのトラフィックを急激かつ大幅に増加させると、一時的な [Publish](#) API スロットリングが発生する可能性があります。このスロットリングは、トピックがトラフィックの増加に合わせてスケールアップしている間、最長で重複排除ウィンドウの期間まで続く可能性があります。

メッセージグループ ID によるデータの配布

FIFO トピックにメッセージを発行すると、Amazon SNS は各メッセージのメッセージグループ ID の値を内部ハッシュ関数への入力値として使用します。ハッシュ関数からの出力値は、メッセージを処理するパーティションを決定します。その際、1 つ以上のメッセージグループ ID が、特定のパーティションで処理される場合があります。

Note

Amazon SNS は、パーティションの数に関係なく、FIFO トピックのパーティション間で項目を均一に分散するように最適化されています。では、多数の個別の値を持つメッセージグループ IDs を使用する AWS ことをお勧めします。

Amazon SNS FIFO トピックでの高スループットを有効にする

デフォルトでは、Amazon SNS FIFO トピックはトピックレベルの重複排除用に設定されています。これは、Topic に設定されたトピック属性 [FifoThroughputScope](#) によって制御されており、スループットクォータはより厳しく制限されています。詳細については「Amazon Web Services 全般のリファレンス」の「[Amazon SNS Service Quotas](#)」を参照してください。

Amazon SNS FIFO トピックの高スループットを有効にするには、FifoThroughputScope 属性を MessageGroup に更新します。この変更は、コンソールまたは AWS CLI と SDK を使用して実行でき、トピックの作成時に設定することもできます。Amazon SNS では、最高のカスタマーエクスペリエンスを実現し、トピックがスロットリングされる可能性を減らすためにこれを推奨しています。

Important

トピックの FifoThroughputScope を MessageGroup に設定して有効にすると Topic スループットに戻すことはできません。

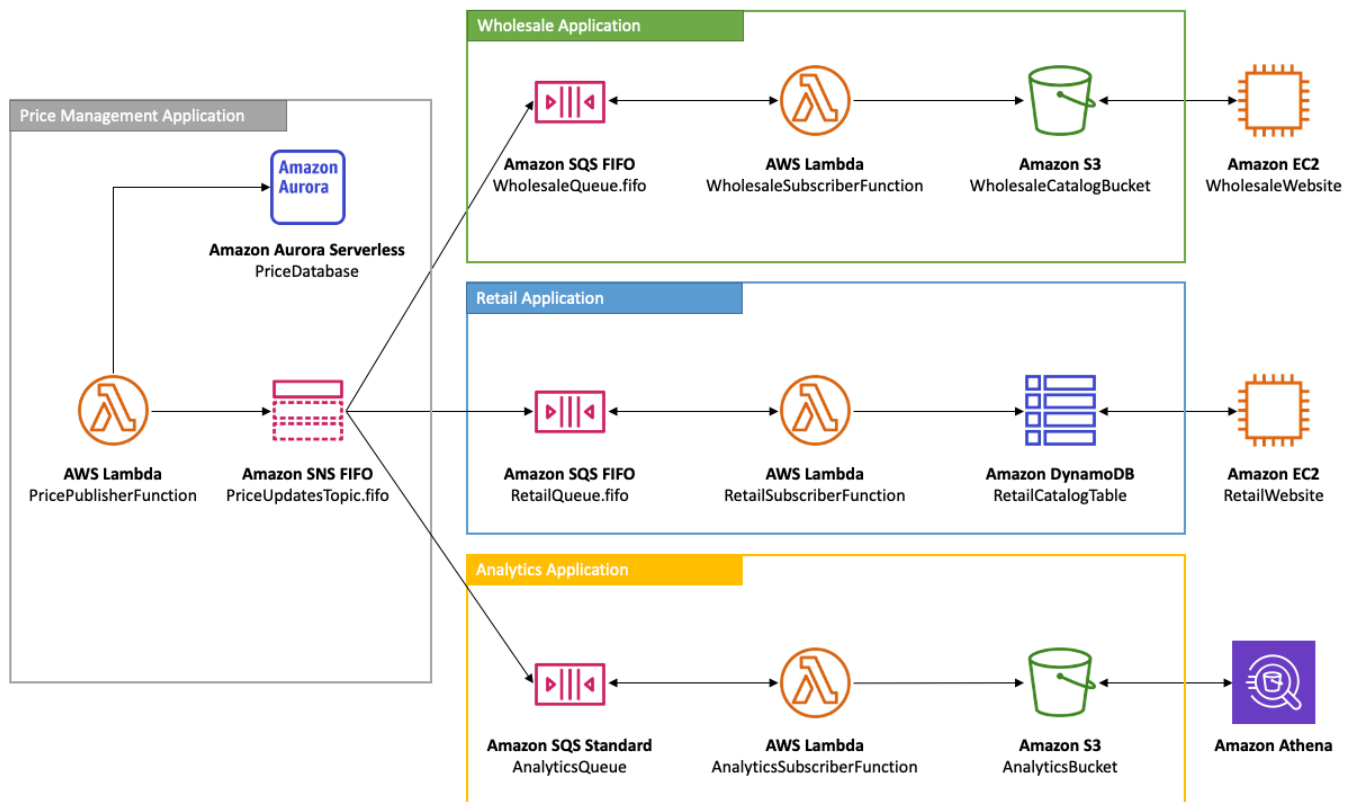
サブスクライブしている Amazon SQS FIFO キューの高スループットモードを有効にする

高スループットが有効になっている Amazon SNS FIFO トピックにメッセージを発行しており、1 つ以上の Amazon SQS FIFO キューがサブスクライブされている場合は、Amazon SQS FIFO キューでも高スループットを有効にすることが推奨されます。これにより、Amazon SNS FIFO 高スループットトピックがスムーズにメッセージを配信することが可能になります。詳細については、「Amazon Simple Queue Service デベロッパーガイド」の「[FIFO キューの高スループット](#)」を参照してください。

Amazon SNS FIFO トピックのユースケース例

次の例は、自動車部品メーカーが Amazon SNS FIFO トピックと Amazon SQS キューを使用して構築した e コマースプラットフォームを示しています。プラットフォームは 4 つのサーバーレスアプリケーションで構成されています。

- インベントリ管理者は、価格管理アプリケーションを使用して、在庫の各品目の価格を設定します。この会社では、為替変動や市場需要、販売戦略の推移などにより、商品価格が変動する可能性があります。価格管理アプリケーションは、AWS Lambda 関数を使用して、価格が変更されるたびに Amazon SNS FIFO トピックに価格の更新を発行します。
- 卸売アプリケーションは、自動車修理工場や自動車メーカーがその自動車部品を一括で購入できるウェブサイトのバックエンドを提供します。価格変更通知を取得する場合、卸売アプリケーションは Amazon SQS FIFO キューを価格管理アプリケーションの Amazon SNS FIFO トピックにサブスクライブします。
- 小売アプリケーションは、車の所有者や車のチューニング愛好家が自分の車の個々の自動車部品を購入できるウェブサイトのバックエンドを提供します。価格変更通知を取得する場合、小売アプリケーションも Amazon SQS FIFO キューを価格管理アプリケーションの Amazon SNS FIFO トピックにサブスクライブします。
- 分析アプリケーションは、価格の更新を集約して Amazon S3 バケットに保存し、Amazon Athena がビジネスインテリジェンス (BI) 目的でバケットをクエリできるようにします。価格変更通知を取得する場合、分析アプリケーションは Amazon SQS 標準キューを価格管理アプリケーションの Amazon SNS FIFO トピックにサブスクライブします。他のアプリケーションとは異なり、分析アプリケーションでは、価格更新を厳密に順序付ける必要はありません。

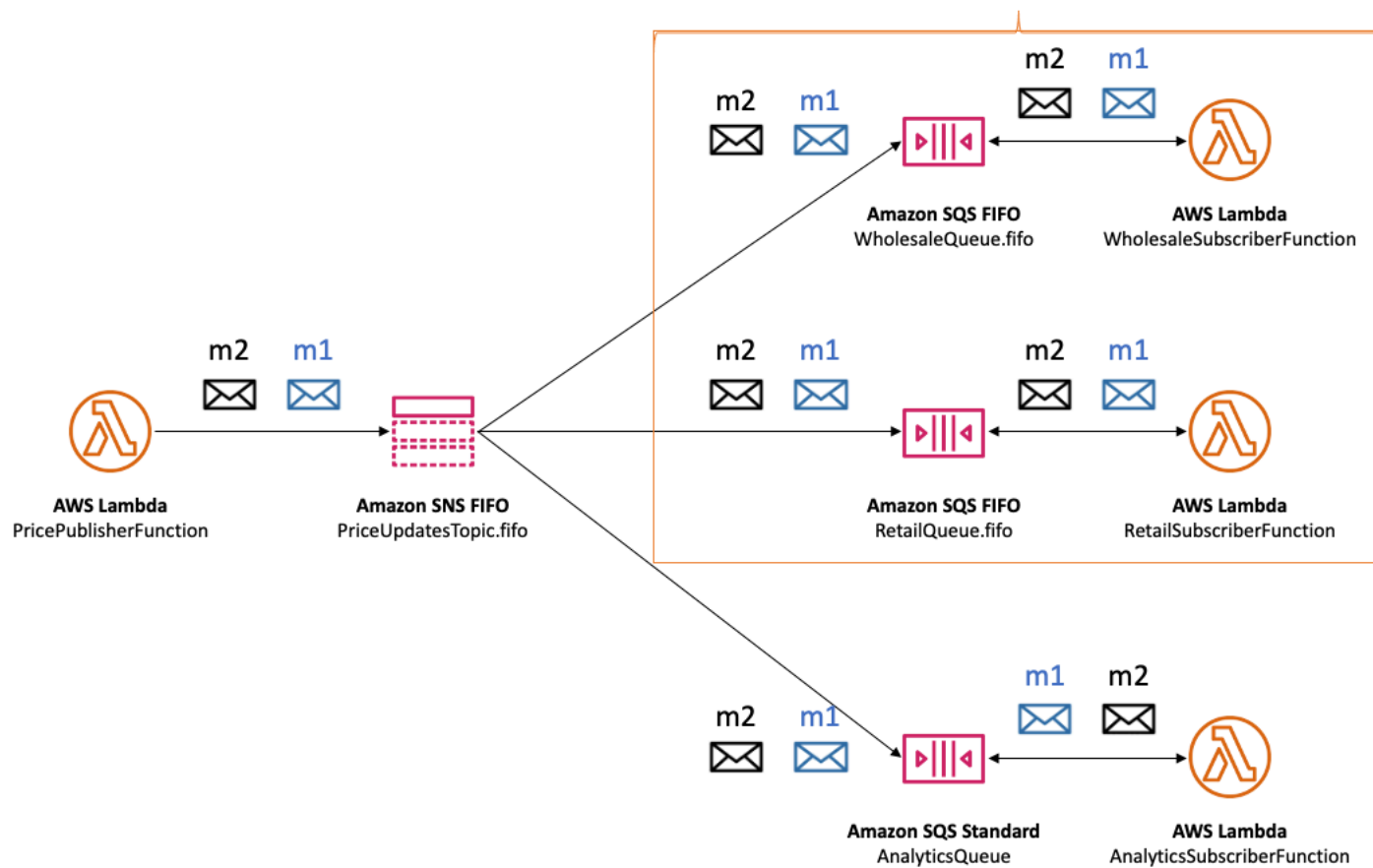


卸売アプリケーションおよび小売アプリケーションが正しい順序で価格の更新を受信するには、価格管理アプリケーションは、厳密に注文されたメッセージ配信システムを使用する必要があります。Amazon SNS FIFO トピックと Amazon SQS FIFO キューを使用すると、メッセージを順序どおりに重複なく処理できます。詳細については、「[FIFO トピックの Amazon SNS メッセージ順序の詳細](#)」を参照してください。このユースケースを実装するコードスニペットについては、「[FIFO トピックの Amazon SNS コード例](#)」を参照してください。

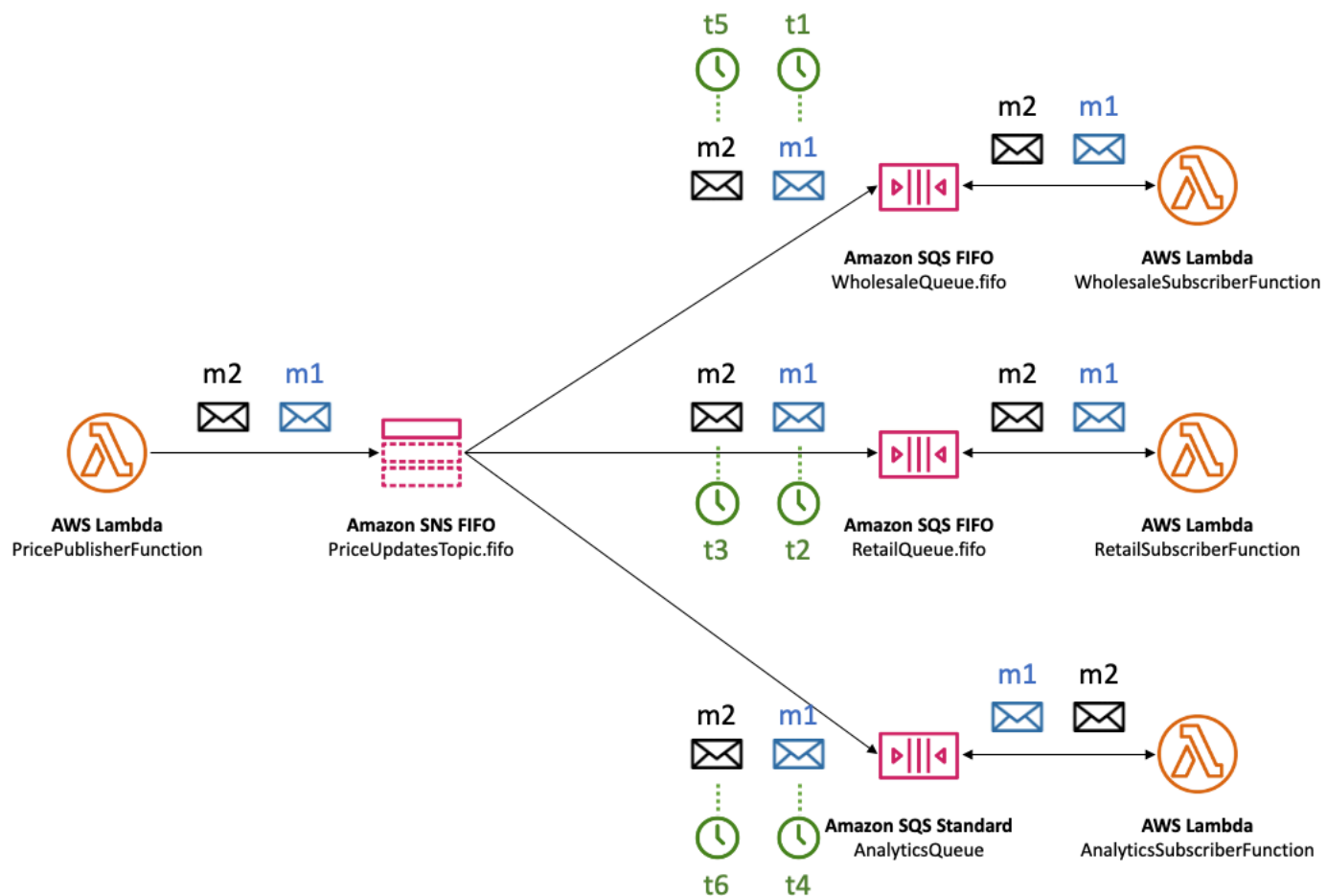
FIFO トピックの Amazon SNS メッセージ順序の詳細

Amazon SNS FIFO トピックは、常に、メッセージがトピックに発行されるのとまったく同じ順序で、サブスクライブされた Amazon SQS FIFO キューにメッセージを 1 回だけ配信します。Amazon SQS FIFO キューをサブスクライブすると、キューのコンシューマーは、メッセージがキューに配信されるのとまったく同じ順序で、メッセージを重複することなく受信します。ただし、SQS 標準キューをサブスクライブすると、キューのコンシューマーはメッセージを順序どおりではなく、さらに重複して受信する可能性があります。これにより、サブスクライバーをパブリッシャーからさらに切り離すことができ、次の図に示すように、「[Amazon SNS FIFO トピックのユースケース例](#)」に基づいて、サブスクライバーはメッセージの使用とコストの最適化をより柔軟に行うことができます。

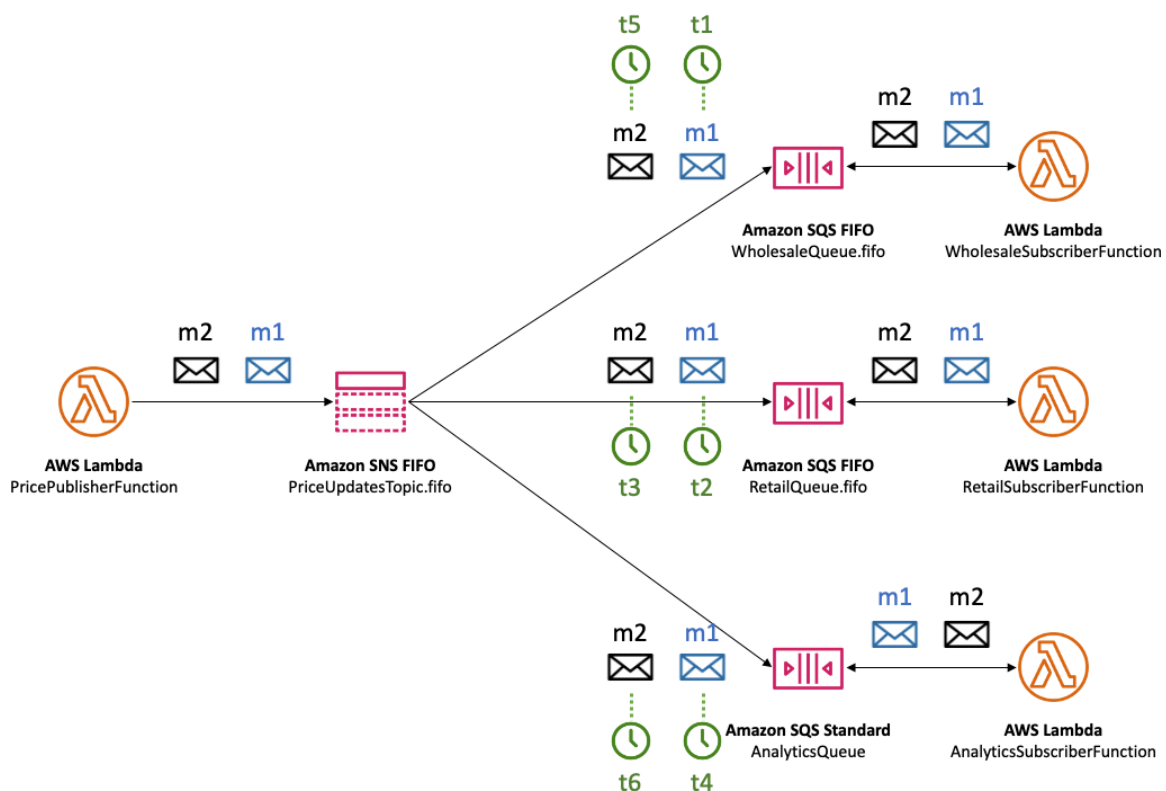
Messages are strictly ordered, using SNS FIFO and SQS FIFO



サブスクライバーの暗黙の順序付けはないことに注意してください。次の例は、メッセージ m1 が卸売サブスクライバー、小売サブスクライバー、分析サブスクライバーの順に配信されることを示しています。メッセージ m2 は、小売サブスクライバー、卸売サブスクライバー、分析サブスクライバーの順に配信されます。2つのメッセージは異なる順序でサブスクライバーに配信されますが、メッセージの順序は Amazon SQS FIFO サブスクライバーごとに保持されます。各サブスクライバーは、他のサブスクライバーとは切り離して認識されます。

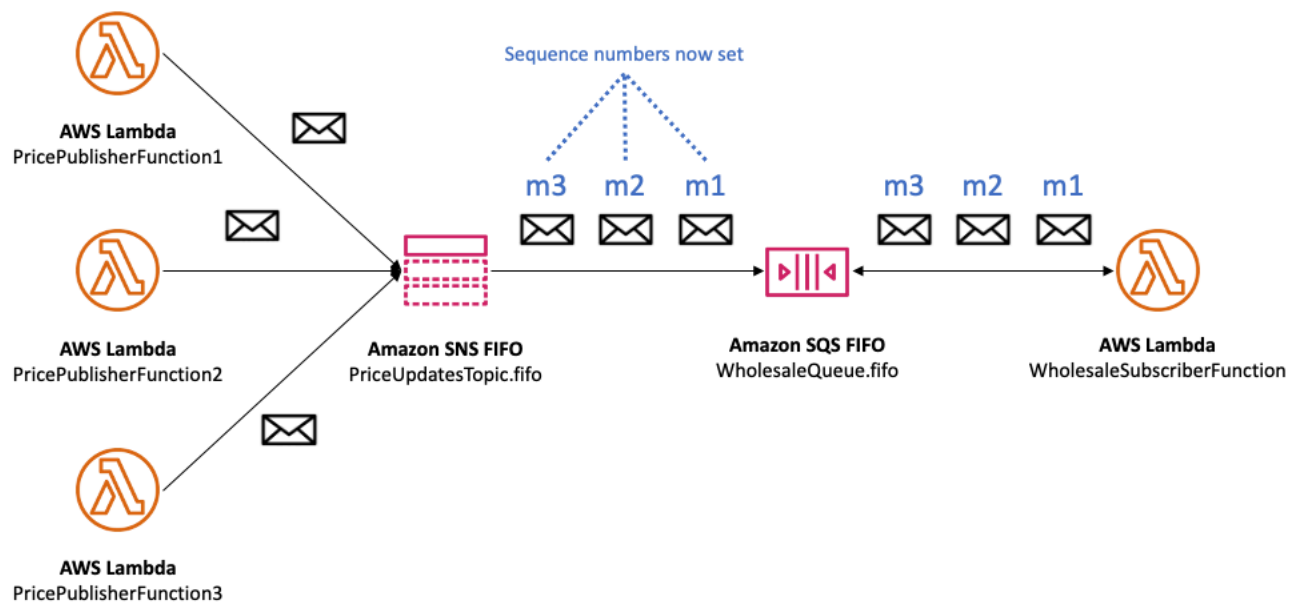


Amazon SQS キューサブスクライバーが到達不能になると、同期が失われる可能性があります。例えば、卸売アプリケーションキュー所有者が誤って [Amazon SQS キューポリシー](#) を変更し、Amazon SNS サービスプリンシパルがメッセージをキューに配信できなくなったとします。この場合、卸売キューへの価格更新配信は失敗し、小売キューと分析キューへの配信は成功するため、サブスクライバーは同期しなくなります。卸売アプリケーションキュー所有者がキューポリシーを修正すると、Amazon SNS はサブスクライブされたキューへのメッセージ配信を再開します。対応するサブスクリプションに [デッドレターキュー](#) が設定されていない限り、誤設定されたキューを対象とするトピックに発行されたメッセージはすべてドロップされます。



複数のアプリケーション (または同じアプリケーション内の複数のスレッド) が、SNS FIFO トピックにメッセージを並行して発行できます。これを行うと、メッセージシーケンスを Amazon SNS サービスに効果的に委任します。確立されたメッセージのシーケンスを決定するには、シーケンス番号を確認します。

シーケンス番号は、Amazon SNS が各メッセージに割り当てる、連続しない大きな数字です。シーケンス番号の長さは 128 ビットで、[メッセージグループ](#)ごとに増え続けます。シーケンス番号は、メッセージ本文の一部として、サブスクライブされた Amazon SQS キューに渡されます。ただし、[raw メッセージ配信](#)を有効にすると、Amazon SQS キューに配信されるメッセージには、シーケンス番号やその他の Amazon SNS メッセージメタデータは含まれません。



Amazon SNS FIFO トピックは、メッセージグループのコンテキストで順序を定義します。詳細については、「[FIFO トピックの Amazon SNS メッセージのグループ化](#)」を参照してください。

FIFO トピックの Amazon SNS メッセージのグループ化

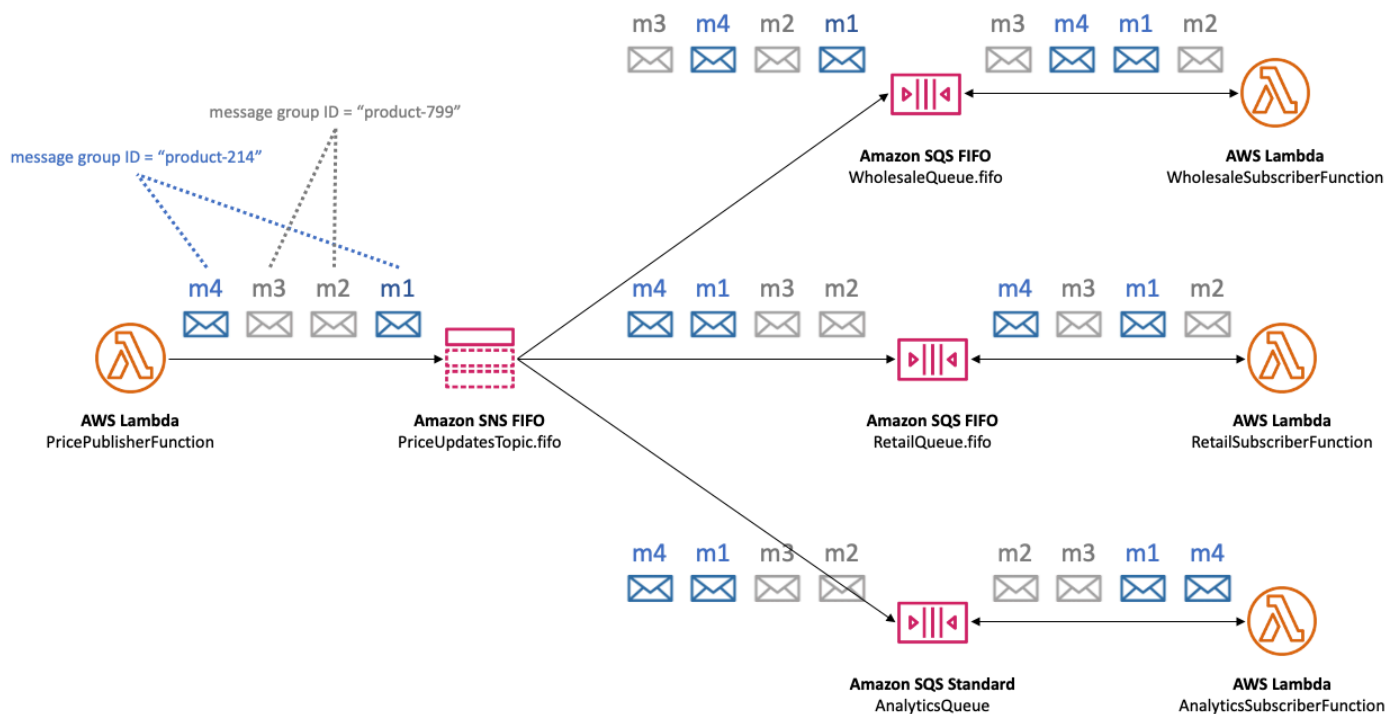
同じグループに属するメッセージは、グループに対する厳密な順序で 1 つずつ処理されます。

メッセージを Amazon SNS FIFO トピックに発行する際、メッセージのグループ ID を設定します。グループ ID は、メッセージが特定のメッセージグループに属することを指定する必須トークンです。SNS FIFO トピックは、グループ ID をサブスクライブされた Amazon SQS FIFO キューに渡します。SNS FIFO トピックまたは SQS FIFO キュー内のグループ ID の数に制限はありません。メッセージグループ ID は Amazon SQS 標準キューには渡されません。

メッセージグループとサブスクリプションの間にアフィニティはありません。したがって、任意のメッセージグループに発行されたメッセージは、サブスクリプションに添付されたフィルターポリシーに従って、すべてのサブスクライブ済みキューに配信されます。詳細については、「[FIFO トピックの Amazon SNS メッセージ配信](#)」および「[SNS FIFO トピックの Amazon SNS メッセージフィルター処理](#)」を参照してください。

[自動車部品価格管理のユースケース例](#)では、プラットフォームで販売されている製品ごとに専用のメッセージグループがあります。すべての価格更新の処理には、同じ Amazon SNS FIFO トピックが使用されます。価格更新の順序は、単一の自動車部品製品のコンテキスト内で保持されますが、ない複数の製品にまたがっていません。この仕組みを以下に示します。メッセージグループ ID が product-214 の製品の場合、メッセージ m1 は m4 の前に処理されることに注意してください。この

順序は、Amazon SNS FIFO と Amazon SQS FIFO を使用するワークフロー全体を通じて保持されます。同様に、メッセージグループ ID が product-799 の製品の場合、メッセージ m2 は m3 の前に処理されます。ただし、Amazon SQS 標準キューを使用すると、メッセージの順序は保証されなくなり、メッセージグループは存在しません。product-214 および product-799 メッセージグループは互いに独立しているため、メッセージの順序付けには関係がありません。



メッセージグループ ID 別のデータ配布によりパフォーマンスを改善

配信スループットを最適化するために、Amazon SNS FIFO トピックは異なるメッセージグループからのメッセージを並列配信しますが、メッセージの順序は各メッセージグループ内で厳密に維持されます。個々のメッセージグループごとに、1 秒あたり最大 300 件のメッセージを配信できます。したがって、1 つのトピックで高いスループットを実現するには、多数の異なるメッセージグループ ID を使用してください。Amazon SNS FIFO トピックは、多様なメッセージグループのセットを利用することで、メッセージを多数の並列パーティションに自動的に配布します。

Note

Amazon SNS FIFO トピックは、グループ数に関係なく、メッセージグループ IDs 間でメッセージを均一に分散するように最適化されています。パフォーマンスを最適化するために、多数の個別のメッセージグループ IDs を使用する AWS ことをお勧めします。

スループットが高い Amazon SNS FIFO トピックに公開し、1 つ以上の Amazon SQS FIFO キューがサブスクライブされている場合は、キューで高スループットを有効にすることをお勧めします。詳細については、「Amazon Simple Queue Service デベロッパーガイド」の「[FIFO キューの高スループット](#)」を参照してください。

FIFO トピックの Amazon SNS メッセージ配信

Amazon SNS FIFO (先入れ先出し) トピックは、Amazon SQS 標準キューと FIFO キューの両方への配信をサポートしており、ほぼリアルタイムのデータ整合性を必要とする分散アプリケーションを統合する際に柔軟性と制御をお客様に提供します。

厳密なメッセージ順序付けや重複排除を維持する必要があるワークロードの場合、Amazon SNS FIFO トピックと、配信エンドポイントとしてサブスクライブされている [Amazon SQS FIFO キュー](#) を組み合わせると、オペレーションやイベントの順序が重要であるときや重複が許容されないときにアプリケーション間のメッセージングを強化できます。

ベストエフォート型の順序付けと少なくとも 1 回の配信を許容するワークロードでは、[Amazon SQS 標準キュー](#) を Amazon SNS FIFO トピックにサブスクライブすると、FIFO を利用しないワークロード間でキューを共有できるだけでなく、コストを削減できます。

Note

Amazon SNS FIFO トピックから AWS Lambda 関数へメッセージをファンアウトするには、余分なステップが必要となります。まず、Amazon SQS FIFO キューまたは標準キューをトピックにサブスクライブします。次に、関数をトリガーするようにキューを設定します。詳細については、『AWS コンピューティングブログ』の「[イベントソースとしての SQS FIFO](#)」を参照してください。

SNS FIFO トピックでは、E メールアドレス、モバイルアプリケーション、テキストメッセージング (SMS) の電話番号、HTTP (S) エンドポイントなど、顧客マネージドエンドポイントにメッセージを

配信することはできません。これらのエンドポイントタイプは、厳密なメッセージの順序を維持することは保証されません。顧客マネージドエンドポイントを SNS FIFO トピックにサブスクライブしようとすると、エラーが発生します。

SNS FIFO トピックは、標準トピックと同じメッセージフィルター処理機能をサポートします。詳細については、「[SNS FIFO トピックの Amazon SNS メッセージフィルター処理](#)」および『AWS コンピューティングブログ』の「[Amazon SNS メッセージフィルター処理による Pub/Sub メッセージングの簡素化](#)」の投稿を参照してください。

SNS FIFO トピックの Amazon SNS メッセージフィルター処理

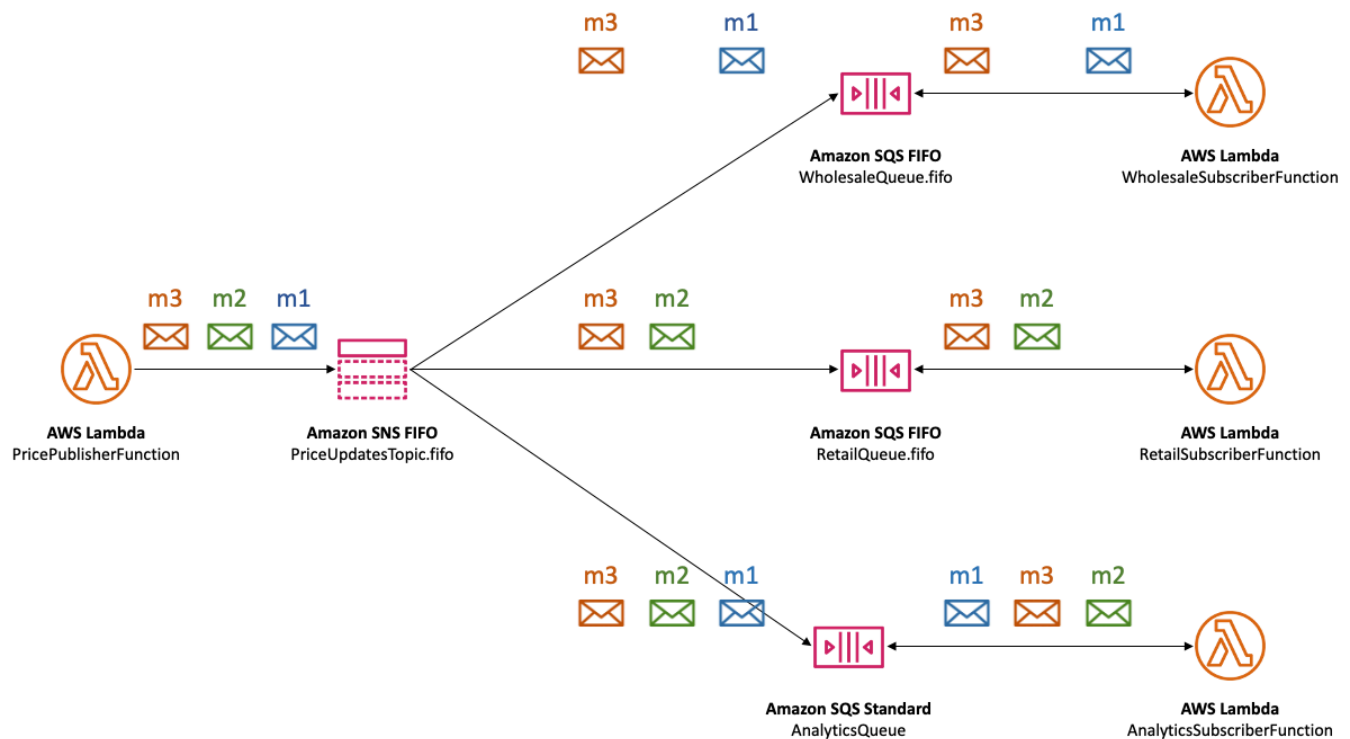
Amazon SNS FIFO トピックでは、メッセージフィルター処理がサポートされています。メッセージのフィルター処理により、受信者システムからのメッセージのルーティングロジックと、受信者システムからのメッセージのフィルター処理ロジックをオフロードすることにより、アーキテクチャが簡素化されます。

Amazon SQS FIFO キューまたは標準キューを SNS FIFO トピックにサブスクライブする場合、メッセージフィルタリングを使用して、サブスクライバーが、すべてのメッセージではなく、メッセージのサブセットを受信するように指定できます。各サブスクライバーは、独自のフィルターポリシーをサブスクリプション属性として設定できます。フィルターポリシーの範囲に基づいて、フィルターポリシーが受信メッセージ属性またはメッセージ本文と照合されます。フィルターポリシーが一致する場合、トピックはメッセージのコピーをサブスクライバーに配信します。一致するものがない場合、トピックはメッセージのコピーを配信しません。

[自動車部品価格管理のユースケース例](#)では、以下の Amazon SNS フィルターポリシーが設定されていて、フィルターポリシーの範囲が MessageBody であると仮定します。

- 卸売キューについては、フィルターポリシー `{"business":["wholesale"]}` は、business という名前のキーと値のセットの wholesale を持つすべてのメッセージと一致します。次の図では、メッセージ m1 内のキーの 1 つは wholesale の値を持つ business です。メッセージ m3 内の 1 つのキーは `["wholesale,retail"]` の値を持つ business です。したがって、m1 および m3 の両方がフィルターポリシーの基準に一致し、両方のメッセージが卸売キューに配信されます。
- 小売キューについては、フィルターポリシー `{"business":["retail"]}` は、business という名前のキーと値のセットの retail を持つすべてのメッセージと一致します。図では、メッセージ m2 内のキーの 1 つは retail の値を持つ business です。メッセージ m3 内の 1 つのキーは、`["wholesale,retail"]` の値を持つ business です。したがって、m2 および m3 の両方がフィルターポリシーの基準に一致し、両方のメッセージが小売キューに配信されます。

- 分析キューでは、Amazon Athena ですべてのレコードを受信したいため、フィルタポリシーは適用しません。



SNS FIFO トピックは、属性文字列値、属性数値、属性キーなど、さまざまなマッチング演算子をサポートします。詳細については、「[Amazon SNS メッセージフィルター処理](#)」を参照してください。

SNS FIFO トピックは、サブスクライブされたエンドポイントに重複メッセージを配信しません。詳細については、「[FIFO トピックの Amazon SNS メッセージ重複排除](#)」を参照してください。

FIFO トピックの Amazon SNS メッセージ重複排除

Amazon SNS FIFO トピックおよび Amazon SQS FIFO キューでは、メッセージの重複排除がサポートされます。これは、以下の条件が満たされている限り、メッセージの配信と処理を 1 回だけ行います。

- サブスクライブされた SQS FIFO キューがあり、このキューのアクセス許可により、このキューへのメッセージ配信を Amazon SNS サービスプリンシパルに許可します。

- Amazon SQS FIFO キューコンシューマーはメッセージを処理し、可視性タイムアウトの期限が切れる前に、キューからメッセージを削除します。
- Amazon SNS サブスクリプショントピックには、[メッセージのフィルター処理](#)がありません。メッセージフィルター処理を設定すると、サブスクリプションフィルターポリシーに基づいてメッセージをフィルター処理できるため、Amazon SNS FIFO トピックは最大 1 回の配信をサポートします。
- メッセージ配信の確認応答を妨げるネットワークの中断はありません。

Note

メッセージの重複排除は、トピック属性 `FifoThroughputScope` が `Topic` に設定されている場合、Amazon SNS FIFO トピック全体に適用されます。トピック属性 `FifoThroughputScope` が `MessageGroup` に設定されている場合、メッセージの重複排除は個々の[メッセージグループ](#)に適用されます。

メッセージを Amazon SNS FIFO トピックに発行する場合、メッセージには重複排除 ID を含める必要があります。この ID は、Amazon SNS FIFO トピックから、サブスクライブされた Amazon SQS FIFO キューに配信するメッセージに含まれます。

特定の重複排除 ID を持つメッセージが Amazon SNS FIFO トピックに正常に発行された場合、5 分間の重複排除インターバルの間、同じ重複排除 ID で発行されたメッセージは受け付けられますが、配信されません。Amazon SNS FIFO トピックは、サブスクライブされたエンドポイントにメッセージが配信された後も、トピック属性 `FifoThroughputScope` で設定された重複排除スコープでメッセージの重複排除 ID を追跡し続けます。

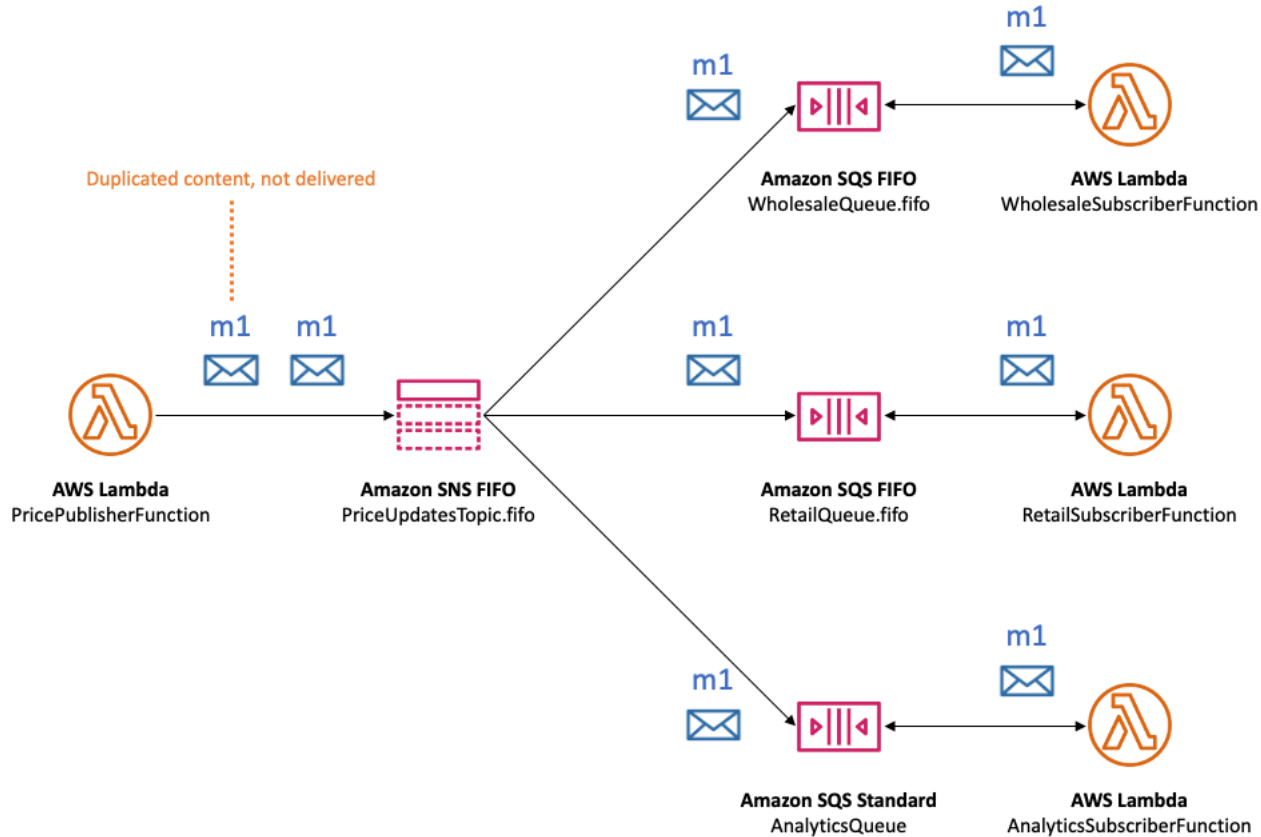
メッセージの発行ごとにメッセージ本文が一意であることが保証されている場合は、Amazon SNS FIFO トピックとサブスクライブされた Amazon SQS FIFO キューに対してコンテンツベースの重複排除を有効にできます。Amazon SNS は、メッセージ本文を使用して、各メッセージの重複除外 ID として使用する一意のハッシュ値を生成するため、各メッセージを送信するときに重複除外 ID を設定する必要はありません。

Note

メッセージ属性はハッシュ計算に含まれません。

Amazon SNS FIFO トピックでコンテンツベースの重複排除が有効になっている場合、ある重複排除 ID を使ってメッセージが発行されると、発行された重複排除 ID によって生成されたコンテンツベースの重複排除 ID が上書きされます。

[自動車部品価格管理のユースケース例](#)では、価格更新ごとに汎用的に一意の重複除外 ID を設定する必要があります。これは、メッセージ属性が卸売と小売で異なる場合でも、メッセージ本文が同一になる可能性があるためです。ただし、企業が、メッセージ本文に製品 ID および製品価格と共にビジネスタイプ (卸売または小売) を追加した場合、Amazon SNS FIFO トピックおよびサブスクライブされた Amazon SQS FIFO キューでコンテンツベースの重複を有効にできます。



メッセージの順序付けと重複排除に加えて、Amazon SNS FIFO トピックでは、AWS KMS キーによるメッセージのサーバー側の暗号化 (SSE)、および AWS PrivateLink による VPC エンドポイント経由のメッセージプライバシーをサポートしています。詳細については、「[FIFO トピックの Amazon SNS メッセージセキュリティ](#)」を参照してください。

FIFO トピックの Amazon SNS メッセージセキュリティ

[AWS Key Management Service \(AWS KMS\) カスタマーマスタークー \(CMK\)](#) を使用して、Amazon SNS FIFO トピックと Amazon SQS FIFO キューの暗号化を有効にすることができます。

- 新しい暗号化された FIFO トピックや FIFO キューを作成することも、既存のトピックやキューに対して暗号化を有効にすることもできます。
- メッセージ本文のみが暗号化されます。メッセージ属性、リソースメタデータ、リソースメトリクスは暗号化されません。

Note

既存の FIFO トピックまたはキューに暗号化を追加しても、バックログされたメッセージは暗号化されません。トピックまたはキューから暗号化を削除すると、バックログされたメッセージは暗号化されたままになります。

SNS FIFO トピックは、サブスクライブされたエンドポイントにメッセージを配信する直前にメッセージを復号化します。SQS FIFO キューは、メッセージをコンシューマーアプリケーションに返す直前にメッセージを復号化します。詳細については、「[Amazon SNS データ暗号化](#)」と AWS コンピューティングブログの「[AWS KMS で Amazon SNS に発行したメッセージを暗号化する](#)」の投稿を参照してください。

さらに、SNS FIFO トピックと SQS FIFO キューでは、AWS PrivateLink による [インターフェイス VPC エンドポイント](#) を用いたメッセージプライバシーをサポートしています。インターフェイスエンドポイントを使用すると、公開インターネットを経由することなく、Amazon Virtual Private Cloud (Amazon VPC) サブネットから FIFO トピックおよびキューにメッセージを送信できます。このモデルでは、メッセージは AWS インフラストラクチャとネットワークの中にとどめ、これにより、アプリケーションの全体的なセキュリティが強化されます。AWS PrivateLink を使用すると、インターネットゲートウェイ、ネットワークアドレス変換 (NAT)、またはバーチャルプライベートネットワーク (VPN) を設定する必要はありません。詳細については、「[VPC エンドポイントによる Amazon SNS トラフィックの保護](#)」と AWS コンピューティングブログの「[AWS PrivateLink で Amazon SNS に発行したメッセージの確保](#)」の投稿を参照してください。

SNS FIFO トピックは、デッドレターキューとアベイラビリティーゾーン間のメッセージストレージもサポートします。詳細については、「[FIFO トピックの Amazon SNS メッセージ耐久性](#)」を参照してください。

FIFO トピックの Amazon SNS メッセージ耐久性

Amazon SNS FIFO トピックと Amazon SQS キューは耐久性があります。どちらのリソースタイプも、複数のアベイラビリティゾーンにわたって冗長的にメッセージを保存し、例外的なケースを処理するためのデッドレターキューを提供します。

Amazon SNS では、クライアント側またはサーバー側のエラーにより、Amazon SNS トピックがサブスクライブされた Amazon SQS キューにアクセスできない場合、メッセージの配信は失敗します。

- Amazon SNS FIFO トピックに古いサブスクリプションのメタデータがあると、クライアント側のエラーが発生します。クライアント側のエラーの一般的な 2 つの原因は、Amazon SQS キュー所有者が次のいずれかを行う場合です。
 - キューを削除します。
 - Amazon SNS サービスプリンシパルがメッセージを配信できないようにキューポリシーを変更します。

Amazon SNS は、クライアント側のエラーにより失敗したメッセージの配信を再試行しません。

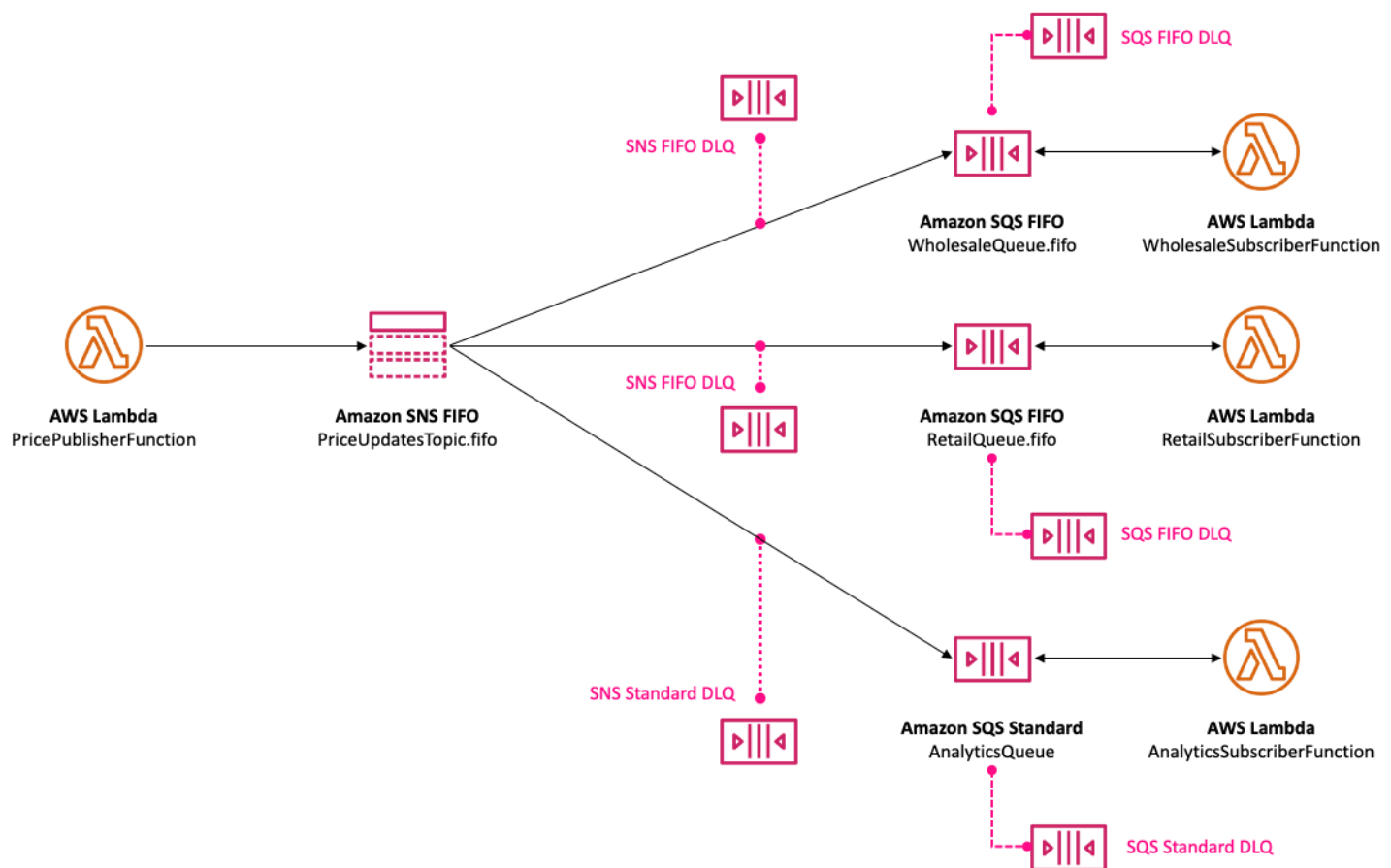
- サーバー側のエラーは、次の状況で発生する可能性があります。
 - Amazon SQS サービスは利用できません。
 - Amazon SQS は、Amazon SNS サービスからの有効なリクエストを処理できません。

サーバー側のエラーが発生すると、Amazon SNS FIFO トピックは、失敗した配信を 23 日間、最大 100,015 回再試行します。詳細については、「[Amazon SNS メッセージ配信の再試行](#)」を参照してください。

どのようなタイプのエラーでも、Amazon SNS は Amazon SQS デッドレターキューへのメッセージをサイドラインして、データが失われないようにします。

Amazon SQS では、コンシューマーアプリケーションがメッセージの受信、処理、およびキューからの削除に失敗すると、メッセージ処理は失敗します。最大数の受信リクエストが失敗した場合、Amazon SQS はメッセージをデッドレターキューにサイドラインし、データが失われないようにします。

[自動車部品価格管理のユースケース例](#)の場合、会社は Amazon SQS デッドレターキュー (DLQ) を各 Amazon SNS FIFO トピックサブスクリプションおよびサブスクライブされた各 Amazon SQS キューに割り当てることができます。これにより、価格更新の損失から会社を保護します。



Amazon SNS サブスクリプションに関連するデッドレターキューは、サブスクライブキューと同じタイプの Amazon SQS キューである必要があります。例えば、Amazon SQS FIFO キューの Amazon SNS FIFO サブスクリプションでは、デッドレターキューとして Amazon SQS FIFO キューが必要です。同様に、Amazon SQS 標準キューの Amazon SNS FIFO サブスクリプションには、デッドレターキューとして Amazon SQS 標準キューが必要です。詳細については、[Amazon SNS デッドレターキュー](#)と『AWS コンピューティングブログ』の「[Amazon SNS、Amazon SQS 用の DLQ を使用した耐久性のあるサーバーレスアプリケーションの設計、AWS Lambda](#)」の投稿を参照してください。

ダウンストリームの障害からの回復に役立つ耐久性が拡張され、トピック所有者は FIFO トピックを使用してメッセージを最大 365 日間アーカイブすることもできます。トピックサブスクライバーは、これらのメッセージをサブスクライブされたエンドポイントにリプレイして、ダウンストリームアプリケーションの障害によるメッセージを回復したり、既存のアプリケーションの状態を複製したりできます。詳細については、「[FIFO トピックの Amazon SNS メッセージのアーカイブとリプレイ](#)」を参照してください。

FIFO トピックの Amazon SNS メッセージのアーカイブとリプレイ

メッセージのアーカイブとリプレイとは

Amazon SNS は、FIFO (先入れ先出し) トピック用に特別に設計された、ノーコードのメッセージアーカイブおよびリプレイ機能を提供します。この機能により、トピック所有者はトピックアーカイブ内にメッセージを最大 365 日間直接保存して、必要に応じてサブスクライバーにリプレイできます。メッセージのアーカイブとリプレイは、失われたメッセージを復旧し、状態をレプリケートして、リージョンやシステム間でアプリケーションを同期するために不可欠です。

この機能は API AWS、SDK CloudFormation、および からアクセスできます AWS マネジメントコンソール。

主なユースケース

- メッセージの復旧 – ダウンストリームアプリケーションの障害により失われたメッセージを、サブスクライバーのエンドポイントにリプレイして復旧できます。
- 状態のレプリケーション – 特定のタイムスタンプからメッセージをリプレイし、既存のシステムの状態を新しい環境にレプリケートします。
- エラーの修正 – 停止中に失われたメッセージを再送信して、すべてのイベントが正しく処理されるようにします。

メッセージのアーカイブとリプレイのコンポーネント

Amazon SNS FIFO トピックのメッセージのアーカイブとリプレイを管理します。これには、保持期間の設定、CloudWatch を使用したアーカイブ済みメッセージのモニタリング、サブスクリプション属性によるリプレイの開始、リプレイの変更と開始に必要なアクセス許可の理解が含まれます。

メッセージのアーカイブ

- トピック所有者はアーカイブ機能を有効にして、メッセージの保持期間 (最大 365 日間) を設定できます。詳細については、「[FIFO トピック所有者の Amazon SNS メッセージアーカイブ](#)」を参照してください。
- CloudWatch メトリクスは、アーカイブされたメッセージのモニタリングに役立ちます。

メッセージのリプレイ

- サブスクライバーがメッセージの時間枠を選択してリプレイを開始すると、サブスクライブされたエンドポイントに再処理されます。詳細については、「[FIFO トピックサブスクライバーの Amazon SNS メッセージリプレイ](#)」を参照してください。
- `ReplayPolicy` の機能を使用して、サブスクリプション属性によってリプレイを管理します。

関連するアクセス許可

- **`SetSubscriptionAttributes`** – サブスクリプションの `ReplayPolicy` 属性を使用してリプレイ設定を設定または変更するために必要です。
- **`Subscribe`** – 新しいサブスクリプションをアタッチしリプレイを開始するために必要です。
- **`GetTopicAttributes`** – トピックのプロパティを表示できますが、リプレイの開始は主にサブスクリプション管理を中心に行われます。

FIFO トピック所有者の Amazon SNS メッセージアーカイブ

メッセージのアーカイブでは、トピックに発行されたすべてのメッセージの 1 つのコピーをアーカイブできます。トピックのメッセージアーカイブポリシーを有効にすることで、発行されたメッセージをトピック内に保存できます。これにより、そのトピックにリンクされているすべてのサブスクリプションのメッセージアーカイブが有効になります。メッセージのアーカイブ期間は、最短で 1 日、最長で 365 日です。

アーカイブポリシーを設定すると、追加料金がかかります。料金については、「[Amazon SNS 料金](#)」を参照してください。

を使用してメッセージアーカイブポリシーを作成する 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する新しいメッセージアーカイブポリシーを作成するには、このオプションを使用します。

1. [Amazon SNS コンソール](#)にサインインします。
2. トピックを選択するか、新しいトピックを作成できます。トピックの作成の詳細については、「[Amazon SNS トピックを作成する](#)」を参照してください。

Note

Amazon SNS メッセージのアーカイブとリプレイは、Application-to-Application (A2A) FIFO トピックでのみ使用できます。

3. [トピックの編集] ページで [アーカイブポリシー] セクションを展開します。
4. アーカイブポリシー機能を有効にし、トピック内でメッセージを保存する日数を入力します。
5. [Save changes] (変更の保存) をクリックします。

メッセージアーカイブトピックポリシーを表示、編集、非アクティブ化するには

- トピックの詳細ページでは、設定されている日数などのアーカイブポリシーのステータスがリテンションポリシーに表示されます。[アーカイブポリシー] タブを選択すると、以下のメッセージアーカイブの情報が表示されます。
 - ステータス — アーカイブポリシーが適用されると、アーカイブとリプレイのステータスは [有効] と表示されます。アーカイブポリシーが空の JSON オブジェクトに設定されている場合は、アーカイブとリプレイのステータスは [無効] と表示されます。
 - メッセージ保持期間 — 指定されたメッセージ保持日数。
 - アーカイブ開始日 — サブスクライバーがメッセージを再生できる日付。
 - JSON のプレビュー — アーカイブポリシーの JSON プレビュー。
- (オプション) アーカイブポリシーを編集するには、トピックの [概要] ページに移動して [編集] を選択します。
- (オプション) アーカイブポリシーを無効化するには、トピックの [概要] ページに移動して [編集] を選択します。アーカイブポリシーを無効化し、[変更の保存] を選択します。
- (オプション) アーカイブポリシーを含むトピックを削除するには、前に説明したように、アーカイブポリシーを無効化する必要があります。

 Important

メッセージが誤って削除されるのを防ぐため、メッセージアーカイブポリシーが有効なトピックは削除できません。トピックを削除する前に、トピックのメッセージアーカイブポリシーを無効化する必要があります。メッセージアーカイブポリシーを無効にすると、Amazon SNS ではアーカイブされたすべてのメッセージが削除されます。トピックを削除すると、サブスクリプションは削除され、転送中のメッセージが配信されない場合があります。

API を使用するメッセージアーカイブポリシーを作成する

API を使用するメッセージアーカイブポリシーを作成するには、属性 `ArchivePolicy` をトピックに追加する必要があります。`ArchivePolicy` は、API アクション `CreateTopic` と `SetTopicAttributes` を使用して設定できます。`ArchivePolicy` には Amazon SNS がメッセージを保持する日数を表す単一の値 `MessageRetentionPeriod` が含まれています。トピックのメッセージアーカイブを有効にするには、`MessageRetentionPeriod` を 0 より大きい整数値に設定します。たとえば、メッセージをアーカイブに 30 日間保持するには、`ArchivePolicy` を次のように設定します。

```
{
  "ArchivePolicy": {
    "MessageRetentionPeriod": "30"
  }
}
```

トピックのメッセージアーカイブを無効にしてアーカイブを消去するには、次のように `ArchivePolicy` の設定を解除します。

```
{}
```

SDK を使用するメッセージアーカイブポリシーを作成する

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、[SDK config および ツールリファレンスガイドの「共有ファイル」および credentials「ファイル」](#)を参照してください。AWS SDKs

次のコード例は、Amazon SNS トピック用にそのトピックに発行されたすべてのメッセージを 30 日間保持するように `ArchivePolicy` を設定する方法を示しています。

```
// Specify the ARN of the Amazon SNS topic to set the ArchivePolicy for.
String topicArn =
    "arn:aws:sns:us-east-2:123456789012:MyArchiveTopic.fifo";

// Set the MessageRetentionPeriod to 30 days for the ArchivePolicy.
String archivePolicy =
    "{\"MessageRetentionPeriod\":\"30\"}";

// Set the ArchivePolicy for the Amazon SNS topic
SetTopicAttributesRequest request = new SetTopicAttributesRequest()
```

```
.withTopicArn(topicArn)
.withAttributeName("ArchivePolicy")
.withAttributeValue(archivePolicy);
sns.setTopicAttributes(request);
```

を使用してメッセージアーカイブポリシーを作成する CloudFormation

を使用してアーカイブポリシーを作成するには、AWS CloudFormation ユーザーガイド [AWS::SNS::Topic](#) の CloudFormation 「」を参照してください。

暗号化されたアーカイブにアクセス権を付与する

暗号化されたトピックからのメッセージのリプレイをサブスクライバーが開始する前に、次の手順を完了する必要があります。過去のメッセージがリプレイされるため、Amazon SNS では、アーカイブ内のメッセージの暗号化に使用された KMS キーへの Decrypt アクセス権がプロビジョニングされている必要があります。

1. KMS キーを使用してメッセージを暗号化し、トピック内に保存する場合、キーポリシーを使用してこれらのメッセージを復号化する機能を Amazon SNS に付与する必要があります。詳細については、「[Amazon SNS に復号のアクセス権限を付与する](#)」を参照してください。
2. Amazon SNS AWS KMS で を有効にします。詳細については、「[AWS KMS アクセス許可の設定](#)」を参照してください。

Important

新しいセクションを KMS キーポリシーに追加するときは、ポリシー内の既存のセクションを変更しないでください。トピックの暗号化が有効で、KMS キーが無効か、または削除されている場合、または KMS キーポリシーが Amazon SNS に対して正しく設定されていない場合は、Amazon SNS はサブスクライバーに対してメッセージをリプレイできません。

Amazon SNS に復号のアクセス権限を付与する

Amazon SNS がトピックのアーカイブ内の暗号化されたメッセージにアクセスし、サブスクライブされたエンドポイントにリプレイするには、Amazon SNS サービスプリンシパルでこれらのメッセージを復号する必要があります。

トピック内からの履歴メッセージのリプレイ中に、保存されたメッセージを Amazon SNS サービスプリンシパルが復号することを許可するために必要なポリシーの例を次に示します。


```
{
  "Sid": "Allow SNS to decrypt archived messages",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*"
}
```

Amazon CloudWatch を使用してメッセージアーカイブメトリクスをモニタリングする

Amazon CloudWatch では、以下のメトリックスを使用してアーカイブされたメッセージをモニタリングできます。ワークロードの異常を通知し、影響を回避するのに役立てるために、これらのメトリックスに Amazon CloudWatch アラームを設定できます。詳細については、[Amazon SNS でのログ記録とモニタリング](#)を参照してください。

メトリクス	説明
ApproximateNumberOfMessagesArchived	トピックアーカイブにアーカイブされたメッセージの総数を 60 分の解像度でトピック所有者に提供します。
ApproximateNumberOfBytesArchived	トピックアーカイブのすべてのメッセージで、アーカイブされたバイトの総数を 60 分の解像度でトピック所有者に提供します。
NumberOfMessagesArchiveProcessing	1 分の解像度のインターバル中に、トピックアーカイブに保存されたメッセージの数をトピックの所有者に通知します。
NumberOfBytesArchiveProcessing	1 分の解像度のインターバル中に、トピックアーカイブに保存されたバイトの総数をトピックの所有者に通知します。

GetTopicAttributes API には、購読者がリプレイを開始できる最も古いタイムスタンプを表す BeginningArchiveTime プロパティが含まれています。以下に、この API アクションのレスポンスの例を示します。

```
{
  "ArchivePolicy": {
    "MessageRetentionPeriod": "<integer>"
  },
  "BeginningArchiveTime": "<timestamp>",
  ...
}
```

FIFO トピックサブスクライバーの Amazon SNS メッセージリプレイ

Amazon SNS リプレイでは、トピックサブスクライバーはトピックデータストアからアーカイブされたメッセージを取得し、サブスクライブされたエンドポイントに再配信できます。

- メッセージは、サブスクリプションが作成されるとすぐにリプレイできます。
- リプレイされたメッセージは、元のメッセージと同じコンテンツ、MessageId、および Timestamp を保持します。
- メッセージには、リプレイされたメッセージであることを示す Replayed 属性が含まれています。
- 特定のメッセージのみをリプレイするには、サブスクリプションにフィルターポリシーを適用します。

メッセージのフィルタリングの詳細については、「[リプレイされたメッセージをフィルタリングする](#)」を参照してください。

を使用してメッセージ再生ポリシーを作成する 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する新しいリプレイポリシーを作成するには、このオプションを使用します。

1. [Amazon SNS コンソール](#)にサインインします。
2. トピックのサブスクリプションを選択するか、新しいトピックを作成できます。サブスクリプションの作成の詳細については、「[Amazon SNS トピックのサブスクリプションの作成](#)」を参照してください。

3. メッセージのリプレイを開始するには、[再生] ドロップダウンに移動し、[再生を開始] を選択します。
4. [リプレイ期間] モーダルから、次の選択を行います。
 - a. リプレイ開始日時を選択 — アーカイブされたメッセージのリプレイを開始する日付 (YYYY/MM/DD 形式) と時刻 (24 時間の hh:mm:ss 形式) を選択します。開始時刻は、おおよそのアーカイブ時間の開始時刻よりも遅くする必要があります。
 - b. (オプション) リプレイ終了日時を選択 — アーカイブされたメッセージのリプレイを停止する日付 (YYYY/MM/DD 形式) と時刻 (24 時間の hh:mm:ss 形式) を選択します。
 - c. [Start replay] (再生を開始) を選択します。
5. (オプション) メッセージのリプレイを停止するには、[サブスクリプションの詳細] ページに移動し、[リプレイ] ドロップダウンから [リプレイ停止] を選択します。
6. (オプション) CloudWatch を使用してこのワークフロー内からメッセージリプレイメトリクスをモニタリングするには、「[Amazon CloudWatch を使用してメッセージリプレイメトリクスをモニタリングする](#)」を参照してください。

メッセージリプレイポリシーを表示および編集するには

[サブスクリプションの詳細] ページでは、次のアクションを実行できます。

- メッセージリプレイステータスについては、[再生ステータス] フィールドに表示される以下の値を確認します。
 - 完了済み — リプレイによりすべてのメッセージが正常に再配信され、現在、新しく発行されたメッセージが配信されています。
 - 進行中 — 現在、選択したメッセージがリプレイ中です。
 - 失敗 — リプレイを完了できませんでした。
 - 保留中 — リプレイ開始時のデフォルト状態です。
- (オプション) メッセージリプレイポリシーを変更するには、[サブスクリプションの詳細] ページに移動し、[リプレイ] ドロップダウンから [リプレイ開始] を選択します。リプレイを開始すると、既存のリプレイが置き換えられます。

API を使用してサブスクリプションにリプレイポリシーを追加する

アーカイブされたメッセージをリプレイするには、`ReplayPolicy` 属性を使用します。`ReplayPolicy` は、`Subscribe` および `SetSubscriptionAttributes` API アクションで使用できます。このポリシーには以下の値があります。

- **StartingPoint** (必須) — メッセージのリプレイを開始する場所を通知します。
- **EndingPoint** (オプション) — メッセージのリプレイを停止するタイミングを通知します。`EndingPoint` を省略すると、現在の時刻に追いつくまでリプレイが続行されます。
- **PointType** (必須) — 開始ポイントと終了ポイントの種類を設定します。現在、`PointType` でサポートされている値は `Timestamp` です。

たとえば、ダウンストリームの障害から回復し、2023 年 10 月 1 日に 2 時間にわたってすべてのメッセージを再送信するには、`SetSubscriptionAttributes` API アクションを使用して `ReplayPolicy` を次のように設定します。

```
{
  "PointType": "Timestamp",
  "StartingPoint": "2023-10-01T10:00:00.000Z",
  "EndingPoint": "2023-10-01T12:00:00.000Z"
}
```

2023 年 10 月 1 日時点でトピックに送信されたすべてのメッセージをリプレイし、そのトピックに新しく発行されたすべてのメッセージを引き続き受信するには、`SetSubscriptionAttributes` API アクションを使用してサブスクリプションで `ReplayPolicy` を次のように設定します。

```
{
  "PointType": "Timestamp",
  "StartingPoint": "2023-10-01T00:00:00.000Z"
}
```

メッセージがリプレイされたことを確認するため、リプレイされた各メッセージに `boolean` 属性 `Replayed` が追加されます。

SDK を使用してサブスクリプションにリプレイポリシーを追加する

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、[SDK config および ツールリファレンスガイドの「共有ファイル」および credentials「ファイル」](#) を参照してください。AWS SDKs

次のコード例は、Amazon SNS FIFO トピックのアーカイブからのメッセージについて、2023 年 10 月 1 日に 2 時間の時間ウィンドウで再配信するようにサブスクリプションで `ReplayPolicy` を設定する方法を示しています。

```
// Specify the ARN of the Amazon SNS subscription to initiate the ReplayPolicy on.
String subscriptionArn =
    "arn:aws:sns:us-
    east-2:123456789012:MyArchiveTopic.fifo:1d2a3e9d-7f2f-447c-88ae-03f1c68294da";

// Set the ReplayPolicy to replay messages from the topic's archive
// for a 2 hour time period on October 1st 2023 between 10am and 12pm UTC.
String replayPolicy =
    "{\"PointType\":\"Timestamp\",\"StartingPoint\":\"2023-10-01T10:00:00.000Z\",
    \"EndingPoint\":\"2023-10-01T12:00:00.000Z\"}";

// Set the ArchivePolicy for the Amazon SNS topic
SetSubscriptionAttributesRequest request = new SetSubscriptionAttributesRequest()
    .withSubscriptionArn(subscriptionArn)
    .withAttributeName("ReplayPolicy")
    .withAttributeValue(replayPolicy);
sns.setSubscriptionAttributes(request);
```

EndingPoint について

`ReplayPolicy` を Amazon SNS サブスクリプションに適用する場合、`EndingPoint` の値はオプションです。`EndingPoint` が指定されていない場合、リプレイは指定された `StartingPoint` から開始され、新しく発行されたメッセージの処理を含め、現在の時刻になるまで続行されます。現在の時刻にキャッチアップすると、サブスクリプションは通常のサブスクリプションとして機能し、発行された新しいメッセージを受信します。

`EndingPoint` が指定されている場合、サービスは `StartingPoint` から `EndingPoint` までのメッセージをリプレイして停止します。このアクションによってサブスクリプションは事実上一時停止します。サブスクリプションが一時停止している間、新しく発行されたメッセージは、サブスクライブされたエンドポイントに配信されません。

メッセージ配信を再開するには、`EndingPoint` を指定せずに新しい `ReplayPolicy` を適用し、`StartingPoint` をメッセージの受信を継続する時点に設定します。例えば、以前のリプレイが終了したところからサブスクリプションを再開するには、新しい `StartingPoint` を以前に提供された `EndingPoint` に設定します。

リプレイされたメッセージをフィルタリングする

Amazon SNS メッセージフィルタリングにより、Amazon SNS がサブスクライバーエンドポイントにリプレイするリプレイメッセージを制御できます。メッセージフィルタリングとメッセージアーカイブの両方が有効になっている場合、Amazon SNS は最初にトピックのデータストアからメッセージを取得し、次にサブスクリプションの `FilterPolicy` に対してメッセージを適用します。メッセージは、一致した場合は、サブスクライブされたエンドポイントに配信され、それ以外の場合は、除外されます。詳細については、「[Amazon SNS サブスクリプションフィルターポリシー](#)」を参照してください。

Amazon CloudWatch を使用してメッセージリプレイメトリクスをモニタリングする

Amazon CloudWatch では、以下のメトリクスを使用してリプレイされたメッセージをモニタリングできます。ワークロードの異常を通知し、影響を回避するのに役立てるために、これらのメトリクスに Amazon CloudWatch アラームを設定できます。詳細については、[Amazon SNS でのログ記録とモニタリング](#)を参照してください。

メトリクス	説明
<code>NumberOfReplayedNotificationsDelivered</code>	1 分の解像度で、トピックアーカイブからリプレイされたメッセージの総数をサブスクライバーに提供します。
<code>NumberOfReplayedNotificationsFailed</code>	1 分の解像度で、リプレイされ、トピックアーカイブからの配信に失敗したメッセージの総数をサブスクライバーに提供します。

FIFO トピックの Amazon SNS コード例

以下のコード例を使用して、[自動車部品価格管理のユースケース例](#)を Amazon SNS FIFO トピック、および Amazon SQS FIFO キューまたは標準キューのいずれかと統合します。

AWS SDK の使用

AWS SDK を使用して、`FifoTopic` 属性を `true` に設定して Amazon SNS FIFO トピックを作成します **true**。 `FifoQueue` 属性を `true` に設定することにより、Amazon SQS FIFO キューを作成します また、**.fifo** 接尾辞を各 FIFO リソースの名前に追加する必要があります。FIFO トピックまたはキューを作成した後は、それを標準トピックまたはキューに変換することはできません。

以下のコード例では、これらの FIFO および標準キューリソースを作成します。

- 価格更新を配信する Amazon SNS FIFO トピック
- これらの更新を卸売アプリケーションと小売アプリケーションに提供する Amazon SQS FIFO キュー
- ビジネスインテリジェンス (BI) のクエリが可能なレコードを格納する分析アプリケーション用の Amazon SQS 標準キュー
- 3 つのキューをトピックに接続する Amazon SNS FIFO サブスクリプション

この例では、サブスクリプションで[フィルターポリシー](#)を設定します。トピックにメッセージを発行することで、サンプルをテストする場合は、メッセージは必ず `business` 属性で発行してください。retail または wholesale のいずれかの属性値を指定します。それ以外を指定すると、メッセージは除外され、サブスクライブされたキューに配信されません。詳細については、「[SNS FIFO トピックの Amazon SNS メッセージフィルター処理](#)」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。

この例では

- 1 つの Amazon SNS FIFO トピック、2 つの Amazon SQS FIFO キュー、および 1 つの標準キューを作成します。
- キューをトピックにサブスクライブし、メッセージをトピックに発行します。

[テスト](#)では、各キューへのメッセージの受信を検証します。[完全な例](#)では、アクセスポリシーの追加と、最後にリソースの削除も示しています。

```
public class PriceUpdateExample {
    public final static SnsClient snsClient = SnsClient.create();
    public final static SqsClient sqsClient = SqsClient.create();

    public static void main(String[] args) {
```

```

        final String usage = "\n" +
            "Usage: " +
            "    <topicName> <wholesaleQueueFifoName> <retailQueueFifoName> <analyticsQueueName>\n\n" +
            "Where:\n" +
            "    fifoTopicName - The name of the FIFO topic that you want to create. \n\n" +
            "    wholesaleQueueARN - The name of a SQS FIFO queue that will be created for the wholesale consumer. \n\n"
            +
            "    retailQueueARN - The name of a SQS FIFO queue that will be created for the retail consumer. \n\n" +
            "    analyticsQueueARN - The name of a SQS standard queue that will be created for the analytics consumer. \n\n";
        if (args.length != 4) {
            System.out.println(usage);
            System.exit(1);
        }

        final String fifoTopicName = args[0];
        final String wholeSaleQueueName = args[1];
        final String retailQueueName = args[2];
        final String analyticsQueueName = args[3];

        // For convenience, the QueueData class holds metadata about a queue:
        // ARN, URL,
        // name and type.
        List<QueueData> queues = List.of(
            new QueueData(wholeSaleQueueName, QueueType.FIFO),
            new QueueData(retailQueueName, QueueType.FIFO),
            new QueueData(analyticsQueueName, QueueType.Standard));

        // Create queues.
        createQueues(queues);

        // Create a topic.
        String topicARN = createFIFOTopic(fifoTopicName);

        // Subscribe each queue to the topic.
        subscribeQueues(queues, topicARN);

        // Allow the newly created topic to send messages to the queues.
        addAccessPolicyToQueuesFINAL(queues, topicARN);

```

```
// Publish a sample price update message with payload.
publishPriceUpdate(topicARN, "{\"product\": 214, \"price\": 79.99}",
"Consumables");

// Clean up resources.
deleteSubscriptions(queues);
deleteQueues(queues);
deleteTopic(topicARN);
}

public static String createFIFOTopic(String topicName) {
    try {
        // Create a FIFO topic by using the SNS service client.
        Map<String, String> topicAttributes = Map.of(
            "FifoTopic", "true",
            "ContentBasedDeduplication", "false",
            "FifoThroughputScope", "MessageGroup");

        CreateTopicRequest topicRequest = CreateTopicRequest.builder()
            .name(topicName)
            .attributes(topicAttributes)
            .build();

        CreateTopicResponse response = snsClient.createTopic(topicRequest);
        String topicArn = response.topicArn();
        System.out.println("The topic ARN is" + topicArn);

        return topicArn;

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

public static void subscribeQueues(List<QueueData> queues, String topicARN) {
    queues.forEach(queue -> {
        SubscribeRequest subscribeRequest = SubscribeRequest.builder()
            .topicArn(topicARN)
            .endpoint(queue.queueARN)
            .protocol("sqs")
            .build();
```



```
        // Subscribe to the endpoint by using the SNS service client.
        // Only Amazon SQS queues can receive notifications from an Amazon
SNS FIFO
        // topic.
        SubscribeResponse subscribeResponse =
snsClient.subscribe(subscribeRequest);
        System.out.println("The queue [" + queue.queueARN + "] subscribed to
the topic [" + topicARN + "]");
        queue.subscriptionARN = subscribeResponse.subscriptionArn();
    });
}

    public static void publishPriceUpdate(String topicArn, String payload, String
groupId) {

        try {
            // Create and publish a message that updates the wholesale price.
            String subject = "Price Update";
            String dedupId = UUID.randomUUID().toString();
            String attributeName = "business";
            String attributeValue = "wholesale";

            MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
                .dataType("String")
                .stringValue(attributeValue)
                .build();

            Map<String, MessageAttributeValue> attributes = new HashMap<>();
            attributes.put(attributeName, msgAttValue);
            PublishRequest pubRequest = PublishRequest.builder()
                .topicArn(topicArn)
                .subject(subject)
                .message(payload)
                .messageGroupId(groupId)
                .messageDeduplicationId(dedupId)
                .messageAttributes(attributes)
                .build();

            final PublishResponse response = snsClient.publish(pubRequest);
            System.out.println(response.messageId());
            System.out.println(response.sequenceNumber());
            System.out.println("Message was published to " + topicArn);
        }
    }
}
```

```
    } catch (SnsException e) {  
        System.err.println(e.awsErrorDetails().errorMessage());  
        System.exit(1);  
    }  
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
 - [CreateTopic](#)
 - [Publish](#)
 - [Subscribe](#)

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。

FIFO トピックを作成し、そのトピックに Amazon SQS FIFO キューと標準キューをサブスクライブして、メッセージを Amazon SNS トピックに発行します。

```
def usage_demo():  
    """Shows how to subscribe queues to a FIFO topic."""  
    print("-" * 88)  
    print("Welcome to the `Subscribe queues to a FIFO topic` demo!")  
    print("-" * 88)  
  
    sns = boto3.resource("sns")  
    sqs = boto3.resource("sqs")  
    fifo_topic_wrapper = FifoTopicWrapper(sns)  
    sns_wrapper = SnsWrapper(sns)  
  
    prefix = "sqs-subscribe-demo-"  
    queues = set()  
    subscriptions = set()
```

```
wholesale_queue = sqs.create_queue(
    QueueName=prefix + "wholesale.fifo",
    Attributes={
        "MaximumMessageSize": str(4096),
        "ReceiveMessageWaitTimeSeconds": str(10),
        "VisibilityTimeout": str(300),
        "FifoQueue": str(True),
        "ContentBasedDeduplication": str(True),
    },
)
queues.add(wholesale_queue)
print(f"Created FIFO queue with URL: {wholesale_queue.url}.")

retail_queue = sqs.create_queue(
    QueueName=prefix + "retail.fifo",
    Attributes={
        "MaximumMessageSize": str(4096),
        "ReceiveMessageWaitTimeSeconds": str(10),
        "VisibilityTimeout": str(300),
        "FifoQueue": str(True),
        "ContentBasedDeduplication": str(True),
    },
)
queues.add(retail_queue)
print(f"Created FIFO queue with URL: {retail_queue.url}.")

analytics_queue = sqs.create_queue(QueueName=prefix + "analytics",
Attributes={})
queues.add(analytics_queue)
print(f"Created standard queue with URL: {analytics_queue.url}.")

topic = fifo_topic_wrapper.create_fifo_topic("price-updates-topic.fifo")
print(f"Created FIFO topic: {topic.attributes['TopicArn']}.")

for q in queues:
    fifo_topic_wrapper.add_access_policy(q, topic.attributes["TopicArn"])

print(f"Added access policies for topic: {topic.attributes['TopicArn']}.")

for q in queues:
    sub = fifo_topic_wrapper.subscribe_queue_to_topic(
        topic, q.attributes["QueueArn"]
    )
```

```
        subscriptions.add(sub)

    print(f"Subscribed queues to topic: {topic.attributes['TopicArn']}.")

    input("Press Enter to publish a message to the topic.")

    message_id = fifo_topic_wrapper.publish_price_update(
        topic, '{"product": 214, "price": 79.99}', "Consumables"
    )

    print(f"Published price update with message ID: {message_id}.")

    # Clean up the subscriptions, queues, and topic.
    input("Press Enter to clean up resources.")
    for s in subscriptions:
        sns_wrapper.delete_subscription(s)

    sns_wrapper.delete_topic(topic)

    for q in queues:
        fifo_topic_wrapper.delete_queue(q)

    print(f"Deleted subscriptions, queues, and topic.")

    print("Thanks for watching!")
    print("-" * 88)

class FifoTopicWrapper:
    """Encapsulates Amazon SNS FIFO topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def create_fifo_topic(self, topic_name):
        """
        Create a FIFO topic.
        Topic names must be made up of only uppercase and lowercase ASCII
        letters,
```

numbers, underscores, and hyphens, and must be between 1 and 256 characters long.

For a FIFO topic, the name must end with the `.fifo` suffix.

```
:param topic_name: The name for the topic.
:return: The new topic.
"""
try:
    topic = self.sns_resource.create_topic(
        Name=topic_name,
        Attributes={
            "FifoTopic": str(True),
            "ContentBasedDeduplication": str(False),
            "FifoThroughputScope": "MessageGroup",
        },
    )
    logger.info("Created FIFO topic with name=%s.", topic_name)
    return topic
except ClientError as error:
    logger.exception("Couldn't create topic with name=%s!", topic_name)
    raise error
```

```
@staticmethod
def add_access_policy(queue, topic_arn):
    """
    Add the necessary access policy to a queue, so
    it can receive messages from a topic.

    :param queue: The queue resource.
    :param topic_arn: The ARN of the topic.
    :return: None.
    """
    try:
        queue.set_attributes(
            Attributes={
                "Policy": json.dumps(
                    {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Sid": "test-sid",
                                "Effect": "Allow",
                                "Principal": {"AWS": "*"},

```

```

        "Action": "SQS:SendMessage",
        "Resource": queue.attributes["QueueArn"],
        "Condition": {
            "ArnLike": {"aws:SourceArn": topic_arn}
        },
    ],
    ],
)
)
logger.info("Added trust policy to the queue.")
except ClientError as error:
    logger.exception("Couldn't add trust policy to the queue!")
    raise error

@staticmethod
def subscribe_queue_to_topic(topic, queue_arn):
    """
    Subscribe a queue to a topic.

    :param topic: The topic resource.
    :param queue_arn: The ARN of the queue.
    :return: The subscription resource.
    """
    try:
        subscription = topic.subscribe(
            Protocol="sqs",
            Endpoint=queue_arn,
        )
        logger.info("The queue is subscribed to the topic.")
        return subscription
    except ClientError as error:
        logger.exception("Couldn't subscribe queue to topic!")
        raise error

@staticmethod
def publish_price_update(topic, payload, group_id):
    """
    Compose and publish a message that updates the wholesale price.

    :param topic: The topic to publish to.

```

```

        :param payload: The message to publish.
        :param group_id: The group ID for the message.
        :return: The ID of the message.
        """
        try:
            att_dict = {"business": {"DataType": "String", "StringValue":
"wholesale"}}
            dedup_id = uuid.uuid4()
            response = topic.publish(
                Subject="Price Update",
                Message=payload,
                MessageAttributes=att_dict,
                MessageGroupId=group_id,
                MessageDeduplicationId=str(dedup_id),
            )
            message_id = response["MessageId"]
            logger.info("Published message to topic %s.", topic.arn)
        except ClientError as error:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise error
        return message_id

    @staticmethod
    def delete_queue(queue):
        """
        Removes an SQS queue. When run against an AWS account, it can take up to
        60 seconds before the queue is actually deleted.

        :param queue: The queue to delete.
        :return: None
        """
        try:
            queue.delete()
            logger.info("Deleted queue with URL=%s.", queue.url)
        except ClientError as error:
            logger.exception("Couldn't delete queue with URL=%s!", queue.url)
            raise error

```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
 - [CreateTopic](#)
 - [Publish](#)
 - [Subscribe](#)

SAP ABAP

SDK for SAP ABAP

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。

FIFO トピックを作成し、そのトピックに Amazon SQS FIFO キューをサブスクライブして、Amazon SNS トピックにメッセージを発行します。

```
" Creates a FIFO topic. "
DATA lt_tpc_attributes TYPE /aws1/
cl_snstopicattrsm_w=>tt_topicattributesmap.
DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsm_w=>ts_topicattributesmap_maprow.
ls_tpc_attributes-key = 'FifoTopic'.
ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsm_w( iv_value =
'true' ).
INSERT ls_tpc_attributes INTO TABLE lt_tpc_attributes.

TRY.
  DATA(lo_create_result) = lo_sns->createtopic(
    iv_name = iv_topic_name
    it_attributes = lt_tpc_attributes ).
  DATA(lv_topic_arn) = lo_create_result->get_topicarn( ).
  ov_topic_arn = lv_topic_arn.
  "
  ov_topic_arn is returned for testing purposes. "
  MESSAGE 'FIFO topic created' TYPE 'I'.
CATCH /aws1/cx_snstopiclimitexcdex.
```



```

    MESSAGE 'Unable to create more topics. You have reached the maximum
number of topics allowed.' TYPE 'E'.
    ENDTRY.

    " Subscribes an endpoint to an Amazon Simple Notification Service (Amazon
SNS) topic. "
    " Only Amazon Simple Queue Service (Amazon SQS) FIFO queues can be subscribed
to an SNS FIFO topic. "
    TRY.
        DATA(lo_subscribe_result) = lo_sns->subscribe(
            iv_topicarn = lv_topic_arn
            iv_protocol = 'sqs'
            iv_endpoint = iv_queue_arn ).
        DATA(lv_subscription_arn) = lo_subscribe_result->get_subscriptionarn( ).
        ov_subscription_arn = lv_subscription_arn.
    "
    ov_subscription_arn is returned for testing purposes. "
    MESSAGE 'SQS queue was subscribed to SNS topic.' TYPE 'I'.
    CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
    CATCH /aws1/cx_snssubscriptionlmt00.
    MESSAGE 'Unable to create subscriptions. You have reached the maximum
number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.

    " Publish message to SNS topic. "
    TRY.
        DATA lt_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>tt_messageattributemap.
        DATA ls_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>ts_messageattributemap_maprow.
        ls_msg_attributes-key = 'Importance'.
        ls_msg_attributes-value = NEW /aws1/cl_snsmessageattrvalue( iv_datatype =
'String'

iv_stringvalue = 'High' ).
        INSERT ls_msg_attributes INTO TABLE lt_msg_attributes.

        DATA(lo_result) = lo_sns->publish(
            iv_topicarn = lv_topic_arn
            iv_message = 'The price of your mobile plan has been increased from
$19 to $23'
            iv_subject = 'Changes to mobile plan'
            iv_messagegroupid = 'Update-2'
            iv_messagededuplicationid = 'Update-2.1'

```

```

        it_messageattributes = lt_msg_attributes ).
        ov_message_id = lo_result->get_messageid( ).
        "
        ov_message_id is returned for testing purposes. "
        MESSAGE 'Message was published to SNS topic.' TYPE 'I'.
        CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    ENDTRY.

```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の以下のトピックを参照してください。
 - [CreateTopic](#)
 - [Publish](#)
 - [Subscribe](#)

FIFO サブスクリプションからメッセージを受信する

3 つのサブスクライブされたアプリケーションで価格の更新を受け取ることができるようになりました。に示すように[the section called “FIFO トピックのユースケース”](#)、各コンシューマーアプリケーションのエントリポイントは Amazon SQS キューであり、対応する AWS Lambda 関数が自動的にポーリングできます。Amazon SQS キューが Lambda 関数のイベントソースである場合、Lambda はメッセージを効率的に使用するために、必要に応じてポーラーのフリートをスケールします。

詳細については、「AWS Lambda デベロッパーガイド」の[Amazon SQS AWS Lambda での使用](#)」を参照してください。独自のキューポーラーの書き込み方法については、『Amazon Simple Queue Service デベロッパーガイド』の「[Amazon SQS スタンダードおよび FIFO キューのレコメンデーション](#)」および『Amazon Simple Queue Service API リファレンス』の「[ReceiveMessage](#)」を参照してください。

の使用 AWS CloudFormation

CloudFormation では、テンプレートファイルを使用して、AWS リソースのコレクションを 1 つのユニットとしてまとめて作成および設定できます。このセクションでは、以下を作成するサンプルテンプレートを使用します。

- 価格更新を配信する Amazon SNS FIFO のトピック
- これらの更新を卸売アプリケーションと小売アプリケーションに提供する Amazon SQS FIFO キュー

- ビジネスインテリジェンス (BI) のクエリが可能なレコードを格納する分析アプリケーション用の Amazon SQS 標準キュー
- 3 つのキューをトピックに接続する Amazon SNS FIFO サブスクリプション
- 受信者アプリケーションが必要とする価格の更新のみを受信することを指定する [フィルターポリシー](#)

 Note

トピックにメッセージを発行することで、サンプルをテストする場合は、メッセージは必ず `business` 属性で発行してください。 `retail` または `wholesale` のいずれかの属性値を指定します。それ以外を指定すると、メッセージは除外され、サブスクライブされたキューに配信されません。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Resources": {
    "PriceUpdatesTopic": {
      "Type": "AWS::SNS::Topic",
      "Properties": {
        "TopicName": "PriceUpdatesTopic.fifo",
        "FifoTopic": true,
        "ContentBasedDeduplication": false,
        "ArchivePolicy": {
          "MessageRetentionPeriod": "30"
        }
      }
    },
    "WholesaleQueue": {
      "Type": "AWS::SQS::Queue",
      "Properties": {
        "QueueName": "WholesaleQueue.fifo",
        "FifoQueue": true,
        "ContentBasedDeduplication": false
      }
    },
    "RetailQueue": {
      "Type": "AWS::SQS::Queue",
      "Properties": {
        "QueueName": "RetailQueue.fifo",
```

```
        "FifoQueue": true,
        "ContentBasedDeduplication": false
    },
    "AnalyticsQueue": {
        "Type": "AWS::SQS::Queue",
        "Properties": {
            "QueueName": "AnalyticsQueue"
        }
    },
    "WholesaleSubscription": {
        "Type": "AWS::SNS::Subscription",
        "Properties": {
            "TopicArn": {
                "Ref": "PriceUpdatesTopic"
            },
            "Endpoint": {
                "Fn::GetAtt": [
                    "WholesaleQueue",
                    "Arn"
                ]
            },
            "Protocol": "sqs",
            "RawMessageDelivery": "false",
            "FilterPolicyScope": "MessageBody",
            "FilterPolicy": {
                "business": [
                    "wholesale"
                ]
            }
        }
    },
    "RetailSubscription": {
        "Type": "AWS::SNS::Subscription",
        "Properties": {
            "TopicArn": {
                "Ref": "PriceUpdatesTopic"
            },
            "Endpoint": {
                "Fn::GetAtt": [
                    "RetailQueue",
                    "Arn"
                ]
            }
        }
    },
```

```

    "Protocol": "sqs",
    "RawMessageDelivery": "false",
    "FilterPolicyScope": "MessageBody",
    "FilterPolicy": {
      "business": [
        "retail"
      ]
    }
  },
  "AnalyticsSubscription": {
    "Type": "AWS::SNS::Subscription",
    "Properties": {
      "TopicArn": {
        "Ref": "PriceUpdatesTopic"
      },
      "Endpoint": {
        "Fn::GetAtt": [
          "AnalyticsQueue",
          "Arn"
        ]
      },
    },
    "Protocol": "sqs",
    "RawMessageDelivery": "false"
  },
  "SalesQueuesPolicy": {
    "Type": "AWS::SQS::QueuePolicy",
    "Properties": {
      "PolicyDocument": {
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "sns.amazonaws.com"
            },
            "Action": [
              "sqs:SendMessage"
            ],
            "Resource": "*",
            "Condition": {
              "ArnEquals": {
                "aws:SourceArn": {
                  "Ref": "PriceUpdatesTopic"
                }
              }
            }
          }
        ]
      }
    }
  }
}

```

```
    }
  }
}
]
},
"Queues": [
  {
    "Ref": "WholesaleQueue"
  },
  {
    "Ref": "RetailQueue"
  },
  {
    "Ref": "AnalyticsQueue"
  }
]
}
}
```

CloudFormation テンプレートを使用した AWS リソースのデプロイの詳細については、「CloudFormation ユーザーガイド」の「開始方法<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/GettingStarted.Walkthrough.html>」を参照してください。

Amazon SNS メッセージフィルター処理

デフォルトでは、Amazon SNS トピックの受信者は、トピックに対して発行されたすべてのメッセージを受信します。メッセージのサブセットのみを受信する場合、受信者はトピックのサブスクリプションにフィルターポリシーを割り当てる必要があります。

フィルターポリシーは、受信者が受信するメッセージを定義するプロパティが含まれている JSON オブジェクトです。Amazon SNS は、サブスクリプションに設定したフィルターポリシーの範囲に従って、メッセージ属性またはメッセージ本文に適用されるポリシーに対応します。メッセージ本文のフィルターポリシーは、メッセージペイロードが正しい形式の JSON オブジェクトであることを前提としています。

サブスクリプションにフィルターポリシーがない場合、サブスクライバーは、そのトピックに発行されたすべてのメッセージを受信します。フィルターポリシーを設定してトピックにメッセージを発行すると、Amazon SNS はメッセージ属性またはメッセージ本文をトピックの各サブスクリプションのフィルターポリシー内のプロパティを比較します。すべてのメッセージ属性またはメッセージ本文プロパティがフィルターポリシーで指定された条件を満たす場合、Amazon SNS はメッセージをサブスクライバーに送信します。そうでない場合、Amazon SNS はそのサブスクライバーにメッセージを送信しません。

詳細については、「[トピックに発行されたメッセージのフィルター処理](#)」を参照してください。

Amazon SNS サブスクリプションフィルターポリシーの範囲

FilterPolicyScope サブスクリプション属性を使用すると、次のいずれかの値を設定してフィルタリング範囲を定義できます。

- MessageAttributes – フィルターポリシーをメッセージ属性に適用します (デフォルト設定)。
- MessageBody – フィルターポリシーをメッセージ本文に適用します。

Note

既存のフィルターポリシーに範囲が定義されていない場合、範囲はデフォルトで MessageAttributes に設定されます。

Amazon SNS サブスクリプションフィルターポリシー

サブスクリプションフィルターポリシーを使用すると、プロパティ名を指定して、プロパティ名ごとに値のリストを割り当てることができます。詳細については、「[Amazon SNS メッセージフィルター処理](#)」を参照してください。

Amazon SNS がサブスクリプションフィルターポリシーに照らしてメッセージ属性またはメッセージ本文のプロパティを評価する際、ポリシーで指定されていないものは無視されます。

Important

AWS IAM や Amazon SNS などの サービスは、結果整合性と呼ばれる分散コンピューティングモデルを使用します。サブスクリプションフィルターポリシーへの追加または変更は、完全に有効になるまでに最大 15 分かかります。

サブスクリプションは、以下の条件に該当するメッセージを受け入れます。

- フィルターポリシーの範囲が `MessageAttributes` に設定されている場合、フィルターポリシーの各プロパティ名はメッセージ属性名と一致します。フィルターポリシー内で一致するプロパティ名ごとに、少なくとも 1 つのプロパティ値がメッセージ属性値と一致します。
- フィルターポリシーの範囲が `MessageBody` に設定されている場合、フィルターポリシーの各プロパティ名はメッセージ本文のプロパティ名と一致します。フィルターポリシー内で一致するプロパティ名ごとに、少なくとも 1 つのプロパティ値がメッセージ本文のプロパティ値と一致します。

Amazon SNS は、現在、以下のフィルター演算子をサポートしています。

- [AND ロジック](#)
- [OR ロジック](#)
- [OR 演算子](#)
- [キーの一致](#)
- [数値の完全一致](#)
- [数値の anything-but 一致](#)
- [数値範囲の一致](#)

- [文字列値の完全一致](#)
- [文字列値の anything-but 一致](#)
- [プレフィックスと anything-but 演算子を使用した文字列の一致](#)
- [文字列値の equals-ignore case](#)
- [文字列値の IP アドレス一致](#)
- [文字列値のプレフィックスマッチング](#)
- [文字列値のサフィックスマッチング](#)

Amazon SNS サンプルフィルターポリシー

次の例は、顧客のトランザクションを処理する Amazon SNS トピックから配信されるメッセージペイロードを示しています。

最初の例には、トランザクションを記述する属性がある MessageAttributes フィールドが含まれます。

- 顧客の興味
- ストア名
- イベント状態
- 購入価格 (USD)

このメッセージには MessageAttributes フィールドが含まれるため、サブスクリプションで FilterPolicyScope が MessageAttributes に設定されている限り、FilterPolicy を設定するトピックサブスクリプションは、メッセージを選択的に許可または拒否することができます。メッセージへの属性の適用の詳細については、「[Amazon SNS メッセージ属性](#)」を参照してください。

```
{
  "Type": "Notification",
  "MessageId": "a1b2c34d-567e-8f90-g1h2-i345j67klmn8",
  "TopicArn": "arn:aws:sns:us-east-2:123456789012:MyTopic",
  "Message": "message-body-with-transaction-details",
  "Timestamp": "2019-11-03T23:28:01.631Z",
  "SignatureVersion": "4",
  "Signature": "signature",
  "UnsubscribeURL": "unsubscribe-url",
  "MessageAttributes": {
```

```
    "customer_interests": {
      "Type": "String.Array",
      "Value": "[\"soccer\", \"rugby\", \"hockey\"]"
    },
    "store": {
      "Type": "String",
      "Value": "example_corp"
    },
    "event": {
      "Type": "String",
      "Value": "order_placed"
    },
    "price_usd": {
      "Type": "Number",
      "Value": "210.75"
    }
  }
}
```

次の例は、Message フィールドに含まれ、メッセージペイロードまたはメッセージ本文とも呼ばれる同じ属性を示しています。サブスクリプションで FilterPolicyScope が MessageBody に設定されている限り、FilterPolicy を含むトピックサブスクリプションは、メッセージを選択的に許可または拒否することができます。

```
{
  "Type": "Notification",
  "MessageId": "a1b2c34d-567e-8f90-g1h2-i345j67klmn8",
  "TopicArn": "arn:aws:sns:us-east-2:123456789012:MyTopic",
  "Message": "{
    \"customer_interests\": [\"soccer\", \"rugby\", \"hockey\"],
    \"store\": \"example_corp\",
    \"event\": \"order_placed\",
    \"price_usd\": 210.75
  }",
  "Timestamp": "2019-11-03T23:28:01.631Z",
  "SignatureVersion": "4",
  "Signature": "signature",
  "UnsubscribeURL": "unsubscribe-url"
}
```

以下のフィルターポリシーでは、プロパティの名前と値に基づいてメッセージを許可または拒否します。

メッセージ例を許可するポリシー

以下のサブスクリプションフィルターポリシーのプロパティは、メッセージ例に割り当てられた属性に一致します。MessageAttributes または MessageBody に設定されているかどうかにかかわらず、同じフィルターポリシーが FilterPolicyScope に対して有効であることに注意してください。各サブスクライバーは、トピックから受信するメッセージの構成に従ってフィルタリング範囲を選択します。

このポリシーの 1 つのプロパティが、メッセージに割り当てられた属性と一致しなかった場合、ポリシーはメッセージを拒否します。

```
{
  "store": ["example_corp"],
  "event": [{"anything-but": "order_cancelled"}],
  "customer_interests": [
    "rugby",
    "football",
    "baseball"
  ],
  "price_usd": [{"numeric": [">=", 100]}]
}
```

メッセージ例を拒否するポリシー

以下のサブスクリプションフィルターポリシーでは、そのプロパティと、メッセージ例に割り当てられた属性の間に、複数の不一致があります。例えば、encrypted プロパティ名はメッセージ属性に存在していないため、割り当てられている値にかかわらず、このポリシープロパティによってメッセージは拒否されます。

不一致が発生すると、ポリシーはメッセージを拒否します。

```
{
  "store": ["example_corp"],
  "event": ["order_cancelled"],
  "encrypted": [false],
  "customer_interests": [
    "basketball",
    "baseball"
  ]
}
```

Amazon SNS フィルターポリシーの制約

Amazon SNS でフィルターポリシーを設定する場合、注意すべき重要なルールがいくつかあります。これらのルールは、システムのパフォーマンスと互換性を維持しながら、フィルターポリシーを効果的に適用するのに役立ちます。

共通のポリシーの制約

Amazon SNS でフィルターポリシーを設定するときは、システムのパフォーマンスと互換性を維持しながら効果的に機能するように、次の重要なルールに従ってください。

- 文字列の一致 – フィルターポリシーでの文字の一致ルールでは、大文字と小文字が区別して比較されます。
- 数値マッチング – 数値マッチングでは、値は $-10^9 \sim 10^9$ (-10 億 ~ 10 億) の範囲で、小数点以下 5 桁の精度で指定できます。
- フィルターポリシーの複雑さ — 値の組み合わせの合計が 150 を超えてはなりません。組み合わせの合計を計算するには、フィルターポリシーの各配列の値の数を乗算します。
- キー数の制限 – フィルターポリシーには最大 5 つのキーを含めることができます。

追加の考慮事項

- フィルターポリシーの JSON には以下のものが含まれます。
 - 引用符で囲まれた文字列
 - 数字
 - 引用符なしのキーワード `true`、`false`、および `null`
- Amazon SNS API を使用する場合は、フィルターポリシーの JSON を有効な UTF-8 文字列として渡す必要があります。
- フィルターポリシーの最大サイズは 256 KB です。
- デフォルトでは、トピックごとに最大 200 個のフィルターポリシー、AWS アカウントごとに最大 10,000 個のフィルターポリシーを持つことができます。

このポリシー制限によっても、[Subscribe](#) API で Amazon SQS キューサブスクリプションが作成されるのを止めることはできません。ただし、Subscribe API コール (または

[SetSubscriptionAttributes](#) API コール) でフィルターポリシーをアタッチすると失敗します。

このクォータを引き上げるには、[AWS Service Quotas](#) を使用できます。

属性ベースのフィルタリングのポリシー制約

属性ベースのフィルタリングがデフォルトのオプションです。[FilterPolicyScope](#) は、サブスクリプションでは `MessageAttributes` に設定されています。

- Amazon SNS は属性ベースのフィルタリングに対して、ネストされたフィルターポリシーを受け入れません。
- Amazon SNS は、以下のデータ型に該当するメッセージ属性とのみポリシーのプロパティを比較します。
 - `String`
 - `String.Array`

Important

Amazon SNS で属性ベースのフィルタリングを使用する場合は、特定の特殊文字、特に次の文字を二重エスケープする必要があります。

- 二重引用符 (`"`)
- バックスラッシュ (`\`)

これらの文字を二重エスケープしないと、フィルターポリシーが公開されたメッセージの属性と一致なくなり、通知は配信されません。

追加の考慮事項

- オブジェクトを配列で渡すことはお勧めしません。属性ベースのフィルタリングではサポートされていないネストが原因で、予期しない結果が発生する可能性があるためです。ネストされたポリシーにペイロードベースのフィルタリングを使用する
- `Number` は数値の属性値でサポートされています。
- Amazon SNS は、バイナリデータ型のメッセージ属性を無視します。

複雑さに関するポリシーの例:

次のポリシーの例では、最初のキーには 3 つの一致演算子、2 番目のキーには 1 つの一致演算子、3 番目のキーには 2 つの一致演算子があります。

```
{
  "key_a": ["value_one", "value_two", "value_three"],
  "key_b": ["value_one"],
  "key_c": ["value_one", "value_two"]
}
```

組み合わせの合計数は、フィルターポリシーに含まれる各キーの一致演算子の数の積として計算されます。

```
3(match operators of key_a)
x 1(match operators of key_b)
x 2(match operators of key_c)
= 6
```

ペイロードベースのフィルタリングのポリシー制約

属性ベース (デフォルト) のフィルタリングからペイロードベースのフィルタリングに切り替えるには、サブスクリプションで [FilterPolicyScope](#) を MessageBody に設定する必要があります。

- Amazon SNS は、ペイロードベースのフィルタリングに対してネストされたフィルターポリシーを受け入れます。
- ネストされたポリシーの場合、リーフキーのみが 5 個のキー制限の対象としてカウントされます。

キー制限に関するポリシーの例:

次のポリシーの例について説明します。

- リーフキーには、key_c と key_e の 2 つがあります。
- key_c にはネストされたレベルが 3 の一致演算子が 4 つあり、key_e にはネストされたレベルが 2 の一致演算子が 3 つがあります。

```
{
  "key_a": {
    "key_b": {
```

```
    "key_c": ["value_one", "value_two", "value_three", "value_four"]
  },
  "key_d": {
    "key_e": ["value_one", "value_two", "value_three"]
  }
}
```

組み合わせの合計数は、フィルターポリシーに含まれる各キーの一致演算子の数とネストされたレベルの積として計算されます。

```
4(match operators of key_c)
x 3(nested level of key_c)
x 3(match operators of key_e)
x 2(nested level of key_e)
= 72
```

ワイルドカードパターンの使用に関するガイドライン

Amazon SQS は、ワイルドカードを含むフィルターポリシーを登録する際の保護機能を実装しており、複雑すぎるフィルターポリシーが作成されないようにしています。こうしたフィルターポリシーは、アプリケーションのパフォーマンスに影響を与えるためです。

パターン構造

フィールドには 1 つ以上のパターンが含まれます。次の例は、2 つのパターンを使用するフィールドを示しています。

```
{
  "greeting": [
    {"anything-but": {"prefix": "Hello"}},
    {"wildcard": "H*"}
  ] // 2 patterns
}
```

複雑さのルール

- すべてのフィールドにおけるワイルドカードの複雑さの合計は、100 ポイントを超えることはできません
- パターンあたりのワイルドカードの最大数は 3 つです

複雑さの計算

- フィールドの複雑さ = (Sum of pattern points) × (Number of patterns)
- パターンポイント:

単一のワイルドカード: 1 ポイント

複数のワイルドカード: 各 3 ポイント

anything-but ワイルドカード: 1 ポイント

複雑さの計算例を次に示します。

```
{
  "filename": [
    {"wildcard": "*.txt"},      // 1 point
    {"wildcard": "log*"}       // 1 point
  ]                             // Total: (1 + 1) × 2 = 4 points
}
```

AND/OR ロジック

フィルターポリシーで AND/OR ロジックを使用し、Amazon SNS でメッセージ属性または本文プロパティを照合する方法について説明します。これにより、より正確で柔軟なメッセージフィルタリングが可能になります。

AND ロジック

複数のプロパティ名を使用して AND ロジックを適用できます。

次のポリシーについて考えます。

```
{
  "customer_interests": ["rugby"],
  "price_usd": [{"numeric": [ ">", 100]}]
}
```

このポリシーは、customer_interests の値が rugby に設定され、さらに price_usd の値が 100 を超える値に設定されているメッセージ属性またはメッセージ本文プロパティと一致します。

Note

AND ロジックを同じ属性の値に適用することはできません。

OR ロジック

プロパティ名に複数の値を割り当てることで OR ロジックを適用できます。

次のポリシーについて考えます。

```
{
  "customer_interests": ["rugby", "football", "baseball"]
}
```

このポリシー属性は、customer_interests の値が rugby、football、または baseball に設定されているメッセージ属性またはメッセージ本文プロパティと一致します。

OR 演算子

"\$or" 演算子を使用してフィルターポリシーを明示的に定義し、ポリシー内の複数の属性間の OR 関係を表すことができます。

Amazon SNS は、ポリシーが以下の条件をすべて満たしている場合にのみ "\$or" 関係を認識します。これらの条件がすべて満たされない場合、"\$or" はポリシー内の他の文字列と同様に通常の属性名として扱われます。

- ルールには "\$or" フィールド属性が付けられ、その後に配列 ("\$or" : [] など) が続きます。
- "\$or" 配列には少なくとも次の 2 つのオブジェクトがあります: "\$or": [{}, {}]。
- "\$or" 配列内のどのオブジェクトにも、予約キーワードであるフィールド名はありません。

それ以外の場合は、ポリシー内の他の文字列と同様に、"\$or" は通常の属性名として扱われます。

数値とプレフィックスは予約キーワードであるため、以下のポリシーは OR 関係として解析されません。

```
{
  "$or": [ {"numeric" : 123}, {"prefix": "abc"} ]
}
```

OR 演算子の例

標準の OR:

```
{
  "source": [ "aws.cloudwatch" ],
  "$or": [
    { "metricName": [ "CPUUtilization" ] },
    { "namespace": [ "AWS/EC2" ] }
  ]
}
```

このポリシーのフィルターロジックは次のとおりです。

```
"source" && ("metricName" || "namespace")
```

このポリシー属性は、以下のメッセージ属性セットのいずれとも一致します。

```
"source": {"Type": "String", "Value": "aws.cloudwatch"},
"metricName": {"Type": "String", "Value": "CPUUtilization"}
```

または

```
"source": {"Type": "String", "Value": "aws.cloudwatch"},
"namespace": {"Type": "String", "Value": "AWS/EC2"}
```

以下のメッセージ本文のいずれとも一致します。

```
{
  "source": "aws.cloudwatch",
  "metricName": "CPUUtilization"
}
```

または

```
{
  "source": "aws.cloudwatch",
  "namespace": "AWS/EC2"
}
```

OR 関係を含むポリシー制約

次のポリシーについて考えます。

```
{
  "source": [ "aws.cloudwatch" ],
  "$or": [
    { "metricName": [ "CPUUtilization", "ReadLatency" ] },
    {
      "metricType": [ "MetricType" ] ,
      "$or" : [
        { "metricId": [ 1234, 4321 ] },
        { "spaceId": [ 1000, 2000, 3000 ] }
      ]
    }
  ]
}
```

このポリシーのロジックは次のように簡略化することもできます。

```
("source" AND "metricName")
OR
("source" AND "metricType" AND "metricId")
OR
("source" AND "metricType" AND "spaceId")
```

OR 関係のあるポリシーの複雑さの計算は、各 OR ステートメントの組み合わせの複雑さの合計として簡略化できます。

組み合わせの合計は次のように計算されます。

```
(source * metricName) + (source * metricType * metricId) + (source * metricType *
spaceId)
= (1 * 2) + (1 * 1 * 2) + (1 * 1 * 3)
= 7
```

source には 1 つの値、metricName には 2 つの値、metricType には 1 つの値、metricId には 2 つの値、spaceId には 3 つの値があります。

次のネストされたフィルターポリシーについて考えてみます。

```
{
```

```

"$or": [
  { "metricName": [ "CPUUtilization", "ReadLatency" ] },
  { "namespace": [ "AWS/EC2", "AWS/ES" ] }
],
"detail" : {
  "scope" : [ "Service" ],
  "$or": [
    { "source": [ "aws.cloudwatch" ] },
    { "type": [ "CloudWatch Alarm State Change" ] }
  ]
}
}

```

このポリシーのロジックは次のように簡略化できます。

```

("metricName" AND ("detail"."scope" AND "detail"."source"))
OR
("metricName" AND ("detail"."scope" AND "detail"."type"))
OR
("namespace" AND ("detail"."scope" AND "detail"."source"))
OR
("namespace" AND ("detail"."scope" AND "detail"."type"))

```

組み合わせの合計の計算は、キーのネストレベルを考慮する必要がある点を除いて、ネストされていないポリシーでも同じです。

組み合わせの合計は次のように計算されます。

$$(2 * 2 * 2) + (2 * 2 * 2) + (2 * 2 * 2) + (2 * 2 * 2) = 32$$

metricName には 2 つの値、namespace には 2 つの値があります。scope は、1 つの値を持つ 2 レベルのネストされたキー、source は 1 つの値を持つ 2 レベルのネストされたキー、type は 1 つの値を持つ 2 レベルのネストされたキーです。

キーの一致

フィルターポリシーで exists 演算子を使用して、特定のプロパティが存在するか存在しないかに基づいて受信メッセージを照合します。

- exists はリーフノード (構造の末端属性) でのみ機能します。
- ネストされた JSON 構造内の中間ノードには適用されません。

- "exists": true を使用して、指定されたプロパティを含む受信メッセージと一致します。キーには、null でなく、空でない値が必要です。

例えば、次のポリシーのプロパティは、true の値の exists 演算子を使用します。

```
"store": [{"exists": true}]
```

次のように、store 属性キーを含むすべてのメッセージ属性のリストに一致します。

```
"store": {"Type": "String", "Value": "fans"}  
"customer_interests": {"Type": "String.Array", "Value": "[\"baseball\", \"basketball\"]"}
```

以下のメッセージ本文のいずれとも一致します。

```
{  
  "store": "fans"  
  "customer_interests": ["baseball", "basketball"]  
}
```

ただし、次のように、store 属性キーを持たないメッセージ属性の一覧とは一致しません。

```
"customer_interests": {"Type": "String.Array", "Value": "[\"baseball\", \"basketball\"]"}
```

また、次のメッセージ本文にも一致しません。

```
{  
  "customer_interests": ["baseball", "basketball"]  
}
```

- "exists": false を使用して、指定されたプロパティを含まない受信メッセージを一致します。

Note

"exists": false は少なくとも 1 つの属性が存在する場合にのみ一致します。属性のセットが空の場合、フィルターは一致しません。

例えば、次のポリシーのプロパティは、`false` の値の `exists` 演算子を使用します。

```
"store": [{"exists": false}]
```

次のように、`store` 属性キーを含むすべてのメッセージ属性のリストに一致しません。

```
"store": {"Type": "String", "Value": "fans"}  
"customer_interests": {"Type": "String.Array", "Value": "[\"baseball\", \"basketball\"]"}
```

次のメッセージ本文にも一致しません。

```
{  
  "store": "fans"  
  "customer_interests": ["baseball", "basketball"]  
}
```

ただし、次のように、`store` 属性キーを持たないメッセージ属性の一覧とは一致します。

```
"customer_interests": {"Type": "String.Array", "Value": "[\"baseball\", \"basketball\"]"}
```

以下のメッセージ本文にも一致します。

```
{  
  "customer_interests": ["baseball", "basketball"]  
}
```

数値の一致

数値をメッセージ属性値またはメッセージ本文のプロパティ値に一致させることで、メッセージをフィルタリングできます。JSON ポリシーでは、数値を二重引用符で囲みません。フィルターには次の数値オペレーションを使用できます。

 Note

プレフィックスは、文字列一致のみがサポートされます。

完全一致

ポリシープロパティ値に `numeric` キーワードと演算子 `=` が含まれている場合、同じ名前と等しい数値を持つ任意のメッセージ属性またはメッセージ本文プロパティ値と一致します。

次のポリシーのプロパティについて考えます。

```
"price_usd": [{"numeric": ["=", 301.5]}]
```

このポリシー属性は、以下のメッセージ属性のいずれとも一致します。

```
"price_usd": {"Type": "Number", "Value": 301.5}
```

```
"price_usd": {"Type": "Number", "Value": 3.015e2}
```

以下のメッセージ本文のいずれとも一致します。

```
{  
  "price_usd": 301.5  
}
```

```
{  
  "price_usd": 3.015e2  
}
```

「以外」のマッチング

ポリシープロパティ値にキーワード `anything-but` が含まれている場合、そのキーワードは、ポリシープロパティ値を含まない任意のメッセージ属性またはメッセージ本文プロパティ値と一致します。

次のポリシーのプロパティについて考えます。

```
"price": [{"anything-but": [100, 500]}]
```

このポリシー属性は、以下のメッセージ属性のいずれとも一致します。

```
"price": {"Type": "Number", "Value": 101}
```

```
"price": {"Type": "Number", "Value": 100.1}
```

以下のメッセージ本文のいずれとも一致します。

```
{  
  "price": 101  
}
```

```
{  
  "price": 100.1  
}
```

さらに、次のメッセージ属性にも一致します (これは 100 でも 500 でもない値を含むためです)。

```
"price": {"Type": "Number.Array", "Value": "[100, 50]"}
```

また、次のメッセージ本文にも一致します (これは 100 でも 500 でもない値が含まれるためです)。

```
{  
  "price": [100, 50]  
}
```

ただし、次のメッセージ属性とは一致しません。

```
"price": {"Type": "Number", "Value": 100}
```

また、次のメッセージ本文にも一致しません。

```
{  
  "price": 100  
}
```


値範囲の一致

数値ポリシープロパティには、= 演算子に加えて、次の <、<=、>、および >= を含めることができます。

次のポリシーのプロパティについて考えます。

```
"price_usd": [{"numeric": ["<", 0]}]
```

負の数値を持つ任意のメッセージ属性またはメッセージ本文プロパティと一致します。

別のポリシー属性について考えます。

```
"price_usd": [{"numeric": [ ">", 0, "<=", 150 ]}]
```

最大 150 までの正の数を持つメッセージ属性またはメッセージ本文プロパティと一致します。

文字列値の一致

文字列値をメッセージ属性値またはメッセージ本文のプロパティ値に一致させることで、メッセージをフィルタリングできます。JSON ポリシーでは、文字列値を二重引用符で囲みます。次の文字列オペレーションを使用して、メッセージ属性またはメッセージ本文のプロパティと一致させることができます。

完全一致

完全一致は、ポリシープロパティ値が 1 つ以上のメッセージ属性値と一致した場合に発生します。String.Array 型属性の場合、配列内の各要素は、マッチングのために個別の文字列として扱われます。

次のポリシーのプロパティについて考えます。

```
"customer_interests": ["rugby", "tennis"]
```

このポリシー属性は、以下のメッセージ属性と一致します。

```
"customer_interests": {"Type": "String", "Value": "rugby"}
```

```
"customer_interests": {"Type": "String", "Value": "tennis"}
```

```
"customer_interests": {"Type": "String.Array", "Value": "[\"rugby\", \"tennis\"]"}
```

また、次のメッセージ本文にも一致します。

```
{  
  "customer_interests": "rugby"  
}
```

```
{  
  "customer_interests": "tennis"  
}
```

ただし、次のメッセージ属性とは一致しません。

```
"customer_interests": {"Type": "String", "Value": "baseball"}
```

```
"customer_interests": {"Type": "String.Array", "Value": "[\"baseball\"]"}
```

また、次のメッセージ本文にも一致しません。

```
{  
  "customer_interests": "baseball"  
}
```

「以外」のマッチング

ポリシープロパティ値にキーワード `anything-but` が含まれている場合、そのキーワードは、ポリシープロパティ値を含まない任意のメッセージ属性またはメッセージ本文値と一致します。`anything-but` は、`"exists": false` と組み合わせることができます。`String.Array` 型属性の場合、配列のどの要素もポリシープロパティに含まれていなければ、このマッチングは一致します。

次のポリシーのプロパティについて考えます。

```
"customer_interests": [{"anything-but": ["rugby", "tennis"]}]
```

このポリシー属性は、次のメッセージ属性と一致します。

```
"customer_interests": {"Type": "String", "Value": "baseball"}
```

```
"customer_interests": {"Type": "String", "Value": "football"}
```

```
"customer_interests": {"Type": "String.Array", "Value": "[\"rugby\", \"baseball\"]"}
```

以下のメッセージ本文のいずれとも一致します。

```
{  
  "customer_interests": "baseball"  
}
```

```
{  
  "customer_interests": "football"  
}
```

さらに、次のメッセージ属性にも一致します (これは rugby でも tennis でもない値を含むためです)。

```
"customer_interests": {"Type": "String.Array", "Value": "[\"rugby\", \"baseball\"]"}
```

また、次のメッセージ本文にも一致します (これには rugby でも tennis でもない値が含まれるためです)。

```
{  
  "customer_interests": ["rugby", "baseball"]  
}
```

ただし、次のメッセージ属性とは一致しません。

```
"customer_interests": {"Type": "String", "Value": "rugby"}
```

```
"customer_interests": {"Type": "String.Array", "Value": "[\"rugby\"]"}
```

また、次のメッセージ本文にも一致しません。

```
{
```

```
"customer_interests": ["rugby"]
}
```

anything-but でプレフィックスを使用する

文字列一致では、anything-but 演算子と共にプレフィックスを使用することもできます。例えば、次のポリシープロパティが order- プレフィックスを拒否します。

```
"event": [{"anything-but": {"prefix": "order-"}}]
```

このポリシー属性は、以下の属性のいずれとも一致します。

```
"event": {"Type": "String", "Value": "data-entry"}
```

```
"event": {"Type": "String", "Value": "order_number"}
```

以下のメッセージ本文のいずれとも一致します。

```
{
  "event": "data-entry"
}
```

```
{
  "event": "order_number"
}
```

ただし、次のメッセージ属性とは一致しません。

```
"event": {"Type": "String", "Value": "order-cancelled"}
```

また、次のメッセージ本文にも一致しません。

```
{
  "event": "order-cancelled"
}
```

anything-but ワイルドカード

次のポリシープロパティは、*ball ワイルドカードを拒否します。

```
"customer_interests" : [{ "anything-but": { "wildcard": "*ball" } }]
```

このポリシー属性は、次の属性と一致します。

```
{"customer_interests": ["hockey", "rugby", "soccer"] }
```

ただし、次のメッセージ属性とは一致しません。

```
{"customer_interests": ["baseball", "basketball"] }
```

anything-but サフィックス

次のポリシープロパティは、-ball

サフィックスを拒否します。

```
"customer_interests": [ { "anything-but": { "suffix": "ball" } } ]
```

このポリシー属性は、次の属性と一致します。

```
{"customer_interests": ["hockey", "rugby", "soccer"] }
```

ただし、次のメッセージ属性とは一致しません。

```
{"customer_interests": ["baseball", "basketball"] }
```

Equals-ignore-case マッチング

ポリシープロパティに equals-ignore-case キーワードが含まれている場合、任意のメッセージ属性または本文プロパティ値を使用して、大文字と小文字を区別しない一致が実行されます。

次のポリシーのプロパティについて考えます。

```
"customer_interests": [{"equals-ignore-case": "tennis"}]
```

このポリシー属性は、以下のメッセージ属性のいずれとも一致します。

```
"customer_interests": {"Type": "String", "Value": "TENNIS"}
```

```
"customer_interests": {"Type": "String", "Value": "Tennis"}
```

以下のメッセージ本文のいずれとも一致します。

```
{  
  "customer_interests": "TENNIS"  
}
```

```
{  
  "customer_interests": "teNnis"  
}
```

IP アドレスマッチング

`cidr` 演算子を使用して、受信メッセージが特定の IP アドレスまたはサブネットから発信されているかどうかを確認できます。

次のポリシーのプロパティについて考えます。

```
"source_ip": [{"cidr": "10.0.0.0/24"}]
```

このポリシー属性は、以下のメッセージ属性のいずれとも一致します。

```
"source_ip": {"Type": "String", "Value": "10.0.0.0"}
```

```
"source_ip": {"Type": "String", "Value": "10.0.0.255"}
```

以下のメッセージ本文のいずれとも一致します。

```
{  
  "source_ip": "10.0.0.0"  
}
```

```
{  
  "source_ip": "10.0.0.255"  
}
```

ただし、次のメッセージ属性とは一致しません。

```
"source_ip": {"Type": "String", "Value": "10.1.1.0"}
```

また、次のメッセージ本文にも一致しません。

```
{  
  "source_ip": "10.1.1.0"  
}
```

プレフィックスマッチング

ポリシープロパティに `prefix` キーワードが含まれている場合、指定した文字で始まる任意のメッセージ属性または本文プロパティ値と一致します。

次のポリシーのプロパティについて考えます。

```
"customer_interests": [{"prefix": "bas"}]
```

このポリシー属性は、以下のメッセージ属性のいずれとも一致します。

```
"customer_interests": {"Type": "String", "Value": "baseball"}
```

```
"customer_interests": {"Type": "String", "Value": "basketball"}
```

以下のメッセージ本文のいずれとも一致します。

```
{  
  "customer_interests": "baseball"  
}
```

```
{  
  "customer_interests": "basketball"  
}
```

ただし、次のメッセージ属性とは一致しません。

```
"customer_interests": {"Type": "String", "Value": "rugby"}
```

また、次のメッセージ本文にも一致しません。

```
{
  "customer_interests": "rugby"
}
```

サフィックスマッチング

ポリシープロパティに `suffix` キーワードが含まれている場合、指定した文字で終わる任意のメッセージ属性または本文プロパティ値と一致します。

次のポリシーのプロパティについて考えます。

```
"customer_interests": [{"suffix": "ball"}]
```

このポリシー属性は、以下のメッセージ属性のいずれとも一致します。

```
"customer_interests": {"Type": "String", "Value": "baseball"}
```

```
"customer_interests": {"Type": "String", "Value": "basketball"}
```

以下のメッセージ本文のいずれとも一致します。

```
{
  "customer_interests": "baseball"
}
```

```
{
  "customer_interests": "basketball"
}
```

ただし、次のメッセージ属性とは一致しません。

```
"customer_interests": {"Type": "String", "Value": "rugby"}
```

また、次のメッセージ本文にも一致しません。

```
{
  "customer_interests": "rugby"
}
```



```
}
```

ワイルドカード

ワイルドカード文字 (*) を使用してイベントパターン内の文字列値と一致させることができます。

次のポリシーでは、ワイルドカード (*) 文字を使用しています。

```
"customer_interests": [ { "wildcard": "*ball" } ]
```

このポリシー属性は、次の属性と一致します。

```
{"customer_interests": ["baseball", "basketball"] }
```

Amazon SNS でサブスクリプションフィルターポリシーを適用する

Amazon SNS でメッセージフィルタリングを使用すると、フィルターポリシーに基づいてサブスクライバーにメッセージを選択的に配信できます。これらのポリシーは、メッセージがサブスクリプションに配信されるために満たす必要がある条件を定義します。raw メッセージ配信はメッセージ処理に影響を与える可能性があるオプションですが、サブスクリプションフィルターが機能するために必須ではありません。

Amazon SNS コンソールを使用して、Amazon SNS サブスクリプションにフィルターポリシーを適用できます。または、プログラムでポリシーを適用するには、Amazon SNS API、AWS Command Line Interface (AWS CLI)、または Amazon SNS をサポートする任意の AWS SDK を使用できます。Amazon SNS を使用することもできます AWS CloudFormation。

raw メッセージ配信の有効化

raw メッセージ配信では、追加のエンコーディングや変換を行わずに、メッセージペイロードがそのままサブスクライバーに配信されます。これは、サブスクライバーが処理のために元のメッセージ形式を必要とする場合に役立ちます。ただし、raw メッセージ配信は、サブスクリプションフィルターの機能とは直接関係ありません。

サブスクリプションフィルターの適用

メッセージフィルターをサブスクリプションに適用するには、JSON 構文を使用してフィルターポリシーを定義します。このポリシーは、サブスクリプションに配信されるためにメッセージが満たす必

要がある条件を指定します。フィルターは、メッセージ属性、メッセージ構造、メッセージコンテンツなどのメッセージ属性に基づいて設定できます。

raw メッセージ配信とサブスクリプションフィルターの関係

raw メッセージ配信を有効にすると、サブスクライバーによるメッセージの配信と処理方法に影響を与える可能性があります。サブスクリプションフィルターを使用するための前提条件ではありません。ただし、サブスクライバーが変更されていない元のメッセージ形式を必要とするシナリオでは、raw メッセージ配信の有効化は、サブスクリプションフィルターとあわせて役立つ場合があります。

効果的なフィルタリングに関する考慮事項

メッセージフィルタリングを実装する場合は、アプリケーションとサブスクライバーの特定の要件を考慮してください。メッセージ配信の基準に正確に一致するフィルターポリシーを定義して、効率的でターゲットを絞ったメッセージ配信を確実に行います。

Important

AWS IAM や Amazon SNS などの サービスは、結果整合性と呼ばれる分散コンピューティングモデルを使用します。サブスクリプションフィルターポリシーへの追加または変更は、完全に有効になるまでに最大 15 分かかります。

AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[サブスクリプション] を選択します。
3. サブスクリプションを選択したら、[編集] を選択します。
4. [Edit](編集) ページで、[Subscription filter policy](サブスクリプションフィルターポリシーセクション) を展開します。
5. 属性ベースのフィルタリングまたはペイロードベースのフィルタリングのいずれかを選択します。
6. [JSON editor](JSON エディタ) フィールドで、フィルターポリシーの [JSON body](JSON 本文) を提供します。
7. [変更を保存] をクリックします。

Amazon SNS により、フィルターポリシーがサブスクリプションに適用されます。

AWS CLI

AWS Command Line Interface (AWS CLI) でフィルターポリシーを適用するには、次の例に示すように [set-subscription-attributes](#) コマンドを使用します。--attribute-name オプションで、FilterPolicy を指定します。--attribute-value で、JSON ポリシーを指定します。

```
$ aws sns set-subscription-attributes --subscription-arn arn:aws:sns: ... --  
attribute-name FilterPolicy --attribute-value '{"store":["example_corp"],"event":  
["order_placed"]}'
```

ポリシーに有効な JSON を提供するには、属性名と値を二重引用符で囲みます。また、ポリシーの引数全体を引用符で囲む必要があります。引用符のエスケープを避けるため、上に示した例のように、一重引用符を使用してポリシーを囲み、二重引用符を使用して JSON 名と値を囲みます。

メッセージフィルタリングを属性ベース (デフォルト) からペイロードベースに切り替える場合は、[set-subscription-attributes](#) コマンドも使用できます。--attribute-name オプションで、FilterPolicyScope を指定します。--attribute-value の場合、MessageBody を指定します。

```
$ aws sns set-subscription-attributes --subscription-arn arn:aws:sns: ... --attribute-  
name FilterPolicyScope --attribute-value MessageBody
```

フィルターポリシーが適用されたことを確認するには、get-subscription-attributes コマンドを使用します。ターミナル出力の属性には、以下の例に示すように、FilterPolicy キーのフィルターポリシーが表示されます。

```
$ aws sns get-subscription-attributes --subscription-arn arn:aws:sns: ...  
{  
  "Attributes": {  
    "Endpoint": "endpoint . . .",  
    "Protocol": "https",  
    "RawMessageDelivery": "false",  
    "EffectiveDeliveryPolicy": "delivery policy . . .",  
    "ConfirmationWasAuthenticated": "true",  
    "FilterPolicy": "{\"store\": [\"example_corp\"], \"event\": [\"order_placed  
\" ]}",  
    "FilterPolicyScope": "MessageAttributes",  
    "Owner": "111122223333",  
    "SubscriptionArn": "arn:aws:sns: . . .",  
    "TopicArn": "arn:aws:sns: . . ."
```

```
}  
}
```

AWS SDKs

次のサンプルコードは、SetSubscriptionAttributes を使用する方法を説明しています。

Important

SDK for Java 2.x の例を使用している場合、クラス SNSMessageFilterPolicy はそのままでは使用できません。このクラスをインストールする方法については、GitHub ウェブサイトの[例](#)を参照してください。

CLI

AWS CLI

サブスクリプション属性を設定するには

次の set-subscription-attributes の例では、RawMessageDelivery 属性を SQS サブスクリプションに設定します。

```
aws sns set-subscription-attributes \  
  --subscription-arn arn:aws:sns:us-east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \  
  --attribute-name RawMessageDelivery \  
  --attribute-value true
```

このコマンドでは何も出力されません。

次の set-subscription-attributes の例では、FilterPolicy 属性を SQS サブスクリプションに設定します。

```
aws sns set-subscription-attributes \  
  --subscription-arn arn:aws:sns:us-east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \  
  --attribute-name FilterPolicy \  
  --attribute-value "{ \"anyMandatoryKey\": [\"any\", \"of\", \"these\"] }"
```

このコマンドでは何も出力されません。

次の `set-subscription-attributes` の例では、`FilterPolicy` 属性を SQS サブスクリプションから削除します。

```
aws sns set-subscription-attributes \  
  --subscription-arn arn:aws:sns:us-east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \  
  --attribute-name FilterPolicy \  
  --attribute-value "{}"
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[SetSubscriptionAttributes](#)」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。

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sns.SnsClient;  
import software.amazon.awssdk.services.sns.model.SnsException;  
import java.util.ArrayList;  
  
/**  
 * 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  
 */  
public class UseMessageFilterPolicy {  
    public static void main(String[] args) {  
        final String usage = ""
```

```
Usage:    <subscriptionArn>

Where:
    subscriptionArn - The ARN of a subscription.

""";

if (args.length != 1) {
    System.out.println(usage);
    System.exit(1);
}

String subscriptionArn = args[0];
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();

usePolicy(snsClient, subscriptionArn);
snsClient.close();
}

public static void usePolicy(SnsClient snsClient, String subscriptionArn) {
    try {
        SNSMessageFilterPolicy fp = new SNSMessageFilterPolicy();
        // Add a filter policy attribute with a single value
        fp.addAttribute("store", "example_corp");
        fp.addAttribute("event", "order_placed");

        // Add a prefix attribute
        fp.addAttributePrefix("customer_interests", "bas");

        // Add an anything-but attribute
        fp.addAttributeAnythingBut("customer_interests", "baseball");

        // Add a filter policy attribute with a list of values
        ArrayList<String> attributeValues = new ArrayList<>();
        attributeValues.add("rugby");
        attributeValues.add("soccer");
        attributeValues.add("hockey");
        fp.addAttribute("customer_interests", attributeValues);

        // Add a numeric attribute
        fp.addAttribute("price_usd", "=", 0);
    }
}
```

```
// Add a numeric attribute with a range
fp.addAttributeRange("price_usd", ">", 0, "<=", 100);

// Apply the filter policy attributes to an Amazon SNS subscription
fp.apply(snsClient, subscriptionArn);

} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[SetSubscriptionAttributes](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def add_subscription_filter(subscription, attributes):
        """
```

Adds a filter policy to a subscription. A filter policy is a key and a list of values that are allowed. When a message is published, it must have an attribute that passes the filter or it will not be sent to the subscription.

```
:param subscription: The subscription the filter policy is attached to.
:param attributes: A dictionary of key-value pairs that define the
filter.
"""
try:
    att_policy = {key: [value] for key, value in attributes.items()}
    subscription.set_attributes(
        AttributeName="FilterPolicy",
        AttributeValue=json.dumps(att_policy)
    )
    logger.info("Added filter to subscription %s.", subscription.arn)
except ClientError:
    logger.exception(
        "Couldn't add filter to subscription %s.", subscription.arn
    )
    raise
```

- APIの詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[SetSubscriptionAttributes](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.
    lo_sns->setsubscriptionattributes(
        iv_subscriptionarn = iv_subscription_arn
```



```
        iv_attributename  = 'FilterPolicy'
        iv_attributevalue = iv_filter_policy ).
    MESSAGE 'Added filter policy to subscription.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Subscription does not exist.' TYPE 'E'.
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[SetSubscriptionAttributes](#)」を参照してください。

Amazon SNS API

Amazon SNS API を使用してフィルターポリシーを適用するに

は、[SetSubscriptionAttributes](#) アクションへのリクエストを作成します。AttributeName パラメータを FilterPolicy に設定し、AttributeValue パラメータをフィルターポリシーの JSON に設定します。

メッセージフィルタリングを属性ベース (デフォルト) からペイロードベースのメッセージフィルタリングに切り替える場合は、[SetSubscriptionAttributes](#) アクションも使用できます。AttributeName パラメータを FilterPolicyScope に設定し、AttributeValue パラメータを MessageBody に設定します。

AWS CloudFormation

を使用してフィルターポリシーを適用するには CloudFormation、JSON または YAML テンプレートを使用してスタックを作成します CloudFormation。詳細については、AWS CloudFormation ユーザーガイドの AWS::SNS::Subscription リソースの [FilterPolicy](#) プロパティと [サンプル CloudFormation テンプレート](#) を参照してください。

1. [CloudFormation コンソール](#) にサインインします。
2. [スタックの作成] を選択します。
3. [テンプレートの選択] ページで、[テンプレートを Amazon S3 にアップロード] を選択してから、ファイルを選択して [次へ] をクリックします。
4. [詳細の指定] ページで、以下の作業を行います。
 - a. [スタックの名前] に「MyFilterPolicyStack」と入力します。

- b. [myHttpEndpoint] に、トピックにサブスクライブする HTTP エンドポイントを入力します。

 Tip

HTTP エンドポイントがない場合は作成します。

5. [オプション] ページで、[次へ] をクリックします。
6. [Review] ページで、[作成] を選択します。

Amazon SNS のサブスクリプションフィルターポリシーを削除する

サブスクリプションに送信されるメッセージのフィルター処理を停止するには、サブスクリプションのフィルターポリシーを空の JSON 本文に上書きすることで削除します。このポリシーを削除したら、サブスクリプションは発行されるすべてのメッセージを受け取るようになります。

の使用 AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[サブスクリプション] を選択します。
3. サブスクリプションを選択したら、[編集] を選択します。
4. [Edit **EXAMPLE1-23bc-4567-d890-ef12g3hij456**] ページで、[サブスクリプションフィルターポリシー] セクションを展開します。
5. [JSON エディタ] フィールドで、フィルターポリシーの空の JSON 本文を指定します ({}).
6. [変更を保存] をクリックします。

Amazon SNS により、フィルターポリシーがサブスクリプションに適用されます。

の使用 AWS CLI

でフィルターポリシーを削除するには AWS CLI、[set-subscription-attributes](#) コマンドを使用して --attribute-value 引数に空の JSON 本文を指定します。

```
$ aws sns set-subscription-attributes --subscription-arn arn:aws:sns: ... --attribute-name FilterPolicy --attribute-value "{}"
```

Amazon SNS API の使用

Amazon SNS API を使用してフィルターポリシーを削除するには、[SetSubscriptionAttributes](#) アクションへのリクエストを行います。AttributeName パラメータを FilterPolicy に設定し、AttributeValue パラメータに空の JSON 本文を指定します。

Amazon SNS でのメッセージデータ保護

メッセージデータ保護とは

メッセージデータ保護は、[データ保護ポリシー](#)を使用してアプリケーションまたは AWS サービス間を移動する機密情報を監査、マスク、編集、またはブロックすることで、Amazon SNS トピックに発行されるデータを保護します。

メッセージデータ保護は、データ識別子を使って、個人を特定できる情報 (PII) および保護医療情報 (PHI) に関する移動中のデータをスキャンします。[事前定義された](#) (または Amazon SNS マネージド) データ識別子 (名前、住所、クレジットカード番号、処方薬コードなど) の使用を選択するか、ビジネスユースケースに固有の独自の[カスタム](#)データ識別子を作成できます。メッセージデータ保護は、スキャンした情報を使用して詳細な監査ログを提供するため、データを保護するための措置を講じることができます。

メッセージデータ保護は、機密性の高い顧客情報を保護するために以下のアクションをサポートしています。

- **監査** — Amazon SNS トピックに発行されたデータの最大 99% を監査します。その後、検出結果を [Amazon CloudWatch](#)、[Amazon S3](#)、または <https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html> のいずれかに送信できます。
- **識別解除** — メッセージの発行や配信を中断することなく、機密データをマスクまたは削除できます。
- **拒否** — ペイロード内に機密データが存在する場合、アプリケーションと AWS リソース間のデータ送信をブロックします。

Note

Amazon SNS は Amazon SNS 標準トピックのみのメッセージデータ保護をサポートします。

メッセージデータ保護を使用すべき理由

ガバナンス、リスク管理、コンプライアンスプログラムにメッセージデータ保護を導入することで、データ漏洩の特定と防止に役立つデータ保護ポリシーを実装できます。これによ

り、HIPAA、GDPR、PCI、FedRAMP などのプライバシー規制に準拠することで、財務、法律、規制上のリスクを軽減するのに役立つツールをチームに提供できます。また、開発者は、機密データを保護するための独自のツールの構築および管理に伴う運用上のオーバーヘッドから解放されます。

たとえば、メッセージデータ保護を使用して監査ポリシーを作成し、誤って機密データを送受信しているシステムがないかどうかを確認できます。監査結果から、システムがクレジットカード情報を必要としないシステムに送信していることが明らかになった場合は、ブロックポリシーを使用してデータの配信を防止できます。

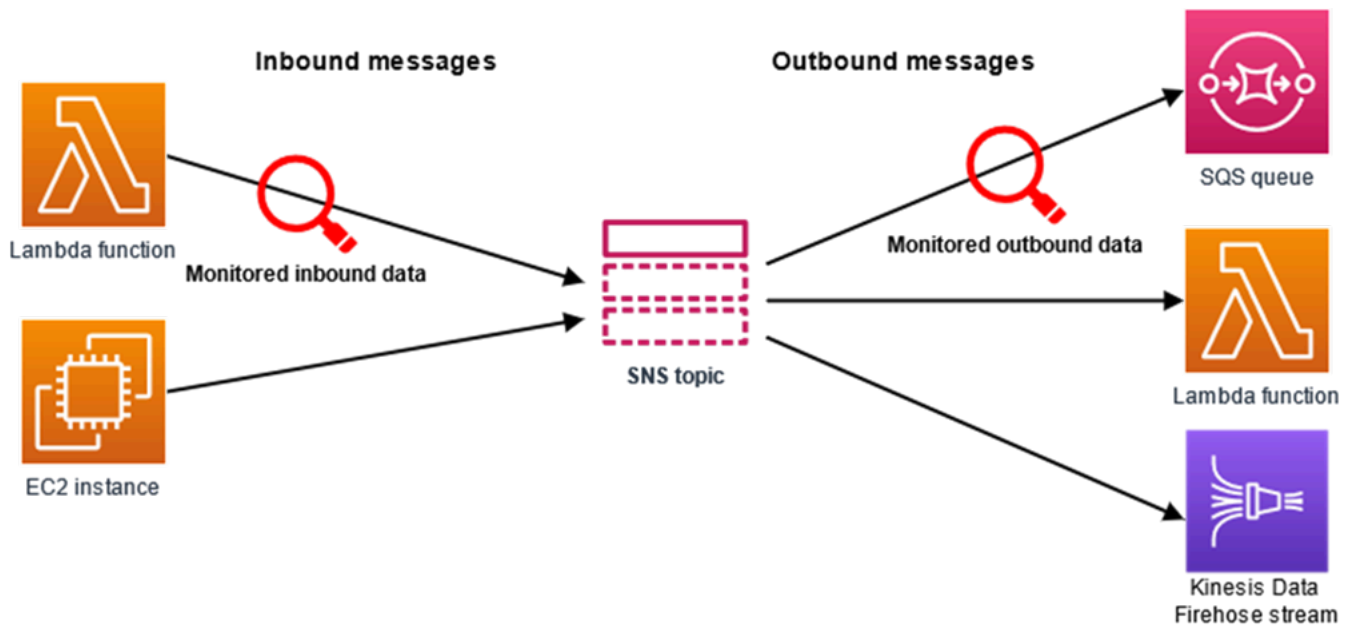
Note

Amazon SNS は Amazon SNS 標準トピックのみのメッセージデータ保護をサポートします。

Amazon SNS データ保護ポリシーの理解

データ保護ポリシーとは

Amazon SNS はデータ保護ポリシーを使って、スキャンする機密データと、そのデータが Amazon SNS トピックで交換されないように保護するアクションを選択します。目的の機密データを選択するには、[データ識別子](#)を使用します。次に、Amazon SNS メッセージデータ保護は、機械学習とパターンマッチングを使用して機密データを検出します。見つかったデータ識別子に基づいて、監査、識別解除、または拒否のオペレーションを定義できます。これらのオペレーションにより、見つかった (または見つからなかった) 機密データをログに記録、機密データをマスクまたは編集、またはメッセージの配信を拒否できます。

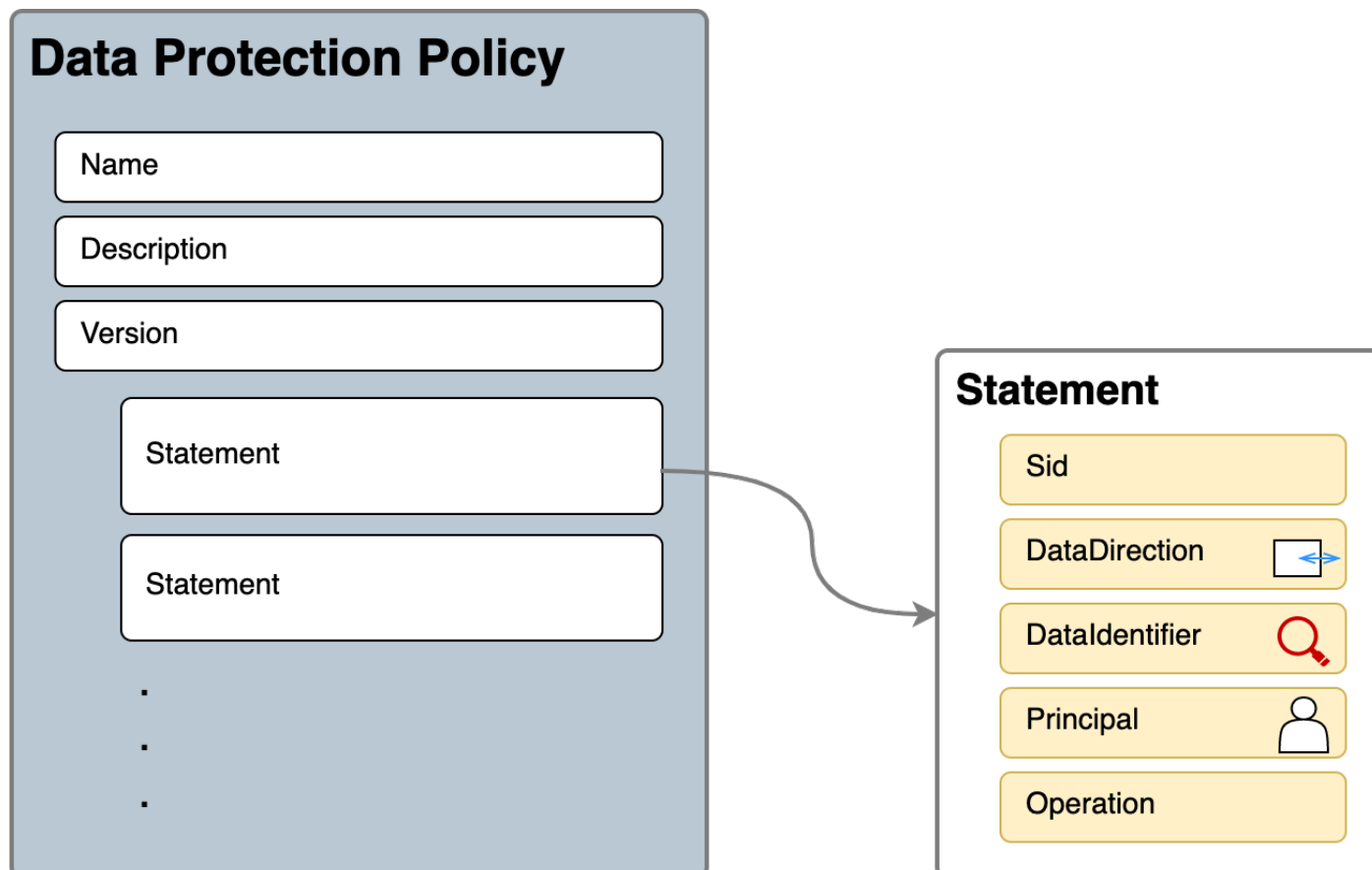


データ保護ポリシーの構成の仕組み

次の図に示すように、データ保護ポリシードキュメントには次の要素が含まれています。

- ドキュメントの最上部に記載されるポリシー全体の情報 (任意)
- 1 つ以上の個別のステートメント

各ステートメントには、1 つのアクセス許可に関する情報が含まれています。



Amazon SNS トピックごとに定義できるデータ保護ポリシーは 1 つだけです。データ保護ポリシーには、1 つまたは複数の拒否または識別解除ステートメントと 1 つの監査ステートメントのみを含めることができます。

データ保護ポリシーの JSON プロパティ

データ保護ポリシーでは、識別のために以下の基本ポリシー情報が必要です。

- Name – ポリシーの名前。
- Description (オプション) – ポリシーの説明。
- Version – ポリシー言語のバージョン。現在のバージョンは 2021-06-01. です。
- Statement – データ保護ポリシーアクションを指定するステートメントのリスト。

```
{
  "Name": "basicPII-protection",
  "Description": "Protect basic types of sensitive data",
  "Version": "2021-06-01",
```

```
"Statement": [  
    ...  
]  
}
```

ポリシーステートメントの JSON プロパティ

ポリシーステートメントは、データ保護オペレーションの検出コンテキストを設定します。

- [Sid] (オプション) — ステートメントの識別子。
- [DataDirection] (データ方向) — Amazon SNS トピックに関するインバウンド (API リクエストの発行) またはアウトバウンド (通知配信)。
- [DataIdentifier] (データ識別子) — Amazon SNS トピックがスキャンすべき機密データ。名前、住所、電話番号などです。
- [Principal] — トピックを発行した IAM プリンシパル、またはトピックにサブスクライブされた IAM プリンシパル。
- [Operation] (オペレーション) — 機密データが見つかった Amazon SNS トピックが実行する、[Audit] (監査)、[De-identify] (マスクまたは編集)、または [Deny] (拒否) (ブロック) のいずれかである後続のアクション。

```
{  
    "Sid": "basicPII-inbound-protection",  
    "DataDirection": "Inbound",  
    "Principal": ["*"],  
    "DataIdentifier": [  
        "arn:aws:dataprotection::aws:data-identifier/Name",  
        "arn:aws:dataprotection::aws:data-identifier/PhoneNumber-US"  
    ],  
    "Operation": {  
        ...  
    }  
}
```

ポリシーステートメントオペレーションの JSON プロパティ

ポリシーステートメントは、次のデータ保護オペレーションのいずれかを設定します。

- [監査](#) – メッセージの発行や配信を中断することなく、メトリクスと検出結果ログを出力します。

- [識別解除](#) — メッセージの公開を中断することなく、機密データをマスク、または編集できます。
- [拒否](#) — Amazon SNS 発行リクエストをブロックするか、メッセージの配信を失敗させます。

データ保護ポリシーの IAM プリンシパルを決定する方法

メッセージデータ保護では、Amazon SNS とやり取りする 2 つの IAM プリンシパルを使用します。

1. [Publish API Principal] (API プリンシパルの発行) (インバウンド) — Amazon SNS Publish API を呼び出す認証済みの IAM プリンシパル。
2. [Subscription Principal] (サブスクリプションプリンシパル) (アウトバウンド) — サブスクリプションの作成中に Subscribe API を呼び出した認証済みの IAM プリンシパル。

SubscriptionPrincipal は公開されている Amazon SNS サブスクリプションプロパティで、GetSubscriptionAttributes API から取得できます。

```
{
  "Attributes": {
    "SubscriptionPrincipal": "arn:aws:iam::123456789012:user/NoNameAccess",
    "Owner": "123412341234",
    "RawMessageDelivery": "true",
    "TopicArn": "arn:aws:sns:us-east-1:123412341234:PII-data-topic",
    "Endpoint": "arn:aws:sqs:us-east-1:123456789012:NoNameAccess",
    "Protocol": "sqs",
    "PendingConfirmation": "false",
    "ConfirmationWasAuthenticated": "true",
    "SubscriptionArn": "arn:aws:sns:us-east-1:123412341234:PII-data-
topic:5d8634ef-67ef-49eb-a824-4042b28d6f55"
  }
}
```

Amazon SNS のデータ保護ポリシーオペレーション

以下は、機密データの監査および拒否に使用できるデータ保護ポリシーの例です。サンプルアプリケーションを含む完全なチュートリアルについては、「[Introducing message data protection for Amazon SNS](#)」(Amazon SNS のメッセージデータ保護の実装) のブログ記事を参照してください。

監査オペレーション

監査オペレーションは、トピックのインバウンドメッセージをサンプリングし、送信 AWS 先の機密データの検出結果をログに記録します。サンプルレートは、0 ~ 99 の整数になります。このオペレーションには、次のいずれかのタイプのロギング先が必要です。

1. [FindingsDestination] — Amazon SNS トピックがペイロードに機密データを見つけたときのロギング先。
2. [NoFindingsDestination] — Amazon SNS トピックがペイロードに機密データを見つけなかったときのロギング先。

次の各ログ送信先タイプ AWS のサービス で以下を使用できます。

- Amazon CloudWatch Logs (オプション) — LogGroup はトピックリージョンにあり、名前が /aws/vendedlogs/ で始まる必要があります。
- (オプション) – DeliveryStream はトピックリージョンにあり、配信ストリームのソースとして Direct PUT が必要です。詳細については、「Amazon Data Firehose デベロッパーガイド」の「[送信元、送信先、および名前](#)」を参照してください。
- Amazon S3 (オプション) — Amazon S3 バケット名。 [SSE-KMS 暗号化を有効にして Amazon S3 バケットを使用するには、追加のアクションが必要です](#)。

```
{
  "Operation": {
    "Audit": {
      "SampleRate": "99",
      "FindingsDestination": {
        "CloudWatchLogs": {
          "LogGroup": "/aws/vendedlogs/log-group-name"
        },
        "Firehose": {
          "DeliveryStream": "delivery-stream-name"
        },
        "S3": {
          "Bucket": "bucket-name"
        }
      },
      "NoFindingsDestination": {
        "CloudWatchLogs": {
          "LogGroup": "/aws/vendedlogs/log-group-name"
        }
      }
    }
  }
}
```

```
    },
    "Firehose": {
      "DeliveryStream": "delivery-stream-name"
    },
    "S3": {
      "Bucket": "bucket-name"
    }
  }
}
```

ログの送信先の指定に必要な権限

データ保護ポリシーでロギング先を指定する場合、Amazon SNS PutDataProtectionPolicy API、または --data-protection-policy パラメータを含む CreateTopic API を呼び出す IAM プリンシパルの IAM ID ポリシーに、次のアクセス権限を追加する必要があります。

監査先	IAM 許可
デフォルト	logs:CreateLogDelivery logs:GetLogDelivery logs:UpdateLogDelivery logs>DeleteLogDelivery logs:ListLogDeliveries
CloudWatchLogs	logs:PutResourcePolicy logs:DescribeResourcePolicies logs:DescribeLogGroups
Firehose	iam:CreateServiceLinkedRole firehose:TagDeliveryStream
S3	s3:PutBucketPolicy s3:GetBucketPolicy

監査先	IAM 許可
	<u>SSE-KMS 暗号化を有効にして Amazon S3 バケットを使用するには、追加のアクションが必要です。</u>

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogDelivery",
        "logs:GetLogDelivery",
        "logs:UpdateLogDelivery",
        "logs>DeleteLogDelivery",
        "logs:ListLogDeliveries"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutResourcePolicy",
        "logs:DescribeResourcePolicies",
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-1:123456789012:SampleLogGroupName:*:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole",
        "firehose:TagDeliveryStream"
      ],
    }
  ]
}
```

```

        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:PutBucketPolicy",
            "s3:GetBucketPolicy"
        ],
        "Resource": [
            "arn:aws:s3:::bucket-name"
        ]
    }
]
}

```

SSE-KMS に使用する必須のキーポリシー

Amazon S3 バケットをログの送信先として使用する場合は、Amazon S3 (S3-Managed) または によるサーバー側の暗号化 AWS KMS keys (SSE-KMS) を有効にすることで、バケット内のデータを保護できます。詳細については、「Amazon S3 ユーザーガイド」の「[サーバー側の暗号化を使用したデータの保護](#)」をご参照ください。

SSE-S3 を選択した場合、追加の設定は必要ありません。Amazon S3 が暗号化キーを処理します。

SSE-KMS を選択した場合は、カスターマネージドキーを使用する必要があります。ログ配信アカウントが S3 バケットに書き込めるように、カスターマネージドキーのキーポリシーを更新する必要があります。SSE-KMS での使用に必要なキーポリシーについては、「Amazon CloudWatch Logs ユーザーガイド」の「[Amazon S3 バケットのサーバー側の暗号化](#)」を参照してください。

監査先ログの例

次の例では、`callerPrincipal` を使用して機密コンテンツのソースを特定します。また、`messageID` を参照として使用し、Publish API レスポンスに対してチェックします。

```

{
  "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",
  "auditTimestamp": "2022-05-12T2:10:44Z",
  "callerPrincipal": "arn:aws:iam::123412341234:role/Publisher",
  "resourceArn": "arn:aws:sns:us-east-1:123412341234:PII-data-topic",
  "dataIdentifiers": [
    {

```

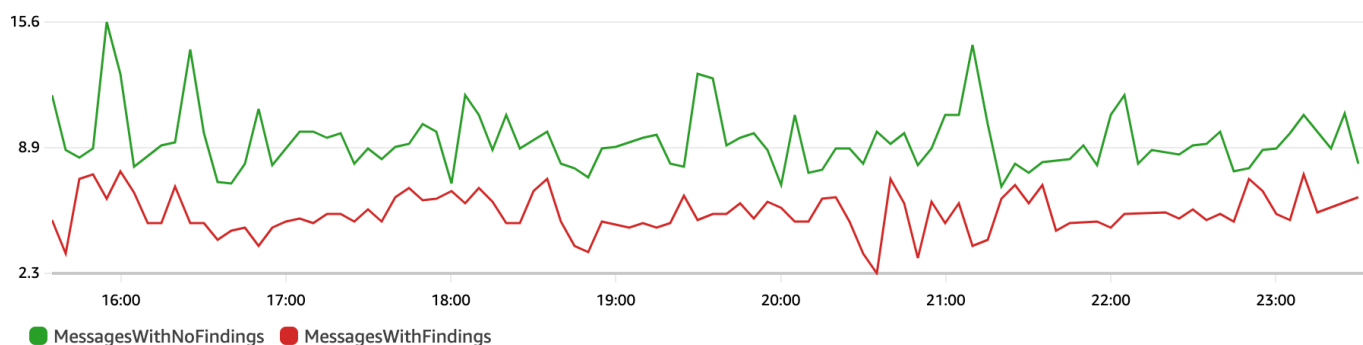
```

    "name": "Name",
    "count": 1,
    "detections": [
      {
        "start": 1,
        "end": 2
      }
    ],
  },
  {
    "name": "PhoneNumber",
    "count": 2,
    "detections": [
      {
        "start": 3,
        "end": 4
      },
      {
        "start": 5,
        "end": 6
      }
    ]
  }
]
}
}

```

監査オペレーションのメトリクス

監査オペレーションで FindingsDestination または NoFindingsDestination プロパティが指定された場合、トピックのオーナーは CloudWatch MessagesWithFindings および MessagesWithNoFindings メトリクスも受信します。



識別解除オペレーション

識別解除オペレーションは、発行または配信するメッセージの機密データをマスキングまたは削除します。このオペレーションは、インバウンドとアウトバウンドの両方のメッセージに使用可能であり、次のいずれかのタイプの設定を必要とします。

- **MaskConfig** — 次の表からサポートされている文字を使用してマスクします。例えば、ssn: 123-45-6789 は ssn: ##### になります。

```
{
  "Operation": {
    "Deidentify": {
      "MaskConfig": {
        "MaskWithCharacter": "#"
      }
    }
  }
}
```

サポートされているマスク文字	名前
*	アスタリスク
A~Z、a~z、0~9	英数字
	Space
!	感嘆符
\$	ドル記号
%	パーセント記号
&	アンパサンド
()	かっこ
+	プラス記号
,	カンマ
-	ハイフン

サポートされているマスク文字	名前
.	Period
\	スラッシュ、バックスラッシュ
#	ナンバー記号
:	コロロン
;	セミコロン
=, <>	等号。小なりまたは大なり
@	アットマーク
[]	角かっこ
^	キャレット記号
_	下線
,	バックティック
	縦棒
~	チルダ記号

- RedactConfig — データを完全に削除して編集します。例えば、ssn: 123-45-6789 は ssn: になります。

```
{
  "Operation": {
    "Deidentify": {
      "RedactConfig": {}
    }
  }
}
```

受信メッセージでは、監査オペレーション後に機密データが識別解除され、メッセージ全体が機密である場合、SNS:Publish API 発信者は次の無効なパラメータエラーを受け取ります。

Error code: AuthorizationError ...

拒否オペレーション

[Deny] (拒否) オペレーションは、メッセージに機密データが含まれている場合に Publish API リクエストまたはメッセージの配信のいずれかを中断します。拒否オペレーションオブジェクトは、追加の設定を必要としないため空です。

```
"Operation": {
  "Deny": {}
}
```

インバウンドメッセージでは、SNS:Publish API 発信者が認可エラーを受け取ります。

Error code: AuthorizationError ...

アウトバウンドメッセージでは、Amazon SNS トピックはメッセージをサブスクリプションに配信しません。不正配信を追跡するには、トピックの[\[delivery status logging\]](#) (配信ステータスのログ記録) を有効にします。以下は、配信ステータスログの例です。

```
{
  "notification": {
    "messageMD5Sum": "29638742ffb68b32cf56f42a79bcf16b",
    "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",
    "topicArn": "arn:aws:sns:us-east-1:123412341234:PII-data-topic",
    "timestamp": "2022-05-12T2:12:44Z"
  },
  "delivery": {
    "deliveryId": "98236591c-56aa-51ee-a5ed-0c7d43493170",
    "destination": "arn:aws:sqs:us-east-1:123456789012:NoNameAccess",
    "providerResponse": "The topic's data protection policy prohibits this message from being delivered to <subscription-arn>",
    "dwellTimeMs": 20,
    "attempts": 1,
    "statusCode": 403
  },
  "status": "FAILURE"
}
```

Amazon SNS データ保護ポリシーの例

以下は、機密データの監査および拒否に使用できるデータ保護ポリシーの例です。サンプルアプリケーションを含む完全なチュートリアルについては、「[Introducing message data protection for Amazon SNS](#)」(Amazon SNS のメッセージデータ保護の実装) のブログ記事を参照してください。

監査ポリシーの例

監査ポリシーを使うと、インバウンドメッセージの最大 99% を監査して、検出結果を [Amazon CloudWatch](#)、<https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html>、[Amazon S3](#) に送信できます。

例えば、監査ポリシーを作成して、システムのいずれかが誤って機密データを送受信していないかを評価できます。監査結果から、システムがクレジットカード情報を必要としないシステムにその情報を送信していることが明らかになった場合は、データ保護ポリシーを実装してデータの配信を防止できます。

次の例では、クレジットカード番号を検索し、その結果を CloudWatch Logs、Firehose、および Amazon S3 に送信することで、トピックを通過するメッセージの 99% を監査します。

データ保護ポリシー:

```
{
  "Name": "__example_data_protection_policy",
  "Description": "Example data protection policy",
  "Version": "2021-06-01",
  "Statement": [
    {
      "DataDirection": "Inbound",
      "Principal": ["*"],
      "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"
      ],
      "Operation": {
        "Audit": {
          "SampleRate": "99",
          "FindingsDestination": {
            "CloudWatchLogs": {
              "LogGroup": "<example log name>"
            },
            "Firehose": {
              "DeliveryStream": "<example stream name>"
            }
          }
        }
      }
    }
  ]
}
```

```
    },
    "S3": {
      "Bucket": "<example bucket name>"
    }
  }
}
]
```

監査結果の形式の例:

```
{
  "messageId": "...",
  "callerPrincipal": "arn:aws:sts::123456789012:assumed-role/ExampleRole",
  "resourceArn": "arn:aws:sns:us-east-1:123456789012:ExampleArn",
  "dataIdentifiers": [
    {
      "name": "CreditCardNumber",
      "count": 1,
      "detections": [
        { "start": 1, "end": 2 }
      ]
    }
  ],
  "timestamp": "2021-04-20T00:33:40.241Z"
}
```

インバウンド識別解除マスキングステートメントを含むポリシーの例

次の例では、メッセージの内容から機密データをマスキングして、ユーザーがメッセージ内の `CreditCardNumber` をトピックに発行できないようにします。

```
{
  "Name": "__example_data_protection_policy",
  "Description": "Example data protection policy",
  "Version": "2021-06-01",
  "Statement": [
    {
      "DataDirection": "Inbound",
```

```
{
  "Principal": [
    "arn:aws:iam::123456789012:user/ExampleUser"
  ],
  "DataIdentifier": [
    "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"
  ],
  "Operation": {
    "Deidentify": {
      "MaskConfig": {
        "MaskWithCharacter": "#"
      }
    }
  }
}
```

インバウンド識別解除マスキング結果の例:

```
// original message
My credit card number is 4539894458086459

// delivered message
My credit card number is #####
```

インバウンド識別解除の削除ステートメントを使用したポリシーの例

次の例では、メッセージ内の機密データを削除して、ユーザーがメッセージ内の CreditCardNumber をトピックに発行できないようにします。

```
{
  "Name": "__example_data_protection_policy",
  "Description": "Example data protection policy",
  "Version": "2021-06-01",
  "Statement": [
    {
      "DataDirection": "Inbound",
      "Principal": [
        "arn:aws:iam::123456789012:user/ExampleUser"
      ],
      "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"
      ]
    }
  ]
}
```

```
    ],  
    "Operation": {  
      "Deidentify": {  
        "RedactConfig": {}  
      }  
    }  
  }  
]  
}
```

インバウンド識別解除の削除結果の例:

```
// original message  
My credit card number is 4539894458086459  
  
// delivered message  
My credit card number is
```

アウトバウンド識別解除マスキングステートメントを使用したポリシーの例

次の例では、メッセージ内の機密データをマスキングして、ユーザーがメッセージ内の CreditCardNumber を受信できないようにします。

```
{  
  "Name": "__example_data_protection_policy",  
  "Description": "Example data protection policy",  
  "Version": "2021-06-01",  
  "Statement": [  
    {  
      "DataDirection": "Outbound",  
      "Principal": [  
        "arn:aws:iam::123456789012:user/ExampleUser"  
      ],  
      "DataIdentifier": [  
        "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"  
      ],  
      "Operation": {  
        "Deidentify": {  
          "MaskConfig": {  
            "MaskWithCharacter": "-"  
          }  
        }  
      }  
    }  
  ]  
}
```

```
}  
]  
}
```

アウトバウンド識別解除マスキング結果の例:

```
// original message  
My credit card number is 4539894458086459  
  
// delivered message  
My credit card number is -----
```

アウトバウンド識別解除の削除ステートメントを使用したポリシーの例

次の例では、メッセージ内の機密データを削除して、ユーザーがメッセージ内の `CreditCardNumber` を受信できないようにします。

```
{  
  "Name": "__example_data_protection_policy",  
  "Description": "Example data protection policy",  
  "Version": "2021-06-01",  
  "Statement": [  
    {  
      "DataDirection": "Outbound",  
      "Principal": [  
        "arn:aws:iam::123456789012:user/ExampleUser"  
      ],  
      "DataIdentifier": [  
        "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"  
      ],  
      "Operation": {  
        "Deidentify": {  
          "RedactConfig": {}  
        }  
      }  
    }  
  ]  
}
```

アウトバウンド識別解除の削除結果の例:

```
// original message
```

```
My credit card number is 4539894458086459
```

```
// delivered message
```

```
My credit card number is
```

インバウンド拒否ステートメントを使用するポリシーの例

次の例では、メッセージの内容に `CreditCardNumber` が含まれている場合、ユーザーがメッセージをトピックに発行するのをブロックします。API レスポンスで拒否されたペイロードのステータスコードは「403 `AuthorizationError`」です。

```
{
  "Name": "__example_data_protection_policy",
  "Description": "Example data protection policy",
  "Version": "2021-06-01",
  "Statement": [
    {
      "DataDirection": "Inbound",
      "Principal": [
        "arn:aws:iam::123456789012:user/ExampleUser"
      ],
      "DataIdentifier": [
        "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"
      ],
      "Operation": {
        "Deny": {}
      }
    }
  ]
}
```

アウトバウンド拒否ステートメントを使用するポリシーの例

次の例では、AWS アカウントが を含むメッセージの受信をブロックします `CreditCardNumber`。

```
{
  "Name": "__example_data_protection_policy",
  "Description": "Example data protection policy",
  "Version": "2021-06-01",
  "Statement": [
    {
      "DataDirection": "Outbound",
```

```

    "Principal": [
      "arn:aws:iam::123456789012:user/ExampleUser"
    ],
    "DataIdentifier": [
      "arn:aws:dataprotection::aws:data-identifier/CreditCardNumber"
    ],
    "Operation": {
      "Deny": {}
    }
  }
]
}

```

Amazon CloudWatch にロギングされたアウトバウンド拒否結果の例:

```

{
  "notification": {
    "messageMD5Sum": "2e8f58ff2eed723b56b15493fbfb5a5",
    "messageId": "8747a956-ebf1-59da-b291-f2c2e4b87c9c",
    "topicArn": "arn:aws:sns:us-east-2:664555388960:test1",
    "timestamp": "2022-09-08 15:40:57.144"
  },
  "delivery": {
    "deliveryId": "6a422437-78cc-5171-ad64-7fa3778507aa",
    "destination": "arn:aws:sqs:us-east-2:664555388960:test",
    "providerResponse": "The topic's data protection policy prohibits this message from being delivered to <subscription arn>",
    "dwellTimeMs": 22,
    "attempts": 1,
    "statusCode": 403
  },
  "status": "FAILURE"
}

```

Amazon SNS でのデータ保護ポリシーの作成

[データ保護ポリシー](#)は、アプリケーションまたは AWS のサービス間を移動する機密情報を監査、識別解除 (マスキングまたは編集)、および拒否 (ブロック) することで、Amazon SNS トピックに対して発行されたデータを保護します。AWS API、AWS CLI CloudFormation、または AWS マネジメントコンソールを使用して、Amazon SNS でデータ保護ポリシーを作成できます。Amazon SNS トピックごとに定義できるポリシーは 1 つだけです。各データ保護ポリシーには、1 つまたは複数の識別解除および拒否ステートメントと 1 つの監査ステートメントのみを含めることができます。

トピック

- [API を使用して Amazon SNS でデータ保護ポリシーを作成する](#)
- [CLI を使用して Amazon SNS でデータ保護ポリシーを作成する](#)
- [CloudFormation を使用して Amazon SNS でデータ保護ポリシーを作成する](#)
- [コンソールを使用して Amazon SNS でデータ保護ポリシーを作成する](#)
- [SDK を使用してメッセージデータを保護するための Amazon SNS データ保護ポリシーを作成する](#)

API を使用して Amazon SNS でデータ保護ポリシーを作成する

AWS アカウント内の Amazon SNS リソースの数とサイズは限られています。詳細については、「[Amazon Simple Notification Service のエンドポイントとクォータ](#)」を参照してください。

API を使用したデータ保護ポリシーの作成

AWS API を使用して Amazon SNS データ保護ポリシーを作成します。

Amazon SNS トピック (AWS API) とともにデータ保護ポリシーを作成するには

標準 Amazon SNS トピックの DataProtectionPolicy プロパティの使用：

- [CreateTopic](#)

既存の Amazon SNS トピックのデータ保護ポリシーを取得または作成するには (AWS API)

以下のいずれかのオペレーションを呼び出します。

- [GetDataProtectionPolicy](#)
- [PutDataProtectionPolicy](#)

CLI を使用して Amazon SNS でデータ保護ポリシーを作成する

AWS アカウント内の Amazon SNS リソースの数とサイズは限られています。詳細については、「[Amazon Simple Notification Service のエンドポイントとクォータ](#)」を参照してください。

を使用したデータ保護ポリシーの作成 AWS CLI

AWS Command Line Interfaceを使って Amazon SNS データ保護ポリシーを作成することができます。

Amazon SNS トピック (AWS CLI) と一緒にデータ保護ポリシーを作成するには

このオプションを使用して、標準の Amazon SNS トピックと一緒に新しいデータ保護ポリシーを作成します。

- [create-topic](#)

既存の Amazon SNS トピック (AWS CLI) のデータ保護ポリシーを作成または取得するには

以下のいずれかのオペレーションを呼び出します。

- [get-data-protection-policy](#)
- [put-data-protection-policy](#)

CloudFormation を使用して Amazon SNS でデータ保護ポリシーを作成する

AWS アカウント内の Amazon SNS リソースの数とサイズは限られています。詳細については、「[Amazon Simple Notification Service のエンドポイントとクォータ](#)」を参照してください。

データ保護ポリシーの作成 (CloudFormation)

を使用して Amazon SNS データ保護ポリシーを作成します CloudFormation。

Amazon SNS トピック (CloudFormation) と一緒にデータ保護ポリシーを作成するには

このオプションを使用して、標準の Amazon SNS トピックと一緒に新しいデータ保護ポリシーを作成します。

- [AWS::SNS::Topic](#)

コンソールを使用して Amazon SNS でデータ保護ポリシーを作成する

AWS アカウント内の Amazon SNS リソースの数とサイズは限られています。詳細については、「[Amazon Simple Notification Service のエンドポイントとクォータ](#)」を参照してください。

Amazon SNS トピックとともにデータ保護ポリシーを作成するには (コンソール)

このオプションを使用して、標準の Amazon SNS トピックと一緒に新しいデータ保護ポリシーを作成します。

1. [Amazon SNS コンソール](#)にサインインします。
2. トピックを選択するか、新しいトピックを作成できます。トピックの作成の詳細については、「[Amazon SNS のトピックの作成](#)」を参照してください。
3. [Create topic] (トピックの作成) ページの [Details] (詳細) セクションで [Standard] (標準) を選択します。
 - a. トピックの名前を入力します。
 - b. (オプション) トピックの表示名を入力します。
4. [Data protection policy] (データ保護ポリシー) を展開します。
5. [Configuration mode] (設定モード) を選択します。
 - [Basic] (ベーシック) — シンプルなメニューを使用してデータ保護ポリシーを定義します。
 - [Advanced] (アドバンスド) — JSON を使用してカスタムデータ保護ポリシーを定義します。
6. (オプション) 独自のカスタムデータ識別子を作成するには、[カスタムデータ識別子の設定セクション] を展開し、次の操作を行います。
 - a. カスタムデータ識別子の一意の名前を入力します。カスタムデータ識別子名には、英数字、アンダースコア (_)、ハイフン (-) を使用できます。カスタム識別子名の長さは最大 128 文字です。カスタムデータ識別子は、[マネージドデータ識別子](#)と同じ名前を共有することはできません。カスタムデータ識別子の制限の詳細な一覧については、「[カスタムデータ識別子の制約](#)」を参照してください。
 - b. カスタムデータ識別子の正規表現 (Regex) を入力します。Regex は、英数字、Regex 予約文字、および記号をサポートしています。Regex の最大長は 200 文字です。Regex が複雑すぎる場合、Amazon SNS は API コールに失敗します。Regex の制限の詳細な一覧については、[カスタムデータ識別子の制約](#) を参照してください。
 - c. (オプション) 必要に応じて [カスタムデータ識別子の追加] を選択し、データ識別子を追加します。各データ保護ポリシーに使用できるカスタムデータ識別子は最大 10 個です。
7. データ保護ポリシーに追加したいステートメントを選択します。監査、識別解除 (マスクまたは編集)、拒否 (ブロック) の各ステートメントタイプを同じデータ保護ポリシーに追加できます。
 - a. [Add audit statement] (監査ステートメントの追加) — 監査する機密データ、そのデータについて監査するメッセージの割合、監査ログの送信先を設定します。

Note

データ保護ポリシーまたはトピックごとに許可される監査ステートメントは 1 つだけです。

- i. データ識別子を指定して監査する機密データを定義します。
- ii. [Audit sample rate] (監査サンプルレート) に、機密情報を監査するメッセージの割合を最大 99% で入力します。
- iii. 監査送信先で、監査結果の送信 AWS のサービス 先を選択し、AWS のサービス 使用する各送信先名を入力します。次のAmazon Web Services から選択できます。
 - Amazon CloudWatch — CloudWatch Logs は AWS の標準ロギングソリューションです。CloudWatch Logs を使用すると、Logs Insights ([サンプルはこちら](#)) を使用してログ分析を実行し、メトリクスとアラームを作成できます。CloudWatch Logs は多くのサービスがログを公開する場所であるため、1 つのソリューションを使用してすべてのログを簡単に集約できます。Amazon CloudWatch の詳細については、「[Amazon CloudWatch ユーザーガイド](#)」を参照してください。
 - – Firehose は、さらにログ分析を行うために、Splunk、OpenSearch、Amazon Redshift へのリアルタイムストリーミングのニーズに対応します。詳細については、「[ユーザーガイド](#)」を参照してください。
 - Amazon Simple Storage Service — Amazon S3 は、アーカイブ用のログ記録先として経済的です。ログは、数年間にわたって保持が必要な場合があります。こうした場合、ログを Amazon S3 に保存することで、コストを節約できます。Amazon Simple Storage Service の詳細については、「[Amazon Simple Storage Service ユーザーガイド](#)」を参照してください。
- b. 識別解除ステートメントの追加 — メッセージで識別解除したい機密データを設定して、そのデータをマスクまたは編集するか、アカウントでそのデータの配信を停止します。
 - i. データ識別子の場合は、識別解除する機密データを選択します。
 - ii. この識別解除ステートメントを定義する で、この識別解除ステートメントが適用される AWS アカウントまたは IAM プリンシパルを選択します。このステートメントは、すべての AWS アカウントに適用できます。また、アカウント ID や IAM エンティティ ARN を使用する特定の AWS アカウントまたは IAM エンティティ (アカウント

ルート、ロール、またはユーザー) にも適用できます。複数の ID または ARN を設定するには、それぞれをカンマ (,) で区切ります。

サポートされている [IAM](#) プリンシパルは次のとおりです。

- IAM アカウントプリンシパル — 例: `arn:aws:iam::AWS-account-ID:root`。
- IAM ロールプリンシパル — 例: `arn:aws:iam::AWS-account-ID:role/role-name`。
- IAM ユーザープリンシパル — 例: `arn:aws:iam::AWS-account-ID:user/user-name`。

iii. 識別解除オプションの場合は、機密データを識別解除する方法を選択します。以下のオペレーションがサポートされています。

- 編集 — データを完全に削除します。例えば、`email: classified@amazon.com` は `email:` になります。
- マスク — データを単一の文字に置き換えます。例えば、`email: classified@amazon.com` は `email: *****` になります。

iv. (オプション) 必要に応じて、さらに識別解除ステートメントを追加します。

c. [Add deny statement] (拒否ステートメントの追加) — トピック内を移動しないようにする機密データと、そのデータの送信を防ぐプリンシパルを構成します。

i. データ方向については、拒否ステートメントのメッセージの方向を選択します。

- [Inbound messages] (インバウンドメッセージ) — この拒否ステートメントをトピックに送信されるメッセージに適用します。
- [Outbound messages] (アウトバウンドメッセージ) — この拒否ステートメントを、トピックがサブスクリプションエンドポイントに配信するメッセージに適用します。

ii. データ識別子を選択して拒否する機密データを定義します。

iii. この拒否ステートメントに適用する IAM プリンシパルを選択します。アカウント IDs または IAM エンティティ ARNs を使用するすべてのアカウント、AWS 特定の AWS アカウント または IAM エンティティ (アカウントルート、ロール、ユーザーなど) に適用できます。複数の ID または ARN を設定するには、それぞれをカンマ (,) で区切ります。サポートされている [IAM](#) プリンシパルは次のとおりです。

- IAM アカウントプリンシパル — 例: `arn:aws:iam::AWS-account-ID:root`。

- IAM ロールプリンシパル — 例: arn:aws:iam::AWS-account-ID:role/role-name。
 - IAM ユーザープリンシパル — 例: arn:aws:iam::AWS-account-ID:user/user-name。
- iv. (オプション) 必要に応じて、さらに拒否ステートメントを追加します。

SDK を使用してメッセージデータを保護するための Amazon SNS データ保護ポリシーを作成する

AWS アカウント内の Amazon SNS リソースの数とサイズは限られています。詳細については、「[Amazon Simple Notification Service のエンドポイントとクォータ](#)」を参照してください。

AWS SDK を使用したデータ保護ポリシーの作成

AWS SDK を使用して Amazon SNS データ保護ポリシーを作成します。

Amazon SNS トピック (AWS SDK) とともにデータ保護ポリシーを作成するには

以下のオプションを使用して、標準の Amazon SNS トピックと一緒に新しいデータ保護ポリシーを作成します。

Java

```
/**
 * For information regarding CreateTopic see this documentation topic:
 *
 * https://docs.aws.amazon.com/code-samples/latest/catalog/javav2-sns-src-main-java-com-example-sns-CreateTopic.java.html
 */

public static String createSNSTopicWithDataProtectionPolicy(SnsClient snsClient,
    String topicName, String dataProtectionPolicy) {

    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
            .name(topicName)
            .dataProtectionPolicy(dataProtectionPolicy)
            .build();

        CreateTopicResponse result = snsClient.createTopic(request);
        return result.topicArn();
    }
```

```
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

JavaScript

```
// Import required AWS SDK clients and commands for Node.js
import {CreateTopicCommand } from "@aws-sdk/client-sns";
import {snsClient } from "../libs/snsClient.js";

// Set the parameters
const params = { Name: "TOPIC_NAME", DataProtectionPolicy:
  "DATA_PROTECTION_POLICY" };

const run = async () => {
  try {
    const data = await snsClient.send(new CreateTopicCommand(params));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
run();
```

既存の Amazon SNS トピック (SDK) のデータ保護ポリシーを作成または取得するにはAWS

以下のオプションを使用して、標準の Amazon SNS トピックと一緒に新しいデータ保護ポリシーを作成または取得します。

Java

```
public static void putDataProtectionPolicy(SnsClient snsClient, String topicName,
String dataProtectionPolicy) {

    try {
        PutDataProtectionPolicyRequest request =
        PutDataProtectionPolicyRequest.builder()
```

```

        .resourceArn(topicName)
        .dataProtectionPolicy(dataProtectionPolicy)
        .build();

    PutDataProtectionPolicyResponse result =
snsClient.putDataProtectionPolicy(request);
    System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode()
        + "\n\nTopic " + request.resourceArn()
        + " DataProtectionPolicy " + request.dataProtectionPolicy());
} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
}

public static void getDataProtectionPolicy(SnsClient snsClient, String topicName) {

    try {
        GetDataProtectionPolicyRequest request =
GetDataProtectionPolicyRequest.builder()
            .resourceArn(topicName)
            .build();

        GetDataProtectionPolicyResponse result =
snsClient.getDataProtectionPolicy(request);

        System.out.println("\n\nStatus is " + result.sdkHttpResponse().statusCode()
            + "\n\nDataProtectionPolicy: \n\n" + result.dataProtectionPolicy());
    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

JavaScript

```

// Import required AWS SDK clients and commands for Node.js
import {PutDataProtectionPolicyCommand, GetDataProtectionPolicyCommand} from "@aws-
sdk/client-sns";
import {snsClient} from "../libs/snsClient.js";

```



```
// Set the parameters
const putParams = { ResourceArn: "TOPIC_ARN", DataProtectionPolicy:
  "DATA_PROTECTION_POLICY" };

const runPut = async () => {
  try {
    const data = await snsClient.send(new
    PutDataProtectionPolicyCommand(putParams));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
runPut();

// Set the parameters
const getParams = { ResourceArn: "TOPIC_ARN" };

const runGet = async () => {
  try {
    const data = await snsClient.send(new
    GetDataProtectionPolicyCommand(getParams));
    console.log("Success.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err.stack);
  }
};
runGet();
```

Amazon SNS でデータ保護ポリシーを削除する

AWS API、AWS CLI CloudFormation、または を使用して Amazon SNS データ保護ポリシーを削除できます AWS マネジメントコンソール。

Amazon SNS データ保護ポリシーの一般情報については、「[Amazon SNS データ保護ポリシーの理解](#)」を参照してください。

AWS アカウントの Amazon SNS データ保護ポリシーリソースの数とサイズには制限があります。詳細については、「AWS 全般のリファレンス」の「[Amazon SNS API スロットリング](#)」を参照してください。

コンソールを使用したデータ保護ポリシーの削除

コンソールを使用してマネージドデータ保護ポリシーを削除するには

1. [Amazon SNS コンソール](#)にサインインします。
2. 削除するデータ保護ポリシーを含むトピックを選択します。
3. [編集] を選択します。
4. [Data protection policy] (データ保護ポリシー) セクションを展開します。
5. 削除するデータ保護ポリシーステートメントの横にある [Remove] (削除) を選択します。
6. [Save changes] (変更の保存) をクリックします。

空の JSON 文字列を使用してデータ保護ポリシーを削除する

データ保護ポリシーは、空の JSON 文字列に更新することで削除できます。

を使用したデータ保護ポリシーの削除 AWS CLI

AWS CLIを使用してデータ保護ポリシーを削除できます。

```
//aws sns put-data-protection-policy --resource-arn topic-arn --data-protection-policy ""
```

Amazon SNS データ識別子

Amazon SNS は、機械学習やパターンマッチングなどの基準と手法の組み合わせを使用して、機密データを検出します。これらの基準と手法は、総称してデータ識別子と呼ばれ、多くの国や地域の機密データタイプの大規模かつ増加しているリストを検出できます。Amazon SNS マネージドデータ識別子には、財務データ、個人健康情報 (PHI)、個人を特定できる情報 (PII) を保護するために、事前設定されたデータタイプが用意されています。また、カスタムデータ識別子を使用して、特定のユースケースに合わせた独自のデータ識別子を作成することもできます。

Using managed data identifiers in Amazon SNS

マネージドデータ識別子とは

Amazon SNS マネージドデータ識別子は、特定の国またはリージョンのクレジットカード番号、AWS シークレットアクセスキー、パスポート番号など、特定のタイプの機密データを検出するように設計されています。データ保護ポリシーを作成すると、これらの識別子を使用してトピックを通過

するメッセージを分析し、検出された場合にアクションを実行するように Amazon SNS を設定できます。

Amazon SNS は、マネージドデータ識別子を使用して、次のカテゴリの機密データを検出できます。

- プライベートキーや AWS シークレットアクセスキーなどの認証情報
- IP アドレスや MAC アドレスなどのデバイス識別子。
- クレジットカード番号などの財務情報
- 健康保険または医療識別番号などの PHI に関する医療情報。
- 個人情報 (運転免許証や社会保障番号など)

各カテゴリ内で、Amazon SNS は複数のタイプの機密データを検出できます。このセクションのトピックでは、各タイプとその検出に関連する要件をリスト化して説明します。各タイプでは、データを検出するように設計されたマネージドデータ識別子の一意の識別子 (ID) も示します。データ保護ポリシーを作成するとき、この ID を使用して、検出するメッセージデータ保護のマネージドデータ識別子を含めることができます。

キーワード要件

特定のタイプの機密データを検出するため、Amazon SNS ではデータの近くにあるキーワードをスキャンします。特定のタイプのデータに当てはまる場合、このセクションのその後のトピックでは、そのデータの特定のキーワード要件を示します。

キーワードでは大文字と小文字が区別されません。さらに、キーワードにスペースが含まれている場合、Amazon SNS は、スペースを含まないキーワードのバリエーションや、スペースではなくアンダースコア (_) またはハイフン (-) を含むキーワードのバリエーションを自動的に照合します。場合によっては、Amazon SNS はキーワードの一般的なバリエーションに対処するためにキーワードを拡張または短縮します。

機密データタイプの Amazon SNS マネージドデータ識別子

次のテーブルでは、Amazon SNS がマネージドデータ識別子を使用して検出できる認証情報、デバイス、財務、医療および個人健康情報 (PHI) のタイプの一覧とその説明を示しています。これらは、個人を特定できる情報 (PII) としても認定される可能性のある特定のタイプのデータに加えたものです。

地域に依存するデータ識別子には、ダッシュ付きの識別子名と 2 文字のコード (ISO 3166-1 alpha-2) が必要です。例えば、DriversLicense-US です。

識別子	Category	国/言語
BankAccountNumber	金融	DE、ES、FR、GB、IT
cepCode	個人	BR
Cnpj	個人	BR
cpfCode	個人	BR
DriversLicense	個人	AT、AU、BE、 BG、CA、CY、 CZ、DE、DK、EE、ES、FI、 FR、GB、GR、 HR、HU、IE、IT、LT、LU、 LV、MT、NL、 PL、PT、RO、SE、SI、SK、 US
DrugEnforcementAgencyNumber	健康	US
ElectoralRollNumber	個人	GB
HealthInsuranceCardNumber	健康	EU
HealthInsuranceClaimNumber	健康	US
HealthInsuranceNumber	健康	FR
HealthcareProcedureCode	健康	US
IndividualTaxIdentificationNumber	個人	US
inSeeCode	個人	FR
MedicareBeneficiaryNumber	健康	US
NationalDrugCode	健康	US

識別子	Category	国/言語
NationalIdentificationNumber	個人	DE、ES、IT
NationalInsuranceNumber	個人	GB
NationalProviderId	健康	米国
NhsNumber	健康	GB
nieNumber	個人	ES
nifNumber	個人	ES
PassportNumber	個人	CA、DE、ES、 FR、GB、IT、US
PermanentResidenceNumber	個人	CA
PersonalHealthNumber	健康	CA
PhoneNumber	個人	BR、DE、ES、 FR、GB、IT、US
PostalCode	個人	CA
rgNumber	個人	BR
SocialInsuranceNumber	個人	CA
SSN	個人	es-US
TaxID	個人	DE、ES、FR、GB
ZipCode	個人	US

言語や地域に依存しないサポート対象の識別子

識別子	Category
Address	個人
AwsSecretKey	認証情報
CreditCardExpiration	金融
CreditCardNumber	金融
CreditCardSecurityCode	金融
EmailAddress	個人
IpAddress	個人
LatLong	個人
名前	個人
OpenSshPrivateKey	認証情報
PgpPrivateKey	認証情報
PkcsPrivateKey	認証情報
PuttyPrivateKey	認証情報
VehicleIdentificationNumber	個人

Amazon SNS 機密データタイプ: 認証情報

次のテーブルでは、Amazon SNS がマネージドデータ識別子を使用して検出できる認証情報のタイプをリスト化して説明します。

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	国とリージョン
AWS シークレットアクセスキー	AwsSecretKey	aws_secret_access_key, credentials,	いずれか

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	国とリージョン
		secret access key, secret key, set-awscred edential	
OpenSSH プライベートキー	OpenSshPrivateKey	いいえ	いずれか
PGP プライベートキー	PgpPrivateKey	いいえ	いずれか
公開鍵暗号標準 (PKCS) プライベートキー	PkcsPrivateKey	いいえ	いずれか
PuTTY プライベートキー	PuttyPrivateKey	いいえ	いずれか

認証情報データタイプのデータ識別子 ARN

以下は、データ保護ポリシーに追加できるデータ識別子の Amazon リソースネーム (ARN) のリストを示しています。

認証情報データ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/AwsSecretKey
```

```
arn:aws:dataprotection::aws:data-identifier/OpenSshPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PgpPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PkcsPrivateKey
```

```
arn:aws:dataprotection::aws:data-identifier/PuttyPrivateKey
```

Amazon SNS 機密データタイプ: デバイス

次のテーブルは、Amazon SNS がマネージドデータ識別子を使用して検出できるデバイス識別子のタイプをリスト化して説明します。

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	国とリージョン
IP アドレス	IpAddress	いいえ	いずれか

デバイスデータタイプのデータ識別子 ARN

以下は、データ保護ポリシーに追加できるデータ識別子の Amazon リソースネーム (ARN) のリストを示しています。

デバイスデータ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/IpAddress
```

Amazon SNS 機密データタイプ: 財務情報

次のテーブルでは、Amazon SNS がマネージドデータ識別子を使用して検出できる財務情報のタイプをリスト化して説明します。

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
銀行口座番号	BankAccountNumber BankAccountNumber-US	はい、 銀行口座番号のキーワード を参照してください。	これには、国コードなどの要素を含む、最大 34 文字の英数字で構成される国際銀行口座番号 (IBAN) が含まれます。	フランス、ドイツ、イタリア、スペイン、英国

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
クレジットカードの有効期限	CreditCardExpiration	exp d、exp m、exp y、expiration、expiry	–	いずれか
クレジットカードの磁気ストライプデータ	CreditCardMagneticStripe	以下が含まれます: card data、iso7813、mag、magstripe、stripe、swipe。	トラック 1 と 2 が含まれます。	いずれか

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
クレジットカード番号	CreditCardNumber	account number、american express、amex、bank card、card、card num、card number、cc #、ccn、check card、credit、credit card#、dankort、debit、debit card、diners club、discover、electron、electron verification code、japanese card bureau、jcb、mastercard、mc、pan、payment account number、payment card number、pcn、union pay、visa	検出では、データを Luhn チェック式に従った 13～19 桁のシーケンスである必要があり、American Express、Dankort、Diner's Club、Discover、Electron、Japanese Card Bureau (JCB)、Mastercard、UnionPay、Visa のいずれかのタイプのクレジットカードについて標準のカード番号プレフィックスを使用します (以下の 1 の上付きリンク)。	いずれか

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
クレジットカード認証コード	CreditCardSecurityCode	card id、card identification code、card identification number、card security code、card validation code、card validation number、card verification data、card verification value、cvc、cvc2、cvv、cvv2、elo verification code	–	いずれか

1. Amazon SNS は、クレジットカード発行会社がパブリックテストのために予約している、以下のシーケンスの出現をレポートしません。

1220000000000003、2222405343248877、2222990905257051、2223007648726984、22235771200176 および 76009244561。

銀行口座番号のキーワード

次のキーワードを使用して、国コードなどの要素を含む、最大 34 文字の英数字で構成される国際銀行口座番号 (IBAN) を検出します。

国またはリージョン	キーワード			
フランス	account code, account number, accountno #, accountnu mber#, bban, code bancaire, compte bancaire, customer account id, customer account number, customer bank account id, iban, numéro de compte			
ドイツ	account code, account number, accountno #, accountnu mber#, bankleitz ahl, bban, customer account id, customer account number, customer bank account id, geheimzahl, iban, kartenum mer, kontonumm er, kreditkar			

国またはリージョン	キーワード			
	tennummer, sepa			
イタリア	account code, account number, accountno #, accountnumber#, bban, codice bancario, conto bancario, customer account id, customer account number, customer bank account id, iban, numero di conto			

国またはリージョン	キーワード			
スペイン	account code, account number, accountno #, accountnu mber#, bban, código cuenta, código cuenta bancaria, cuenta cliente id, customer account ID, customer account number, customer bank account id, iban, número cuenta bancaria cliente, número cuenta cliente			
英国	account code, account number, accountno #, accountnu mber#, bban, customer account id, customer account number, customer bank account id, iban, sepa			

国またはリージョン	キーワード			
米国	bank account、bank acct、checking account、checking acct、deposit account、deposit acct、savings account、savings acct、checking account、checking acct			

財務データタイプのデータ識別子 ARN

以下は、データ保護ポリシーに追加できるデータ識別子の Amazon リソースネーム (ARN) のリストを示しています。

財務データ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-DE
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-ES
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-FR
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-GB
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-IT
```

```
arn:aws:dataprotection::aws:data-identifier/BankAccountNumber-US
```

```
arn:aws:dataprotection::aws:data-identifier/CreditCardExpiration
```

```
arn:aws:dataprotection::aws:data-identifier/CreditCardNumber
```

財務データ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/CreditCardSecurityCode
```

Amazon SNS 機密データタイプ: 保護対象健康情報 (PHI)

次のテーブルでは、Amazon SNS がマネージドデータ識別子を使用して検出できる保護対象の医療情報 (PHI) のタイプをリスト化して説明します。

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	国とリージョン
麻薬取締局 (DEA) 登録番号	DrugEnforcementAgencyNumber	dea number, dea registration	米国
健康保険証番号 (EHIC)	HealthInsuranceCardNumber	assicurazione sanitaria numero、carta assicurazione numero、carte d'assurance maladie、carte européenne d'assurance maladie、ceam、ehic、ehic#、finlandehicnumber#、gesundheitskarte、hälsokort、health card、health card number、health insurance card、health insurance number、insurance card number、krankenversicherungskarte、krankenversicherungsnummer、med	EU

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	国とリージョン
		ical account number、numero conto medico、nu méro d'assurance maladie、numéro de carte d'assuran ce、numéro de compte medical、n úmero de cuenta médica、número de seguro de salud、número de tarjeta de seguro、sa iraanhoitokortin、s airausvakuutuskort ti、sairausvakuutus numero、sjukförsäkr ing nummer、sj ukförsäkringskort、 suomi ehic-nume ro、tarjeta de salud、terveyskortt i、tessera sanitaria assicurazione numero、versicherun gsnummer	
健康保険請求番号 (HICN)	HealthInsuranceClaimNumber	health insurance claim number, hic no, hic no., hic number, hic#, hicn, hicn#., hicno#	US

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	国とリージョン
健康保険または医療識別番号	HealthInsuranceNumber	carte d'assuré social, carte vitale, insurance card	FR
ヘルスケア共通手順コーディングシステム (HCPCS) コード	HealthcareProcedureCode	current procedural terminology, hcpcs, healthcare common procedure coding system	US
メディケア受給者番号 (MBN)	MedicareBeneficiaryNumber	mbi, medicare beneficiary	US
全米医薬品コード (NDC)	NationalDrugCode	national drug code, ndc	US
国家プロバイダー識別子 (NPI)	NationalProviderId	hipaa, n.p.i, national provider, npi	US
National Health Service (NHS) 番号	NhsNumber	national health service, NHS	GB
個人健康管理番号 (PHN)	PersonalHealthNumber	canada healthcare number, msp number, personal healthcare number, phn, soins de santé	CA

健康保険と医療識別番号のキーワード

さまざまなタイプの健康保険と医療識別番号を検出するために、Amazon SNS では番号の近くにあるキーワードが必要です。これには、欧州健康保険カード番号 (EU、フィンランド)、健康保険番号 (フランス)、メディケア受益者識別子 (米国)、国民保険番号 (英国)、NHS 番号 (英国)、個人保険番号 (カナダ) が含まれます。

次のテーブルに、Amazon SNS が特定の国およびリージョンについて認識するキーワードのリストを示します。

国またはリージョン	キーワード
カナダ	Canada healthcare number, msp number, personal healthcare number, phn, soins de santé
EU	assicurazione sanitaria numero, carta assicurazione numero, carte d'assurance maladie, carte européenne d'assurance maladie, ceam, ehic, ehic#, finlandehicnumber#, gesundheitskarte, hälsokort, health card, health card number, health insurance card, health insurance number, insurance card number, krankenversicherungskarte, krankenversicherungsnummer, medical account number, numero conto medico, numéro d'assurance maladie, numéro de carte d'assurance, numéro de compte medical, número de cuenta médica, número de seguro de salud, número de tarjeta de seguro, sairaanhoitokortin, sairausvaikutuskortti, sairausvakuutusnumero, sjukförsäkring nummer, sjukförsäkringskort, suomi ehic-numero, tarjeta de salud, terveyskortti, tessera sanitaria assicurazione numero, versicherungsnummer
フィンランド	ehic, ehic#, finland health insurance card, finlandehicnumber#, finska sjukförsäkringskort, hälsokort, health card, health card number, health insurance card, health insurance number, sairaanhoitokortin, sairaanhoitokortin, sairausvaikutuskortti, sairausvakuutusnumero, sjukförsäkring nummer, sjukförsäkringskort

国またはリージョン	キーワード
	t, suomen sairausvakuutuskortti, suomi ehic-numero, terveyskortti
フランス	carte d'assuré social, carte vitale, insurance card
英国	国民保健サービス、NHS
米国	mbi, medicare beneficiary

保護対象の医療情報 (PHI) データタイプのデータ識別子 ARN

PHI データ保護ポリシーで利用できるデータ識別子 Amazon リソースネーム (ARN) を以下に示します。

PHI データ識別子 ARN

arn:aws:dataprotection::aws:data-identifier/DrugEnforcementAgencyNumber-US

arn:aws:dataprotection::aws:data-identifier/HealthcareProcedureCode-US

arn:aws:dataprotection::aws:data-identifier/HealthInsuranceCardNumber-EU

arn:aws:dataprotection::aws:data-identifier/HealthInsuranceClaimNumber-US

arn:aws:dataprotection::aws:data-identifier/HealthInsuranceNumber-FR

arn:aws:dataprotection::aws:data-identifier/MedicareBeneficiaryNumber-US

arn:aws:dataprotection::aws:data-identifier/NationalDrugCode-US

arn:aws:dataprotection::aws:data-identifier/NationalInsuranceNumber-GB

arn:aws:dataprotection::aws:data-identifier/NationalProviderId-US

arn:aws:dataprotection::aws:data-identifier/NhsNumber-GB

arn:aws:dataprotection::aws:data-identifier/PersonalHealthNumber-CA

Amazon SNS 機密データタイプ: 個人を特定できる情報 (PII)

次の表では、Amazon SNS がマネージドデータ識別子を使用して検出できる個人を特定できる情報 (PII) のタイプをリスト化して説明します。

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
生年月日	DateOfBirth	dob, date of birth, birthdate, birth date, birthday, b-day, bday	Support には、すべての数字や数字と月の名前の組み合わせなど、ほとんどの日付形式が含まれます。日付コンポーネントは、スペース、スラッシュ (/)、またはハイフン (-) で区切ることができます。	いずれか
Código de Endereçamento Postal (CEP)	CepCode	cep, código de endereçamento postal, codigo de endereçamento postal	–	ブラジル
Cadastro Nacional da Pessoa Jurídica (CNPJ)	Cnpj	cadastro nacional da pessoa jurídica, cadastro nacional da pessoa juridica, cnpj	–	ブラジル

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
Cadastro de Pessoas Físicas (CPF)	CpfCode	Cadastro de pessoas físicas, cadastro de pessoas físicas, cadastro de pessoa física, cadastro de pessoa fisica, cpf	–	ブラジル

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
運転免許証識別番号	DriversLicense	はい、 運転免許証識別番号のキーワード を参照してください。	–	オーストラリア、オーストリア、ベルギー、ブルガリア、カナダ、クロアチア、キプロス、チェコ共和国、デンマーク、エストニア、フィンランド、フランス、ドイツ、ギリシャ、ハンガリー、アイルランド、イタリア、ラトビア、リトアニア、ルクセンブルク、マルタ、オランダ、ポーランド、ポルトガル、ルーマニア、スロバキア、スロベニア、スペイン、スウェーデン、英国、米国

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
選挙人名簿番号	Electoral RollNumber	electoral#, electoral #, electoralnumber, electoral number, electoralroll#, electoral roll#, electoral roll #, electoral roll no., electoral roll number, electoralrollno	–	UK
個人納税者識別	IndividualTaxIdentification Number	はい、 納税者識別と参照番号のキーワード を参照してください。	–	米国
国立統計経済研究所 (INSEE)	InseeCode	はい、 国民識別番号のキーワード を参照してください。	–	フランス
国民識別番号	NationalIdentificationNumber	はい、 国民識別番号のキーワード を参照してください。	これには、Documento Nacional de Identidad (DNI) 識別子 (スペイン)、Codice fiscale codes (イタリア)、国民 ID カード番号 (ドイツ語) が含まれます。	ドイツ、イタリア、スペイン

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
国民保険番号 (NINO)	NationalInsuranceNumber	insurance no., insurance number, insurance #, national insurance number, nationalinsurance#, nationalinsurancenum, nin, nino	–	UK
Número de identidad de extranjero (NIE)	NieNumber	はい、 納税者識別と参照番号のキーワード を参照してください。	–	スペイン
Número de Identificación Fiscal (NIF)	NifNumber	はい、 納税者識別と参照番号のキーワード を参照してください。	–	スペイン
パスポート番号	PassportNumber	はい、 パスポート番号のキーワード を参照してください。	–	カナダ、フランス、ドイツ、イタリア、スペイン、英国、米国

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
本籍地	Permanent Residence Number	carte résident permanent , numéro carte résident permanent, numéro résident permanent , permanent resident card, permanent resident card number, permanent resident no, permanent resident no., permanent resident number, pr no, pr no., pr non, pr number, résident permanent no., résident permanent non	–	カナダ

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
Phone number (電話番号)	PhoneNumber	ブラジル: キーワードには cel、cellular、fone、móvel、número residencial、numero residencial、telefone も含まれます その他: cell、contact、fax、fax number、mobile、phone、phone number、tel、telephone、telephone number	これには、米国の通話料無料の番号とファックス番号が含まれます。キーワードがデータの近くにある場合、番号に国コードを含める必要はありません。 キーワードがデータの近くでない場合は、番号に国コードを含める必要があります。	ブラジル、カナダ、フランス、ドイツ、イタリア、スペイン、英国、米国
郵便番号	PostalCode	No	–	カナダ
Registro Geral (RG)	RgNumber	はい、 国民識別番号のキーワード を参照してください。	–	ブラジル

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
社会保険番号 (SIN)	SocialInsuranceNumber	canadian id, numéro d'assurance sociale, social insurance number, sin	–	カナダ
社会保障番号 (SSN)	Ssn	スペイン – número de la seguridad social、social security no.、social security no. número de la seguridad social、social security number、social securityno#、ssn、ssn# US — social social、ss#、ssn	–	スペイン、米国
納税者識別番号または参照番号	TaxId	はい、 納税者識別と参照番号のキーワード を参照してください。	これには、TIN (フランス)、Steueridentifikationsnummer (ドイツ)、CIF (スペイン)、TRN、UTR (英国) が含まれます。	フランス、ドイツ、スペイン、英国

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
米国郵便番号	ZipCode	zip code, zip+4	–	US
郵送先住所	Address	No	キーワードは必要ありませんが、検出では、住所には都市または場所の名前および郵便番号を含める必要があります。	オーストラリア、カナダ、フランス、ドイツ、イタリア、スペイン、英国、米国
電子メールアドレス	EmailAddress	email、email address、email、email address	–	いずれか

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
全地球測位システム (GPS) 座標	LatLong	coordinate, coordinates, lat long, latitude longitude, location, position	Amazon SNS は、緯度と経度の座標がペアとして保存され、それらが10 進度 (DD) 形式の場合、GPS 座標を検出できます (例: 41.948614 ,-87.655311)。サポートには、度 10 進分 (DDM) 形式 (例: 41°56.916 8'N 87°39.318 7'W)、または、度、分、秒 (DMS) 形式 (例: 41°56'55.0104"N 87°39'19. 1196"W) の座標は含まれません。	いずれか
フルネーム	Name	No	Amazon SNS はフルネームのみを検出できます。Support はラテン文字セットに限定されます。	いずれか

検出タイプ	マネージドデータ識別子 ID	キーワードが必須	追加情報	国とリージョン
車両識別番号 (VIN)	VehicleIdentificationNumber	Fahrgeste llnummer, niv, numarul de identificare, numarul seriei de sasiu, serie sasiu, numer VIN, Número de Identificação do Veículo, Número de Identificación de Automóvil es, numéro d'identification du véhicule, vehicle identification number, vin, VIN numerus	Amazon SNS は、17 文字の シーケンスで 構成され、ISO 3779 および 3780 規格に従っ た VIN を検出で きます。これら の規格は、世界 中で使用するた めに設計されて います。	いずれか

運転免許証識別番号のキーワード

さまざまなタイプの運転免許証識別番号を検出するために、Amazon SNS では番号の近くにあるキーワードが必要です。次のテーブルに、Amazon SNS が特定の国およびリージョンについて認識するキーワードのリストを示します。

国またはリージョン	キーワード
オーストラリア	dl# dl:, dl :, dlno# driver licence, driver license, driver permit, drivers lic., drivers licence, driver's licence, drivers license, driver's license, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit

国またはリージョン	キーワード
オーストリア	führerschein, fuhrerschein, führerschein republik österreich, fuhrerschein republik österreich
ベルギー	fuehrerschein, fuehrerschein- nr, fuehrersc heinnummer, fuhrerschein, führerschein, fuhrerschein- nr, führerschein- nr, fuhrersch einnummer, führerscheinnummer, numéro permis conduire, permis de conduire, rijbewijs, rijbewijsnummer
ブルガリア	превозно средство, свидетелство за управление на моторно, свидетелство за управление на мпс, сумпс, шофьорска книжка
カナダ	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit, permis de conduire
クロアチア	vozačka dozvola
キプロス	άρθεια οδήγησης
チェコ共和国	číslo licence, číslo licence řidiče, číslo řidičskéh o průkazu, ovladače lic., povolení k jízdě, povolení řidiče, řidiči povolení, řidičský průkaz, řidičský průkaz
デンマーク	kørekort, kørekortnummer

国またはリージョン	キーワード
エストニア	juhi litsentsi number, juhiloa number, juhiluba, juhiluba number
フィンランド	ajokortin numero, ajokortti, förare lic., körkort, körkort nummer, kuljettaja lic., permis de conduire
フランス	permis de conduire
ドイツ	fuehrerschein, fuehrerschein- nr, fuehrersc heinnummer, fuhrerschein, führerschein, fuhrerschein- nr, führerschein- nr, fuhrersch einnummer, führerscheinnnummer
ギリシャ	δεία οδήγησης, adeia odigisis
ハンガリー	illesztőprogramok lic, jogosítvány, jogsi, licencszám, vezető engedély, vezetői engedély
アイルランド	ceadúnas tiomána
イタリア	patente di guida, patente di guida numero, patente guida, patente guida numero
ラトビア	autovadītāja apliecība, licences numurs, vadītāja apliecība, vadītāja apliecības numurs, vadītāja atļauja, vadītāja licences numurs, vadītāji lic.
リトアニア	vairuotojo pažymėjimas
ルクセンブルグ	fahrerlaubnis, führerschäin
マルタ	liċenzja tas-sewqan
オランダ	permis de conduire, rijbewijs, rijbewijsnummer

国またはリージョン	キーワード
ポーランド	numer licencyjny, prawo jazdy, zezwolenie na prowadzenie
ポルトガル	carta de condução, carteira de habilitação, carteira de motorist, carteira habilitação, carteira motorist, licença condução, licença de condução, número de licença, número licença, permissão condução, permissão de condução
ルーマニア	numărul permisului de conducere, permis de conducere
スロバキア	číslo licencie, číslo vodičského preukazu, ovládače lic., povolenia vodičov, povolenie jazdu, povolenie na jazdu, povolenie vodiča, vodičský preukaz
スロベニア	vozniško dovoljenje
スペイン	carnet conducir, el carnet de conducir, licencia conducir, licencia de manejo, número carnet conducir, número de carnet de conducir, número de permiso conducir, número de permiso de conducir, número licencia conducir, número permiso conducir, permiso conducción, permiso conducir, permiso de conducción
スウェーデン	ajokortin numero, dlno# ajokortti, drivere lic., förare lic., körkort, körkort nummer, körkortsn ummer, kuljettajat lic.

国またはリージョン	キーワード
英国	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit
米国	dl#, dl:, dlno#, driver licence, driver licences, driver license, driver licenses, driver permit, drivers lic., drivers licence, driver's licence, drivers licences, driver's licences, drivers license, driver's license, drivers licenses, driver's licenses, drivers permit, driver's permit, drivers permit number, driving licence, driving license, driving permit

国民識別番号のキーワード

さまざまなタイプの国民識別番号を検出するために、Amazon SNS では番号の近くにあるキーワードが必要です。これには、Documento Nacional de Identidad (DNI) 識別子 (スペイン)、フランス国立統計経済研究所 (INSEE) コード、ドイツの国民 ID カード番号、Registro Geral (RG) 番号 (ブラジル) が含まれます。

次のテーブルに、Amazon SNS が特定の国およびリージョンについて認識するキーワードのリストを示します。

国またはリージョン	キーワード
ブラジル	registro geral, rg
フランス	assurance sociale, carte nationale d'identité, cni, code sécurité sociale, French social security number, fssn#, insee, insurance

国またはリージョン	キーワード
	number, national id number, nationalid#, numéro d'assurance, sécurité sociale, sécurité sociale non., sécurité sociale numéro, social, social security, social security number, socialsecuritynumber, ss#, ssn, ssn#
ドイツ	ausweisnummer, id number, identification number, identity number, insurance number, personal id, personalausweis
イタリア	codice fiscal, dati anagrafici, ehic, health card, health insurance card, p. iva, partita i.v.a., personal data, tax code, tessera sanitaria
スペイン	dni, dni#, dninúmero#, documento nacional de identidad, identidad único, identidadúnico#, insurance number, national identification number, national identity, nationalid#, nationali dno#, número nacional identidad, personal identification number, personal identity no, unique identity number, uniqueid#

パスポート番号のキーワード

さまざまなタイプのパスポート番号を検出するには、Amazon SNS では番号の近くにあるキーワードが必要です。次のテーブルに、Amazon SNS が特定の国およびリージョンについて認識するキーワードのリストを示します。

国またはリージョン	キーワード
カナダ	passepport, passeport#, passport, passport#, passportno, passportno#
フランス	numéro de passeport, passeport, passeport #, passeport #, passeportn °, passeport n °, passeportNon, passeport non

国またはリージョン	キーワード
ドイツ	ausstellungsdatum, ausstellungsort, geburtsdatum, passport, passports, reiseepass, reiseepassnr, reiseepassnummer
イタリア	italian passport number, numéro passeport, numéro passeport italien, passaporto, passaporto italiana, passaporto numero, passport number, repubblica italiana passaporto
スペイン	españa pasaporte, libreta pasaporte, número pasaporte, pasaporte, passport, passport book, passport no, passport number, spain passport
英国	passeport #, passeport n °, passeportNon, passeport non, passeportn °, passport #, passport no, passport number, passport#, passportid
米国	passport, travel document

納税者識別と参照番号のキーワード

さまざまなタイプの納税者識別番号と参照番号を検出するために、Amazon SNS では番号の近くにあるキーワードが必要です。次のテーブルに、Amazon SNS が特定の国およびリージョンについて認識するキーワードのリストを示します。

国またはリージョン	キーワード
ブラジル	cadastro de pessoa física, cadastro de pessoa física, cadastro de pessoas físicas, cadastro de pessoas físicas, cadastro nacional da pessoa jurídica, cadastro nacional da pessoa juridica, cnpj, cpf

国またはリージョン	キーワード
フランス	numéro d'identification fiscale, tax id, tax identification number, tax number, tin, tin#
ドイツ	identifikationsnummer, steuer id, steueridentifikationsnummer, steuernummer, tax id, tax identification number, tax number
スペイン	cif, cif número, cifnúmero#, nie, nif, número de contribuyente, número de identidad de extranjero, número de identificación fiscal, número de impuesto corporativo, personal tax number, tax id, tax identification number, tax number, tin, tin#
英国	paye, tax id, tax id no., tax id number, tax identification, tax identification#, tax no., tax number, tax reference, tax#, taxid#, temporary reference number, tin, trn, unique tax reference, unique taxpayer reference, utr
米国	個別の納税者識別番号、itin、i.t.i.n。

個人を特定できる情報 (PII) のデータ識別子 ARN

以下の表は、データ保護ポリシーに追加できるデータ識別子の Amazon リソースネーム (ARN) のリストを示しています。

PII データ識別子 ARN
arn:aws:dataprotection::aws:data-identifier/Address
arn:aws:dataprotection::aws:data-identifier/CepCode-BR
arn:aws:dataprotection::aws:data-identifier/Cnpj-BR
arn:aws:dataprotection::aws:data-identifier/CpfCode-BR

PII データ識別子 ARN

arn:aws:dataprotection::aws:data-identifier/DateOfBirth

arn:aws:dataprotection::aws:data-identifier/DriversLicense-AT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-AU

arn:aws:dataprotection::aws:data-identifier/DriversLicense-BE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-BG

arn:aws:dataprotection::aws:data-identifier/DriversLicense-CA

arn:aws:dataprotection::aws:data-identifier/DriversLicense-CY

arn:aws:dataprotection::aws:data-identifier/DriversLicense-CZ

arn:aws:dataprotection::aws:data-identifier/DriversLicense-DE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-DK

arn:aws:dataprotection::aws:data-identifier/DriversLicense-EE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-ES

arn:aws:dataprotection::aws:data-identifier/DriversLicense-FI

arn:aws:dataprotection::aws:data-identifier/DriversLicense-FR

arn:aws:dataprotection::aws:data-identifier/DriversLicense-GB

arn:aws:dataprotection::aws:data-identifier/DriversLicense-GR

arn:aws:dataprotection::aws:data-identifier/DriversLicense-HR

arn:aws:dataprotection::aws:data-identifier/DriversLicense-HU

arn:aws:dataprotection::aws:data-identifier/DriversLicense-IE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-IT

PII データ識別子 ARN

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LU

arn:aws:dataprotection::aws:data-identifier/DriversLicense-LV

arn:aws:dataprotection::aws:data-identifier/DriversLicense-MT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-NL

arn:aws:dataprotection::aws:data-identifier/DriversLicense-PL

arn:aws:dataprotection::aws:data-identifier/DriversLicense-PT

arn:aws:dataprotection::aws:data-identifier/DriversLicense-RO

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SE

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SI

arn:aws:dataprotection::aws:data-identifier/DriversLicense-SK

arn:aws:dataprotection::aws:data-identifier/DriversLicense-US

arn:aws:dataprotection::aws:data-identifier/ElectoralRollNumber-GB

arn:aws:dataprotection::aws:data-identifier/EmailAddress

arn:aws:dataprotection::aws:data-identifier/IndividualTaxIdentificationNumber-US

arn:aws:dataprotection::aws:data-identifier/InseeCode-FR

arn:aws:dataprotection::aws:data-identifier/LatLong

arn:aws:dataprotection::aws:data-identifier/Name

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-DE

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-ES

PII データ識別子 ARN

arn:aws:dataprotection::aws:data-identifier/NationalIdentificationNumber-IT

arn:aws:dataprotection::aws:data-identifier/NieNumber-ES

arn:aws:dataprotection::aws:data-identifier/NifNumber-ES

arn:aws:dataprotection::aws:data-identifier/PassportNumber-CA

arn:aws:dataprotection::aws:data-identifier/PassportNumber-DE

arn:aws:dataprotection::aws:data-identifier/PassportNumber-ES

arn:aws:dataprotection::aws:data-identifier/PassportNumber-FR

arn:aws:dataprotection::aws:data-identifier/PassportNumber-GB

arn:aws:dataprotection::aws:data-identifier/PassportNumber-IT

arn:aws:dataprotection::aws:data-identifier/PassportNumber-US

arn:aws:dataprotection::aws:data-identifier/PermanentResidenceNumber-CA

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-BR

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-DE

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-ES

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-FR

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-GB

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-IT

arn:aws:dataprotection::aws:data-identifier/PhoneNumber-US

arn:aws:dataprotection::aws:data-identifier/PostalCode-CA

arn:aws:dataprotection::aws:data-identifier/RgNumber-BR

PII データ識別子 ARN

```
arn:aws:dataprotection::aws:data-identifier/SocialInsuranceNumber-CA
```

```
arn:aws:dataprotection::aws:data-identifier/Ssn-ES
```

```
arn:aws:dataprotection::aws:data-identifier/Ssn-US
```

```
arn:aws:dataprotection::aws:data-identifier/TaxId-DE
```

```
arn:aws:dataprotection::aws:data-identifier/TaxId-ES
```

```
arn:aws:dataprotection::aws:data-identifier/TaxId-FR
```

```
arn:aws:dataprotection::aws:data-identifier/TaxId-GB
```

```
arn:aws:dataprotection::aws:data-identifier/VehicleIdentificationNumber
```

```
arn:aws:dataprotection::aws:data-identifier/ZipCode-US
```

Amazon SNS でのカスタムデータ識別子の使用

カスタムデータ識別子 (CDI) を使用すると、データ保護ポリシーで使用する独自のカスタム正規表現を定義できます。カスタムデータ識別子を使用すると、[マネージドデータ識別子](#)では提供できないビジネス固有の個人を特定できる情報 (PII) のユースケースをターゲットにすることができます。たとえば、カスタムデータ識別子を使用すると、会社固有の従業員 ID を検索できます。カスタムデータ識別子は、マネージドデータ ID と組み合わせて使用できます。

SNS カスタムデータ識別子とは

カスタムデータ識別子 (CDI) を使用すると、データ保護ポリシーで使用する独自のカスタム正規表現を定義できます。カスタムデータ識別子を使用すると、[マネージドデータ識別子](#)では提供できないビジネス固有の個人を特定できる情報 (PII) のユースケースをターゲットにすることができます。たとえば、カスタムデータ識別子を使用すると、会社固有の従業員 ID を検索できます。カスタムデータ識別子は、マネージドデータ ID と組み合わせて使用できます。

データ保護ポリシーでカスタムデータ識別子を使用する

以下のデータ保護ポリシーでは、会社固有の従業員 ID を含むペイロードを検出し、これらの ID をハッシュ記号 (#) でマスクするよう Amazon SNS トピックに指示します。

1. データ保護ポリシー内で Configuration ブロックを作成します。
2. カスタムデータ識別子の Name を入力します。例えば、**EmployeeId**。
3. カスタムデータ識別子の Regex を入力します。例えば、**EID-\d{9}-US**。
4. ポリシーステートメントで、以下のカスタムデータ識別子を参照します。

```
{
  "Name": "__example_data_protection_policy",
  "Description": "Example data protection policy",
  "Version": "2021-06-01",
  "Configuration": {
    "CustomDataIdentifier": [
      {
        "Name": "EmployeeId",
        "Regex": "EID-\d{9}-US"
      }
    ],
    "Statement": [
      {
        "DataDirection": "Inbound",
        "Principal": ["*"],
        "DataIdentifier": [
          "EmployeeId"
        ],
        "Operation": {
          "Deidentify": {
            "MaskConfig": {
              "MaskWithCharacter": "#"
            }
          }
        }
      }
    ]
  }
}
```

5. (オプション) 必要に応じて、Configuration ブロックにさらに カスタムデータ識別子を追加します。現在、データ保護ポリシーでは最大 10 個のカスタムデータ識別子を使用できます。

カスタムデータ識別子の制約

Amazon SNS カスタムデータ識別子には、以下の制限があります。

- 各データ保護ポリシーに使用できるカスタムデータ識別子は最大 10 個です。

- カスタムデータ識別子名に使用できるのは 128 文字までです。以下の文字がサポートされています。
 - 英数字: (a-zA-Z0-9)
 - 記号: ('_' | '-')
- RegEx の最大長は 200 文字です。以下の文字がサポートされています。
 - 英数字: (a-zA-Z0-9)
 - 記号: ('_' | '#' | '=' | '@' | '/' | ';' | ':' | '-' | ''')
 - RegEx 予約文字: ('^' | '\$' | '?' | '[' | ']' | '{' | '}' | '|' | '\\' | '*' | '+' | '!')
- カスタムデータ識別子は、マネージドデータ識別子と同じ名前を共有することはできません。
- カスタムデータ識別子は、各 Amazon SNS トピックのすべてのデータ保護ポリシーで指定する必要があります。

Amazon SNS メッセージ配信

このトピックでは、Amazon SNS がさまざまなシナリオでメッセージ配信を処理する方法について説明します。Amazon SNS がメッセージを元の変更されていない形式でエンドポイントに配信する、raw メッセージの配信について説明します。また、Amazon SNS トピックから別の Amazon SQS キューにメッセージを送信し AWS アカウント、クロスアカウントメッセージングに関するインサイトを提供する方法についても説明します。

このトピックでは、さまざまな Amazon SQS キューまたは Lambda 関数への Amazon SNS Amazon SQS メッセージの配信 AWS リージョン、クロスリージョン配信の仕組み、および関連する考慮事項について説明します。

さらに、メッセージが正常に配信されたか、問題が発生したかに関する重要な情報を取得するため、メッセージ配信ステータスを監視して解釈する方法についても説明します。メッセージ配信が失敗した場合に、Amazon SNS がメッセージを自動的に再配信して目的の宛先に確実に届くようにする方法など、メッセージ配信の再試行プロセスを理解できます。このトピックでは、デッドレターキューを使用して、複数回の試行後に配信できなかったメッセージをキャプチャする方法について説明します。これらの障害を効果的に分析してトラブルシューティングできるようになります。

Amazon SNS raw メッセージの配信

???, [Amazon SQS](#)、および [HTTP/S](#) エンドポイントによるメッセージの JSON フォーマット処理を回避するために、Amazon SNS は raw メッセージの配信を許可します。

- Amazon SQS エンドポイントに対して raw メッセージの配信を有効にすると、発行されたメッセージからすべての Amazon SNS メタデータが削除され、メッセージはそのまま送信されます。
- HTTP/S エンドポイントに対して raw メッセージの配信を有効にすると、値が true に設定された HTTP ヘッダー `x-amz-sns-rawdelivery` がメッセージに追加されます。これは、メッセージが JSON フォーマットなしで発行されたことを示します。
- HTTP/S エンドポイントに対して raw メッセージの配信を有効にすると、メッセージ本文、クライアント IP、および必要なヘッダーが配信されます。メッセージの属性を指定すると、そのメッセージは送信されません。
- Firehose エンドポイントの raw メッセージ配信を有効にすると、メッセージ本文が配信されます。メッセージの属性を指定すると、そのメッセージは送信されません。

AWS SDK を使用して raw メッセージ配信を有効にするには、SetSubscriptionAttribute API アクションを使用し、RawMessageDelivery 属性の値を に設定する必要がありますtrue。

を使用した raw メッセージ配信の有効化 AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[トピック] を選択します。
3. [トピック] ページで、Firehose、Amazon SQS、または HTTP/S エンドポイントにサブスクライブするトピックを選択します。
4. **[MyTopic]** ページの [サブスクリプション] セクションでサブスクリプションを選択したら、[編集] を選択します。
5. [Edit **EXAMPLE1-23bc-4567-d890-ef12g3hij456**] ページの [詳細] セクションで、[raw メッセージ配信の有効化] を選択します。
6. [変更を保存] をクリックします。

メッセージ形式の例

以下は、同じメッセージが同じ Amazon SQS キューに 2 回送信されている例です。唯一の違いは、最初のメッセージでは raw メッセージ配信が無効になり、2 番目のメッセージでは有効になっていることです。

- raw メッセージの配信を無効にします。

```
{
  "Type": "Notification",
  "MessageId": "dc1e94d9-56c5-5e96-808d-cc7f68faa162",
  "TopicArn": "arn:aws:sns:us-east-2:111122223333:ExampleTopic1",
  "Subject": "TestSubject",
  "Message": "This is a test message.",
  "Timestamp": "2021-02-16T21:41:19.978Z",
  "SignatureVersion": "1",
  "Signature":
    "FMG5t1ZhJNHLHUXvZgtZz1k24FzVa7oX0T4P03neeXw8ZEXZx6z35j2F0TuNYShn2h0bKNC/
    zLTnMyIxEzmi2X1sh0BWsJHkrW2xkR58ABZF+4uWHEE73yDVR4SyYAikP9jstZzDRm
    +bcVs8+T0yaLiEGLrIIIL4esi11lhIkGErCuy5btPcWXBdio2fpCRD5x9oR6gmE/
    rd5071X1c1uvnv4r1Lkk4pqP2/iUfxFZva1xLSRvgyfm6D9hNk1VyPfy
    +7Ta1MD01zmJu0rExtnSIbZew3foxgx8GT+1bZkLd0ZdtdRJ1IyPRP44eyq78sU0Eo/
    LsDr0Iak4ZDpg8dXg==",
```

```
"SigningCertURL": "https://sns.us-east-2.amazonaws.com/  
SimpleNotificationService-010a507c1833636cd94bdb98bd93083a.pem",  
"UnsubscribeURL": "https://sns.us-east-2.amazonaws.com/?  
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-  
east-2:111122223333:ExampleTopic1:e1039402-24e7-40a3-a0d4-797da162b297"  
}
```

- raw メッセージの配信を有効にします。

```
This is a test message.
```

Amazon SQS サブスクリプションのメッセージ属性と raw メッセージ配信

Amazon SNS では、メッセージに関する構造化メタデータ項目 (タイムスタンプ、地理空間データ、署名、識別子など) を指定できるメッセージ属性の配信をサポートしています。SQS サブスクリプションの場合、raw メッセージの配信を有効にすると、最大 10 個のメッセージ属性を送信できます。10 個を超えるメッセージ属性を送信するには、raw メッセージの配信を無効にする必要があります。raw メッセージ配信が有効になっている場合、10 個を超えるメッセージ属性を持つ Amazon SQS サブスクリプション宛てのメッセージは、クライアント側のエラーとして破棄されます。

別のアカウントの Amazon SQS キューへ Amazon SNS メッセージを送信する

このドキュメントでは、別のアカウントの Amazon SQS キューに 1 つ以上のサブスクリプションがある Amazon SNS トピックに通知を発行する方法について説明します。同じアカウント内にある場合と同じ方法でトピックとキューをセットアップします (「[Amazon SNS 通知の Amazon SQS キューへのファンアウトと非同期処理](#)」を参照)。主な相違点はサブスクリプションの確認の処理方法で、トピックへのキューのサブスクライブ方法によって異なります。

キューの所有者がサブスクリプションを作成すると確認が自動的に実行されるため、可能な場合は「[キューの所有者がサブスクリプションを作成する](#)」セクションで参照されている手順に従うことがベストプラクティスです。

Note

Amazon SQS キューに大量のメッセージがある場合は、キュー所有者がサブスクリプションを作成することをお勧めします。

キューの所有者がサブスクリプションを作成する

Amazon SQS キューを作成したアカウントがキューの所有者です。キューの所有者がサブスクリプションを作成するとき、サブスクリプションを確認する必要はありません。Subscribe アクションが完了するとすぐに、キューはトピックからの通知の受信を開始します。キューの所有者がトピック所有者のトピックにサブスクライブできるようにするには、トピック所有者が、トピックに対して Subscribe アクションを呼び出す許可をキューの所有者のアカウントに付与する必要があります。

ステップ 1: を使用してトピックポリシーを設定するには AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[トピック] を選択します。
3. トピックを選択し、[編集] を選択します。
4. [Edit **MyTopic**] ページで [アクセスポリシー] セクションを展開します。
5. 以下のポリシーを入力します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "111122223333"
      },
      "Action": "sns:Subscribe",
      "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
    }
  ]
}
```

このポリシーは、123456789012 アカウント内で MyTopic に sns:Subscribe を呼び出す許可を 111122223333 アカウントに付与します。

111122223333 のアカウントに対する認証情報が持っているユーザーは、MyTopic に登録できます。この許可によって、アカウント ID は、アクセス許可を IAM ユーザー/ロールに委任できます。呼び出しは、ルートアカウントまたは管理者ユーザーのみに許可されます。また、IAM ユーザー/ロールは、キューにサブスクリプションを許可するために sns:subscribe を持つ必要があります。

6. [Save changes] (変更の保存) をクリックします。

111122223333 アカウントに対する認証情報があるユーザーは、MyTopic に登録できます。

ステップ 2: AWS アカウント を使用して別の のトピックに Amazon SQS キューサブスクリプションを追加するには AWS マネジメントコンソール

開始する前に、トピックおよびキューの ARN を保持していて、[キューにメッセージを送信する許可がトピックに付与されている](#)ことを確認してください。

1. [Amazon SQSコンソール](#)にサインインします。
2. ナビゲーションパネルで、[Queues](キュー) を選択します。
3. キューのリストからキューを選択して、Amazon SNS トピックを購読します。
4. [Subscribe to Amazon SNS topic] (Amazon SNS トピックを購読する) を選択します。
5. [Specify an Amazon SNS topic available for this queue menu] (このキューメニューで利用できる Amazon SNS トピックを指定する) で、キューの Amazon SNS topic (Amazon SNS トピック) を選択します。
6. [Amazon SNS トピックの ARN を入力する]を選択してから、トピックの Amazon リソースネーム (ARN) を入力します。
7. [保存] を選択します。

Note

- サービスと通信できるようにするには、キューに Amazon SNS へのアクセス権限が必要です。
- キューの所有者であるため、サブスクリプションを確認する必要はありません。

キュー作成サブスクリプションを所有していないユーザー

サブスクリプションを作成するユーザーがキューの所有者ではない場合、そのサブスクリプションを確認する必要があります。

Subscribe アクションを使用すると、Amazon SNS はサブスクリプションの確認をキューに送信します。サブスクリプションが Amazon SNS コンソールに表示され、サブスクリプション ID が [保留中の確認] に設定されます。

サブスクリプションを確認するには、キューからメッセージを読み取る権限を持つユーザーがサブスクリプションの確認 URL を取得し、サブスクリプションの所有者がサブスクリプションの確認 URL を使用してサブスクリプションを確認する必要があります。サブスクリプションが確認されるまで、トピックに対して発行された通知はキューに送信されません。サブスクリプションを確認するには、Amazon SQS コンソールまたは [ReceiveMessage](#) アクションを使用できます。

Note

トピックにエンドポイントをサブスクライブする前に、キューに `sqs:SendMessage` アクセス権を設定してこのキューがトピックからメッセージを受信できるようにする必要があります。詳細については、「[ステップ 2: Amazon SQS キューにメッセージを送信する許可を Amazon SNS トピックに付与する](#)」を参照してください。

ステップ 1: AWS アカウント を使用して別の のトピックに Amazon SQS キューサブスクリプションを追加するには AWS マネジメントコンソール

開始する前に、トピックおよびキューの ARN を保持していて、[キューにメッセージを送信する許可がトピックに付与されている](#)ことを確認してください。

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[サブスクリプション] を選択します。
3. [サブスクリプション] ページで [サブスクリプションの作成] を選択します。
4. [サブスクリプションの作成] ページの [詳細] セクションで、以下を実行します。
 - a. [トピック ARN] にトピックの ARN を入力します。
 - b. [プロトコル] で、[Amazon SQS] を選択します。
 - c. [エンドポイント] にキューの ARN を入力します。
 - d. [サブスクリプションの作成] を選択します。

Note

- サービスと通信できるようにするには、キューに Amazon SNS へのアクセス権限が必要です。

Amazon SNS トピックが Amazon SQS キューへメッセージを送信できるようにするポリシーステートメントの例を以下に示します。

```
{
  "Sid": "Stmt1234",
  "Effect": "Allow",
  "Principal": "*",
  "Action": "sqs:SendMessage",
  "Resource": "arn:aws:sqs:us-west-2:111111111111:QueueName",
  "Condition": {
    "ArnEquals": {
      "aws:SourceArn": "arn:aws:sns:us-west-2:555555555555:TopicName"
    }
  }
}
```

ステップ 2: を使用してサブスクリプションを確認するには AWS マネジメントコンソール

1. [Amazon SQS コンソール](#)にサインインします。
2. トピックへのサブスクリプションが保留になっているキューを選択します。
3. [Send and receive messages] (メッセージの送受信) を選択し、[Poll for messages] (メッセージのポーリング) を選択します。

サブスクリプションの確認を含むメッセージがキューで受信されます。

4. [本文] 列で、次の操作を行います。
 - a. [詳細] を選択します。
 - b. [Message Details] (メッセージの詳細) ダイアログボックスで、[SubscribeURL] の値を見つけてメモします。これはサブスクリプションリンクです (以下に例を示します)。API トークンの検証の詳細については、Amazon SNS API リファレンスの「[ConfirmSubscription](#)」を参照してください。

```
https://sns.us-west-2.amazonaws.com/?
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-
east-2:123456789012:MyTopic&Token=2336412f37fb...
```

- c. サブスクリプション確認リンクを書き留めます。URL は、キュー所有者からサブスクリプション所有者に渡す必要があります。サブスクリプションの所有者は、URL を [Amazon SNS コンソール](#) に入力する必要があります。
5. サブスクリプション所有者として [Amazon SNS コンソール](#) にログインします。サブスクリプション所有者が確認を実行します。
6. 関連する [topic] (トピック) を選択します。
7. トピックのサブスクリプション一覧表で、関連するサブスクリプションを選択します。「Pending confirmation (保留中の確認)」とラベル付けされています。
8. [confirm subscription] (サブスクリプションの確認) を選択します。
9. サブスクリプション確認リンクを求めるモーダルが表示されます。サブスクリプション確認リンクを貼り付けます。
10. モーダルで [Confirm subscription] (サブスクリプションの確認) を選択します。

XML 応答が表示されます。例えば、

```
<ConfirmSubscriptionResponse>
  <ConfirmSubscriptionResult>
    <SubscriptionArn>arn:aws:sns:us-east-2:123456789012:MyTopic:1234a567-
bc89-012d-3e45-6fg7h890123i</SubscriptionArn>
  </ConfirmSubscriptionResult>
  <ResponseMetadata>
    <RequestId>abcd1efg-23hi-jkl4-m5no-p67q8rstuvw9</RequestId>
  </ResponseMetadata>
</ConfirmSubscriptionResponse>
```

サブスクライブしたキューは、トピックからメッセージを受信する準備ができています。

11. (オプション) Amazon SNS コンソールでトピックのサブスクリプションを表示すると、[保留中の確認] メッセージが [サブスクリプション ID] 列でサブスクリプション ARN に置き換えられています。

サブスクリプション解除リクエストでサブスクリプションに認証を要求させるにはどうすればよいですか。

サブスクリプションの所有者は、サブスクリプション確認時に `AuthenticateOnUnsubscribe` フラグを `true` に設定する必要があります。

- キューの所有者がサブスクリプションを作成すると、AuthenticateOnUnsubscribe は自動的に true に設定されます。
- 認証なしでサブスクリプション確認リンクに移動した場合、AuthenticateOnUnsubscribe を true に設定することはできません。

別のリージョンの Amazon SQS キューまたは関数に Amazon SNS Amazon SQS メッセージを送信する AWS Lambda

Amazon SNS はクロスリージョン配信をサポートしています。これは、デフォルトで有効になっているリージョンと [オプトインリージョン](#) です。Amazon SNS がサポートする AWS リージョン (オプトインリージョンを含む) の最新のリストについては、「Amazon Web Services 全般のリファレンス」の「[Amazon Simple Notification Service エンドポイントとクォータ](#)」を参照してください。

Amazon SNS は、Amazon SQS キューへの通知のクロスリージョン配信と AWS Lambda 関数をサポートしています。リージョンの 1 つがオプトインリージョンである場合は、サブスクライブされたリソースのポリシーで別の Amazon SNS サービスプリンシパルを指定する必要があります。

Amazon SNS サブスクリプションコマンドは、Amazon SNS がホストされている対応リージョンで実行する必要があります。例えば、Amazon SNS が us-east-1 リージョンのアカウント「A」にあり、Lambda 関数が us-east-2 リージョンのアカウント「B」にある場合、サブスクリプション CLI コマンドは us-east-1 リージョンのアカウント「A」で実行する必要があります。

オプトインリージョン

Amazon SNS では、次のオプトインリージョンがサポートされています。

リージョン名	リージョン
アフリカ (ケープタウン) リージョン	af-south-1
アジアパシフィック (香港) リージョン	ap-east-1
アジアパシフィック (ハイデラバード) リージョン	ap-south-2
アジアパシフィック (ジャカルタ) リージョン	ap-southeast-3
アジアパシフィック (メルボルン) リージョン	ap-southeast-4

リージョン名	リージョン
欧州 (ミラノ) リージョン	eu-south-1
欧州 (スペイン) リージョン	eu-south-2
欧州 (チューリッヒ) リージョン	eu-central-2
イスラエル (テルアビブ) リージョン	il-central-1
中東 (バーレーン) リージョン	me-south-1
中東 (アラブ首長国連邦) リージョン	me-central-1

オプトインリージョンを有効にする方法については、『』の[AWS「リージョンの管理」](#)を参照してくださいAmazon Web Services 全般のリファレンス。

Amazon SNS を使用してオプトインリージョンからデフォルトで有効になっているリージョンにメッセージを配信する場合、キュー用に作成されたリソースポリシーを変更する必要があります。プリンシパル `sns.amazonaws.com` を `sns.<opt-in-region>.amazonaws.com` に置き換えます。例:

- 例えば、米国東部 (バージニア北部) の Amazon SQS キューをアジアパシフィック (香港) の Amazon SNS トピックにサブスクライブする場合は、キューポリシーのプリンシパルを `sns.ap-east-1.amazonaws.com` に変更します。オプトインリージョンには、2019 年 3 月 20 日以降に開始されたリージョンが含まれます。これには、アジアパシフィック (香港)、アジアパシフィック (ジャカルタ)、中東 (バーレーン)、欧州 (ミラノ)、アフリカ (ケープタウン) が含まれます。2019 年 3 月 20 日以前に開始されたリージョンは、デフォルトで有効になっています。

Amazon SQS へのリージョン間での配信サポート

リージョン間での配信	サポート対象/サポート対象外	
デフォルトで有効なリージョンからオプトインリージョン	キューのサービスプリンシパルで <code>sns.<i>queue-region</i>.amazonaws.com</code> を使用してサポート	

リージョン間での配信	サポート対象/サポート対象外	
オプトインリージョンからデフォルトで有効なリージョン	キューのサービスプリンシパルで <code>sns.<i>topic-region</i>.amazonaws.com</code> を 使用してサポート	
オプトインリージョンからオプトインリージョン	キューのサービスプリンシパルで <code>sns.<i>queue-region</i>.amazonaws.com</code> を 使用してサポート	

オプトインリージョン (af-south-1) の Amazon SNS トピックが、デフォルトで有効なリージョン (us-east-1) の Amazon SQS キューに配信できるようにするアクセスポリシーステートメントの例を次に示します。これには、パス (Statement/Principal/Service) の下に、必要なリージョン化されたサービスプリンシパル設定が含まれています。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "allow_sns_arn:aws:sns:af-south-1:111111111111:source_topic_name",
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.af-south-1.amazonaws.com"
      },
      "Action": "SQS:SendMessage",
      "Resource": "arn:aws:sqs:us-west-1:111111111111:destination_queue_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:sns:af-south-1:111111111111:source_topic_name"
        }
      }
    }
  ]
}
```

```
    ]
}
```

- 米国東部 (バージニア北部) の AWS Lambda 関数をアジアパシフィック (香港) の Amazon SNS トピックにサブスクライブするには、AWS Lambda 関数ポリシーのプリンシパルを `sns.ap-east-1.amazonaws.com` に変更します。オプトインリージョンには、2019 年 3 月 20 日以降に開始されたリージョンが含まれます。これには、アジアパシフィック (香港)、アジアパシフィック (ジャカルタ)、中東 (バーレーン)、欧州 (ミラノ)、アフリカ (ケープタウン) が含まれます。2019 年 3 月 20 日以前に開始されたリージョンは、デフォルトで有効になっています。

へのクロスリージョン配信のサポート AWS Lambda

リージョン間での配信	サポート対象/サポート対象外	
デフォルトで有効なリージョンからオプトインリージョン	Lambda 関数のサービスプリンシパルで <code>sns.<i>lambda-function-region</i>.amazonaws.com</code> を使用してサポート	
オプトインリージョンからデフォルトで有効なリージョン	Lambda 関数のサービスプリンシパルで <code>sns.<i>topic-region</i>.amazonaws.com</code> を使用してサポート	
オプトインリージョンからオプトインリージョン	Lambda 関数のサービスプリンシパルで <code>sns.<i>lambda-function-region</i>.amazonaws.com</code> を使用してサポート	

Amazon SNS メッセージ配信ステータス

Amazon SNS では、以下の Amazon SNS エンドポイントでトピックに送信された通知メッセージの配信ステータスをログに記録できます。

- Amazon Data Firehose
- Amazon Simple Queue Service
- AWS Lambda

- HTTPS
- プラットフォームアプリケーションエンドポイント

配信ステータスログは Amazon CloudWatch Logs に送信され、メッセージ配信オペレーションに関するインサイトを提供します。これらのログは、次の目的に役立ちます。

- メッセージがエンドポイントに正常に配信されたかどうかを確認する。
- Amazon SNS へのエンドポイントから Amazon SNS に送信された応答を特定する。
- メッセージのドウェル時間 (発行のタイムスタンプからエンドポイントへの配信までの時間) を測定する。

配信ステータスのログ記録は AWS マネジメントコンソール、AWS SDKs、クエリ API、またはを使用して設定できます AWS CloudFormation。

配信ステータスのログ記録に関する前提条件

このトピックでは、Amazon SNS による CloudWatch への配信ログの書き込みを有効にするために必要な IAM アクセス許可の概要と、デフォルトのロググループの命名規則について説明します。これにより、CloudWatch Logs でメッセージ配信ログをモニタリングおよび分析するための正しい設定とアクセスが可能になります。

必要な IAM アクセス許可

配信ステータスのログ記録用にアタッチされた IAM ロールには、Amazon SNS が CloudWatch Logs に書き込めるように、次のアクセス許可が含まれている必要があります。設定時に、これらのアクセス許可を持つ既存のロールを使用するか、新しいロールを作成できます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
```

```
        "logs:PutLogEvents"  
    ],  
    "Resource": "arn:aws:logs:*:*:*"  
}  
]  
}
```

ロググループの命名規則

デフォルトでは、Amazon SNS は、次の命名規則を使用して、配信ステータスログ用の CloudWatch ロググループを作成します。このグループ内のログストリームは、エンドポイントプロトコル (Lambda、Amazon SQS など) に対応しています。CloudWatch Logs コンソールでこれらのログを表示するアクセス許可があることを確認してください。

```
sns/<region>/<account-id>/<topic-name>
```

を使用した配信ステータスのログ記録の設定 AWS マネジメントコンソール

このトピックでは、Amazon SNS トピックに関するメッセージ配信ステータスのログ記録を有効にする方法について説明します。ログ記録の設定、IAM ロールの割り当て、モニタリングとトラブルシューティングを目的とした CloudWatch Logs による配信ログのキャプチャの検証などを取り上げます。

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[トピック] を選択します。
3. 目的のトピックを選択し、[編集] を選択します。
4. [配信ステータスのログ記録] セクションを展開します。
5. ログ記録を有効にするプロトコル (HTTP、Lambda、Amazon SQS など) を選択します。
6. [成功サンプルレート] (CloudWatch Logs を受信するための成功メッセージの割合) を入力します。
7. [IAM ロール] セクションでは、成功のログ記録と失敗のログ記録の両方のロールを設定する必要があります。
 - 既存のサービスロールを使用 – Amazon SNS が CloudWatch にログを書き込む際に必要となるアクセス許可を持つ既存の IAM ロールを選択します。

- 新しいサービスロールを作成 – [新しいロールの作成] を選択して、IAM コンソールで成功および失敗した配信の IAM ロールを定義します。アクセス許可の詳細については、「[配信ステータスのログ記録に関する前提条件](#)」を参照してください。

8. [Save changes] (変更の保存) をクリックします。

ログ記録を有効にすると、メッセージ配信ステータスを含む CloudWatch Logs を表示して解析することができます。CloudWatch の使用方法の詳細については、「[CloudWatch のドキュメント](#)」を参照してください。

ログの設定の確認

1. CloudWatch Logs コンソールにサインインします。
2. `sns/<region>/<account-id>/<topic-name>` という名前のロググループを見つけます。
3. 設定したエンドポイントプロトコルのログストリームが存在することを確認します。
4. トピックにテストメッセージを送信して、配信の成功または失敗を示すログエントリが表示されることを確認します。

AWS SDKs を使用した配信ステータスのログ記録の設定

AWS SDKs は、メッセージ配信ステータスのログ記録のトピック属性を設定するための APIs を複数の言語で提供します。例えば、[SetTopicAttributes](#) API を使用して以下を設定します。

- `LambdaSuccessFeedbackRoleArn` – Lambda エンドポイントへのメッセージ配信が成功した場合の IAM ロール。
- `LambdaSuccessFeedbackSampleRate` – Lambda エンドポイントへの配信が成功したメッセージのサンプリングレート。
- `LambdaFailureFeedbackRoleArn` – Lambda エンドポイントへのメッセージ配信が失敗した場合の IAM ロール。

AWS CLI コマンドの例

```
aws sns set-topic-attributes \  
  --topic-arn arn:aws:sns:us-west-2:123456789012:MyTopic \  
  --attribute-name LambdaSuccessFeedbackRoleArn \  
  --value arn:aws:iam::123456789012:role/LambdaSuccessFeedbackRole
```

```
--attribute-value arn:aws:iam::123456789012:role/MyFeedbackRole
```

トピック属性

メッセージ配信ステータスには、次のトピック属性名の値を使用してください。

HTTP

- `HTTPSuccessFeedbackRoleArn` – HTTP エンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信が成功したことを示します。
- `HTTPSuccessFeedbackSampleRate` – HTTP エンドポイントにサブスクライブされている Amazon SNS トピックについてサンプリングする、成功したメッセージの割合を示します。
- `HTTPFailureFeedbackRoleArn` – HTTP エンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信が失敗したことを示します。

Amazon Data Firehose

- `FirehoseSuccessFeedbackRoleArn` – Amazon Data Firehose エンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信が成功したことを示します。
- `FirehoseSuccessFeedbackSampleRate` – Amazon Data Firehose エンドポイントにサブスクライブされている Amazon SNS トピックについてサンプリングする、成功したメッセージの割合を示します。
- `FirehoseFailureFeedbackRoleArn` – Amazon Data Firehose エンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信が失敗したことを示します。

AWS Lambda

- `LambdaSuccessFeedbackRoleArn` – Lambda エンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信が成功したことを示します。
- `LambdaSuccessFeedbackSampleRate` – Lambda エンドポイントにサブスクライブされている Amazon SNS トピックについてサンプリングする、成功したメッセージの割合を示します。
- `LambdaFailureFeedbackRoleArn` – Lambda エンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信が失敗したことを示します。

プラットフォームアプリケーションエンドポイント

- `ApplicationSuccessFeedbackRoleArn` – AWS アプリケーションエンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信ステータスの成功。
- `ApplicationSuccessFeedbackSampleRate` – AWS アプリケーションエンドポイントにサブスクライブされている Amazon SNS トピックについてサンプリングする、成功したメッセージの割合を示します。
- `ApplicationFailureFeedbackRoleArn` – AWS アプリケーションエンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信ステータスが失敗しました。

 Note

配信ステータスをプッシュ通知サービスに直接ログ記録するように、アプリケーションの属性を設定することもできます。詳細については、「[メッセージの配信ステータスの Amazon SNS アプリケーション属性を使用する](#)」を参照してください。

Amazon SQS

- `SQSSuccessFeedbackRoleArn` – Amazon SQS エンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信が成功したことを示します。
- `SQSSuccessFeedbackSampleRate` – Amazon SQS エンドポイントにサブスクライブされている Amazon SNS トピックについてサンプリングする、成功したメッセージの割合を示します。
- `SQSFailureFeedbackRoleArn` – Amazon SQS エンドポイントにサブスクライブされている Amazon SNS トピックのメッセージ配信が失敗したことを示します。

プラットフォームアプリケーションエンドポイントのログは、他のエンドポイントと同じ CloudWatch Logs グループに書き込まれます。

 Note

<ENDPOINT>SuccessFeedbackRoleArn および
<ENDPOINT>FailureFeedbackRoleArn 属性は、お客様に代わって CloudWatch Logs を使用する書き込みアクセス許可を Amazon SNS に付与するために使
用します。<ENDPOINT>SuccessFeedbackSampleRate 属性は、正常な
配信メッセージのサンプルレートの割合 (0~100) を指定するためのもので
す。<ENDPOINT>FailureFeedbackRoleArn 属性を設定した後は、メッセージの配信に失
敗すると、必ず CloudWatch Logs が生成されます。

AWS トピック属性を設定するための SDK の例

次のサンプルコードは、SetTopicAttributes を使用する方法を説明しています。

CLI

AWS CLI

トピックの属性を設定するには

次の set-topic-attributes の例では、指定したトピックの DisplayName 属性を設定します。

```
aws sns set-topic-attributes \  
  --topic-arn arn:aws:sns:us-west-2:123456789012:MyTopic \  
  --attribute-name DisplayName \  
  --attribute-value MyTopicDisplayName
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[SetTopicAttributes](#)」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。

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sns.SnsClient;  
import software.amazon.awssdk.services.sns.model.SetTopicAttributesRequest;  
import software.amazon.awssdk.services.sns.model.SetTopicAttributesResponse;  
import software.amazon.awssdk.services.sns.model.SnsException;  
  
/**  
 * 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
*/
public class SetTopicAttributes {

    public static void main(String[] args) {
        final String usage = ""

            Usage:    <attribute> <topicArn> <value>

            Where:
                attribute - The attribute action to use. Valid parameters are:
Policy | DisplayName | DeliveryPolicy .
                topicArn - The ARN of the topic.\s
                value - The value for the attribute.
            "";

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String attribute = args[0];
        String topicArn = args[1];
        String value = args[2];

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        setTopAttr(snsClient, attribute, topicArn, value);
        snsClient.close();
    }

    public static void setTopAttr(SnsClient snsClient, String attribute, String
topicArn, String value) {
        try {
            SetTopicAttributesRequest request =
SetTopicAttributesRequest.builder()
                .attributeName(attribute)
```

```

        .attributeValue(value)
        .topicArn(topicArn)
        .build();

        SetTopicAttributesResponse result =
snsClient.setTopicAttributes(request);
        System.out.println(
            "\n\nStatus was " + result.sdkHttpResponse().statusCode() +
"\n\nTopic " + request.topicArn()
            + " updated " + request.attributeName() + " to " +
request.attributeValue());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[SetTopicAttributes](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```

import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});

```


SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[SetTopicAttributes](#)」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 setTopAttr(
    attribute: String?,
    topicArnVal: String?,
    value: String?,
) {
    val request =
        SetTopicAttributesRequest {
            attributeName = attribute
            attributeValue = value
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.setTopicAttributes(request)
        println("Topic ${request.topicArn} was updated.")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[SetTopicAttributes](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Configure the message delivery status attributes for an Amazon SNS Topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);
$attribute = 'Policy | DisplayName | DeliveryPolicy';
$value = 'First Topic';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->setTopicAttributes([
        'AttributeName' => $attribute,
        'AttributeValue' => $value,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- APIの詳細については、AWS SDK for PHP API リファレンスの「[SetTopicAttributes](#)」を参照してください。

Ruby

SDK for Ruby

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。

```
# Service class to enable an SNS resource with a specified policy
class SnsResourceEnabler
  # Initializes the SnsResourceEnabler with an SNS resource client
  #
  # @param sns_resource [Aws::SNS::Resource] The SNS resource client
  def initialize(sns_resource)
    @sns_resource = sns_resource
    @logger = Logger.new($stdout)
  end

  # Sets a policy on a specified SNS topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param resource_arn [String] The ARN of the resource to include in the policy
  # @param policy_name [String] The name of the policy attribute to set
  def enable_resource(topic_arn, resource_arn, policy_name)
    policy = generate_policy(topic_arn, resource_arn)
    topic = @sns_resource.topic(topic_arn)

    topic.set_attributes({
      attribute_name: policy_name,
      attribute_value: policy
    })

    @logger.info("Policy #{policy_name} set successfully for topic
    #{topic_arn}.")
    rescue Aws::SNS::Errors::ServiceError => e
      @logger.error("Failed to set policy: #{e.message}")
    end

    private

    # Generates a policy string with dynamic resource ARNs
```

```
#
# @param topic_arn [String] The ARN of the SNS topic
# @param resource_arn [String] The ARN of the resource
# @return [String] The policy as a JSON string
def generate_policy(topic_arn, resource_arn)
  {
    Version: '2008-10-17',
    Id: '__default_policy_ID',
    Statement: [{
      Sid: '__default_statement_ID',
      Effect: 'Allow',
      Principal: { "AWS": '*' },
      Action: ['SNS:Publish'],
      Resource: topic_arn,
      Condition: {
        ArnEquals: {
          "AWS:SourceArn": resource_arn
        }
      }
    }]
  }.to_json
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_arn = 'MY_TOPIC_ARN' # Should be replaced with a real topic ARN
  resource_arn = 'MY_RESOURCE_ARN' # Should be replaced with a real resource ARN
  policy_name = 'POLICY_NAME' # Typically, this is "Policy"

  sns_resource = Aws::SNS::Resource.new
  enabler = SnsResourceEnabler.new(sns_resource)

  enabler.enable_resource(topic_arn, resource_arn, policy_name)
end
```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for Ruby API リファレンスの「[SetTopicAttributes](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.  
    lo_sns->settopicattributes(  
        iv_topicarn = iv_topic_arn  
        iv_attributename = iv_attribute_name  
        iv_attributevalue = iv_attribute_value ).  
    MESSAGE 'Set/updated SNS topic attributes.' TYPE 'I'.  
CATCH /aws1/cx_snsnotfoundexception.  
    MESSAGE 'Topic does not exist.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[SetTopicAttributes](#)」を参照してください。

を使用した配信ステータスのログ記録の設定 CloudFormation

DeliveryStatusLogging を使用して を設定するには CloudFormation、JSON または YAML テンプレートを使用して CloudFormation スタックを作成します。詳細については、「CloudFormation ユーザーガイド」の「AWS::SNS::Topicリソースの DeliveryStatusLoggingプロパティ」を参照してください。以下は、新しいトピックを作成したり、Amazon SQS プロトコルのすべての DeliveryStatusLogging 属性で既存のトピックを更新したりするための JSON および YAML の CloudFormation テンプレートの例です。

SuccessFeedbackRoleArn および FailureFeedbackRoleArn で参照されている IAM ロールに、必要な CloudWatch Logs のアクセス許可があることを確認してください。

JSON

```
"Resources": {
```

```
"MySNSTopic" : {
  "Type" : "AWS::SNS::Topic",
  "Properties" : {
    "TopicName" : "TestTopic",
    "DisplayName" : "TEST",
    "SignatureVersion" : "2",
    "DeliveryStatusLogging" : [{
      "Protocol": "sqs",
      "SuccessFeedbackSampleRate": "45",
      "SuccessFeedbackRoleArn": "arn:aws:iam::123456789012:role/
SNSSuccessFeedback_test1",
      "FailureFeedbackRoleArn": "arn:aws:iam::123456789012:role/
SNSFailureFeedback_test2"
    }]
  }
}
```

YAML

```
Resources:
  MySNSTopic:
    Type: AWS::SNS::Topic
    Properties:
      TopicName: TestTopic
      DisplayName: TEST
      SignatureVersion: 2
      DeliveryStatusLogging:
        - Protocol: sqs
          SuccessFeedbackSampleRate: 45
          SuccessFeedbackRoleArn: arn:aws:iam::123456789012:role/
SNSSuccessFeedback_test1
          FailureFeedbackRoleArn: arn:aws:iam::123456789012:role/
SNSFailureFeedback_test2
```

Amazon SNS メッセージ配信の再試行

Amazon SNS は、各配信プロトコルの配信ポリシーを定義します。配信ポリシーは、サーバー側のエラーが発生したとき (サブスクライブされたエンドポイントをホストするシステムが利用できなくなったとき) に Amazon SNS がメッセージの配信を再試行する方法を定義します。配信ポリシーが

枯渇すると、Amazon SNS は配信の再試行を停止し、にデッドレターキューが添付されていない限り、メッセージを破棄します。詳細については、「[Amazon SNS デッドレターキュー](#)」を参照してください。

配信プロトコルとポリシー

Note

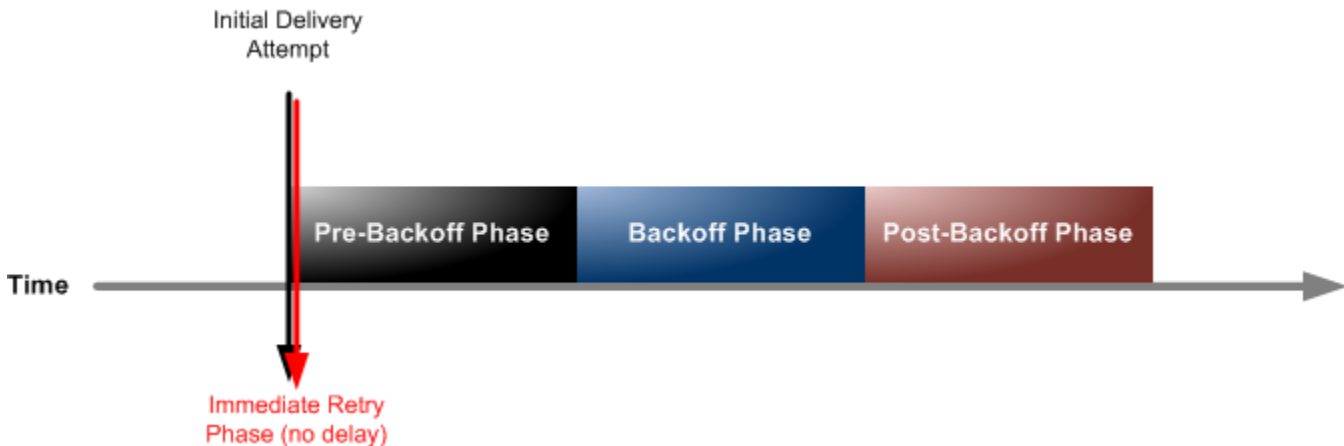
- HTTP/S を除き、Amazon SNS で定義された配信ポリシーを変更することはできません。HTTP/S のみがカスタムポリシーをサポートしています。「[HTTP/S 配信ポリシーの作成](#)」を参照してください。
- Amazon SNS は、配信の再試行にジッタリングを適用します。詳細については、『AWS アーキテクチャブログ』の「[エクスポネンシャルバックオフとジッター](#)」を参照してください。
- HTTP/S エンドポイントのポリシー再試行時間は、全体で 3,600 秒を超えることはできません。これはハード制限であり、時間を延長することはできません。

エンドポイントタイプ	配信プロトコル	即時の再試行 (遅延なし) 段階	バックオフ前段階	バックオフ段階	バックオフ後段階	合計試行回数
AWS マネージドエンドポイント	1	3 回、遅延なし	2 回、1 秒間隔で	10 回、エクスポネンシャルバックオフで 1 秒から 20 秒まで	100,000 回、20 秒間隔で	100,015 回、23 日間
	AWS Lambda					
	Amazon SQS					
カスタマー管理のエンドポイント	SMTP	0 回、遅延なし	2 回、10 秒間隔で	10 回、エクスポネンシャルバックオフで 10 秒から 600 秒まで (10 分)	38 回、600 秒 (10 分) 間隔で	50 回、6 時間以上
	SMS					
	モバイルプッシュ					

¹Firehose プロトコルのスロットリングエラーの場合、Amazon SNS はカスタマーマネージドエンドポイントと同じ配信ポリシーを使用します。

配信ポリシーの段階

次の図は、配信ポリシーの段階を示しています。



各配信ポリシーは、4 つの段階で構成されます。

1. 即時の再試行段階 (遅延なし) – 最初の配信の試行の直後に発生します。この段階では再試行間の遅延時間はありません。
2. バックオフ前段階 – この段階は即時の再試行段階の後に続きます。Amazon SNS は、バックオフ関数を適用する前に、この段階を使用して一連の再試行を試みます。この段階では、再試行回数と再試行間の遅延量を指定します。
3. バックオフ段階 - この段階では、再試行バックオフ関数を使用して、再試行間の遅延を制御します。この段階では、最小遅延、最大遅延、および遅延が最小遅延から最大遅延までどれだけ速く増加するかを定義する再試行バックオフ関数を設定します。バックオフ関数は、数論的、指数、幾何学的、または一次です。
4. バックオフ後段階 – この段階はバックオフ段階の後に続きます。再試行回数とその間の遅延量を指定します。これが最終段階です。

HTTP/S 配信ポリシーの作成

Amazon SNS が HTTP/S エンドポイントへのメッセージ配信を再試行する方法を定義できます。そのためには、遅延なし、バックオフ前、バックオフ、バックオフ後の 4 つの段階を持つ配信ポリシーを使用します。このポリシーを使用すると、デフォルトの再試行設定を上書きし、HTTP サーバーの処理能力に合わせてカスタマイズできます。

HTTP/S 配信ポリシーは、トピックレベルまたはサブスクリプションレベルのいずれかで JSON オブジェクトとして設定できます。

- トピックレベルのポリシー – トピックにリンクされたすべての HTTP/S サブスクリプションに適用されます。[CreateTopic](#) または [SetTopicAttributes](#) API アクションを使用して、このポリシーを設定します。
- サブスクリプションレベルのポリシー – 特定のサブスクリプションにのみ適用されます。[Subscribe](#) または [SetSubscriptionAttributes](#) API アクションを使用して、このポリシーを設定します。

または、CloudFormation テンプレートで [AWS::SNS::Subscription](#) リソースを使用することもできます。

HTTP/S サーバーの処理能力に基づいて配信ポリシーをカスタマイズする必要があります。

- すべてのサブスクリプションで 1 つのサーバーを使用 – トピック内のすべての HTTP/S サブスクリプションが同じサーバーを使用している場合は、配信ポリシーをトピック属性として設定して、すべてのサブスクリプション間で一貫性を確保します。
- サブスクリプションごとに異なるサーバーを使用 – 各サブスクリプションが異なるサーバーをターゲットにしている場合は、特定のサーバーの処理能力に合わせて、サブスクリプションごとに固有の配信ポリシーを作成します。

リクエストポリシーに Content-Type ヘッダーを設定して、通知のメディアタイプを指定することもできます。デフォルトでは、Amazon SNS は、コンテンツタイプが text/plain; charset=UTF-8 に設定された HTTP/S エンドポイントにすべての通知を送信します。ただし、リクエストポリシーの [headerContentType](#) フィールドを使用して、このデフォルト設定を上書きできます。

次の JSON オブジェクトは、4 つの段階で構成された再試行を持つ配信ポリシーを定義します。

1. 遅延なし段階 – 直ちに 3 回再試行します。
2. バックオフ前段階 – 1 秒間隔で 2 回再試行します。
3. バックオフ段階 – 1~60 秒の範囲の指数関数的遅延で 10 回再試行します。
4. バックオフ後段階 – 60 秒の一定間隔で 35 回再試行します。

Amazon SNS では、メッセージを配信するために合計 50 回試行し、その後でメッセージを破棄します。すべての再試行の終了後に配信できないメッセージを保持するには、配信できないメッセージをデッドレターキュー (DLQ) に移動するようにサブスクリプションを設定します。詳細については、「[Amazon SNS デッドレターキュー](#)」を参照してください。

Amazon SNS では、すべての 5XX エラーと 429 エラー (送信されたリクエストが多すぎます) を再試行可能なエラーとして扱います。これらのエラーには、配信ポリシーが適用されます。他のすべてのエラーは永続的なエラーとして扱われ、再試行は行われません。

Note

この配信ポリシーは `maxReceivesPerSecond` プロパティを使用して、各サブスクリプションで 1 秒あたり平均 10 件のメッセージになるように配信トラフィックをスロットリングします。このメカニズムは、高トラフィックによって HTTP/S エンドポイントの負荷が高くなるのを防ぐ際に役立ちますが、平均配信レートを維持するように設計されており、厳密な上限を課すものではありません。特に発行レートがスロットリング制限よりも大幅に高い場合、配信トラフィックが指定された制限を超えることがあります。

発行 (インバウンド) トラフィックが配信 (アウトバウンド) レートを超過すると、メッセージバックログが発生し、配信レイテンシーが高くなる可能性があります。このような問題を回避するには、`maxReceivesPerSecond` の値が、HTTP/S サーバーの処理能力とワークロードの要件に合致していることを確認してください。

次の配信ポリシーの例では、HTTP/S 通知のデフォルトのコンテンツタイプを `application/json` に上書きします。

```
{
  "healthyRetryPolicy": {
    "minDelayTarget": 1,
    "maxDelayTarget": 60,
    "numRetries": 50,
    "numNoDelayRetries": 3,
    "numMinDelayRetries": 2,
    "numMaxDelayRetries": 35,
    "backoffFunction": "exponential"
  },
  "throttlePolicy": {
    "maxReceivesPerSecond": 10
  },
  "requestPolicy": {
```

```

    "headerContentType": "application/json"
  }
}
```

配信ポリシーは、再試行ポリシー、スロットルポリシー、リクエストポリシーで構成されます。配信ポリシーには、合計で 9 つの属性があります。

ポリシー	説明	制約
minDelayTarget	再試行の最小遅延。 単位: 秒	1 から最大遅延まで デフォルト: 20
maxDelayTarget	再試行の最大遅延。 単位: 秒	最小遅延から 3,600 まで デフォルト: 20
numRetries	即時再試行、バックオフ前再試行、バックオフ再試行、ポストバックオフ再試行の合計数。	0 ~ 100 デフォルト: 3
numNoDelayRetries	即時に行う再試行の回数。再試行の間に遅延はありません。	0 以上 デフォルト: 0
numMinDelayRetries	バックオフ前段階での再試行回数と、これらの間に指定された最小遅延時間。	0 以上 デフォルト: 0
numMaxDelayRetries	バックオフ後段階での再試行回数と、その間の最大遅延。	0 以上 デフォルト: 0
backoffFunction	再試行間のバックオフのモデル。	次の 4 つのオプションのいずれか。 • 数論的 • 指数 • 幾何学的

ポリシー	説明	制約
		一次 デフォルト: linear
maxReceivesPerSecond	各サブスクリプションにおける 1 秒あたりのメッセージ配信の最大平均数。	1 以上 デフォルト: スロットリングなし (配信レートに制限なし)

ポリシー	説明	制約
headerContentType	HTTP/S エンドポイントに送信される通知のコンテンツタイプ。	リクエストポリシーが定義されていない場合、コンテンツタイプはデフォルトで <code>text/plain; charset=UTF-8</code> に設定されます。 サブスクリプションに対する raw メッセージ配信が無効になっている場合 (デフォルト)、または配信ポリシーがトピックレベルで定義されている場合、サポートされるヘッダーコンテンツタイプは <code>application/json</code> および <code>text/plain</code> です。 サブスクリプションに対する raw メッセージ配信を有効にすると、以下のコンテンツタイプがサポートされます。 • <code>text/css</code> • <code>text/csv</code> • <code>text/html</code> • <code>text/plain</code> • <code>text/xml</code> • <code>application/atom+xml</code> • <code>application/json</code> • <code>application/octet-stream</code> • <code>application/soap+xml</code> • <code>application/x-www-form-urlencoded</code> • <code>application/xhtml+xml</code>

ポリシー	説明	制約
		application/xml

Amazon SNS は、次の式を使用して、バックオフ段階での再試行回数を計算します。

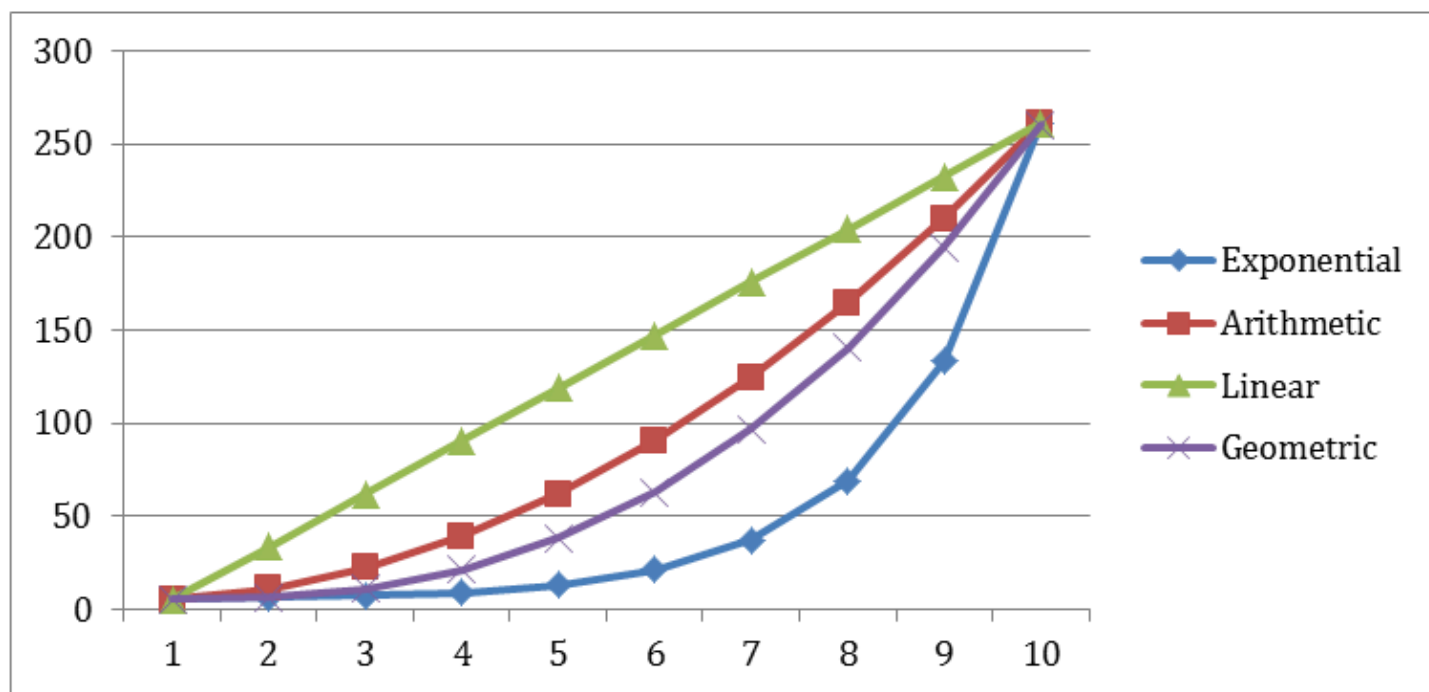
```
numRetries - numNoDelayRetries - numMinDelayRetries - numMaxDelayRetries
```

次の 3 つのパラメータを使用して、バックオフ段階での再試行の頻度を制御できます。

- **minDelayTarget** – バックオフ段階における最初の再試行の遅延を設定します。
- **maxDelayTarget** – バックオフ段階における最後の再試行の遅延を設定します。
- **backoffFunction** – 最初の再試行と最後の再試行の間に行われるすべての再試行について、遅延を計算するために Amazon SNS で使用されるアルゴリズムを決定します。利用可能な 4 つの再試行バックオフ関数から選択できます。

次の図は、さまざまな再試行バックオフ関数が、バックオフ段階における再試行間の遅延にどのように影響するかを示しています。この例で使用される配信ポリシーには、合計再試行回数 10 回、最小遅延時間 5 秒、最大遅延時間 260 秒の設定が含まれます。

- 縦軸は、再試行ごとの遅延 (秒単位) を示します。
- 横軸は、最初の試行から 10 回目の試行までの再試行シーケンスを示します。



Amazon SNS デッドレターキュー

デッドレターキューは、Amazon SNS サブスクリプションが受信者に正常に配信できないメッセージの送信先としての Amazon SQS キューです。クライアントエラーまたはサーバーエラーが原因で配信できないメッセージは、詳細な分析や再処理のためにデッドレターキューに保持されます。詳細については、「[サブスクリプションの Amazon SNS デッドレターキューを設定する](#)」および「[Amazon SNS メッセージ配信の再試行](#)」を参照してください。

Note

- Amazon SNS サブスクリプションと Amazon SQS キューは、同じ AWS アカウントとリージョンにある必要があります。
- [FIFO トピック](#)では、Amazon SNS サブスクリプションのデッドレターキューとして Amazon SQS キューを使用できます。FIFO トピックサブスクリプションでは FIFO キューを使用します。標準トピックサブスクリプションでは標準キューを使用します。
- 暗号化された Amazon SQS キューをデッドレターキューとして使用するには、Amazon SNS サービスプリンシパルに AWS KMS API アクションへのアクセスを許可するキーポリシーを持つカスタム KMS を使用する必要があります。詳細については、このガイドの「[サーバー側の暗号化を使用した Amazon SNS データの保護](#)」と、『Amazon Simple

Queue Service デベロッパーガイド』の「[サーバー側の暗号化 \(SSE\) と AWS KMSを使用した Amazon SQS データの保護](#)」を参照してください。

メッセージ配信が失敗する理由

一般に、Amazon SNS がクライアント側またはサーバー側のエラーにより、サブスクライブされたエンドポイントにアクセスできない場合、メッセージの配信は失敗します。Amazon SNS がクライアント側のエラーを受信した場合、または対応する再試行ポリシーで指定された再試行回数を超えるメッセージに対してサーバー側のエラーを受信し続ける場合、Amazon SNS はメッセージを廃棄します。ただし、デッドレターキューがサブスクリプションに添付されている場合を除きます。配信に失敗しても、サブスクリプションのステータスは変更されません。詳細については、「[Amazon SNS メッセージ配信の再試行](#)」を参照してください。

クライアント側エラー

Amazon SNS に古いサブスクリプションのメタデータがあると、クライアント側のエラーが発生する可能性があります。これらのエラーは、通常、所有者がエンドポイント (Amazon SNS トピックにサブスクライブされている Lambda 関数など) を削除した場合や、サブスクライブしているエンドポイントに添付されているポリシーを、Amazon SNS がエンドポイントにメッセージを配信できないように所有者が変更した場合に発生します。Amazon SNS は、クライアント側のエラーの結果として失敗したメッセージ配信を再試行しません。

サーバー側のエラー

サーバー側のエラーは、サブスクライブされたエンドポイントを担当するシステムが利用できなくなったり、Amazon SNS からの有効なリクエストを処理できないことを示す例外を返す場合に発生します。サーバー側のエラーが発生すると、Amazon SNS は一次バックオフ関数またはエクスポネンシャルバックオフ関数を使用して、失敗した配信を再試行します。Amazon SQS または によってバックアップされた AWS マネージドエンドポイントに起因するサーバー側のエラーの場合 AWS Lambda、Amazon SNS は 23 日間で最大 100,015 回配信を再試行します。

カスタマー管理のエンドポイント (HTTP、SMTP、SMS、モバイルプッシュなど) も、サーバー側のエラーを引き起こす可能性があります。Amazon SNS は、これらのタイプのエンドポイントへの配信も再試行します。HTTP エンドポイントはお客様定義の再試行ポリシーをサポートしますが、Amazon SNS は SMTP、SMS、およびモバイルプッシュエンドポイントに対して、内部配信再試行ポリシーを 6 時間にわたって 50 回に設定します。

デッドレターキューのしくみ

メッセージの配信はサブスクリプションレベルで行われるため、デッドレターキューは (トピックではなく) Amazon SNS サブスクリプションに添付されます。これにより、各メッセージの元のターゲットエンドポイントをより簡単に識別できます。

Amazon SNS サブスクリプションに関連付けられたデッドレターキューは、通常の Amazon SQS キューです。メッセージの保持期間の詳細については、『Amazon Simple Queue Service デベロッパーガイド』の「[メッセージに関連するクォータ](#)」を参照してください。Amazon SQS [SetQueueAttributes](#) API アクションを使用して、メッセージの保持期間を変更できます。アプリケーションの復元性を高めるために、デッドレターキューの最大保持期間を 14 日に設定することをお勧めします。

メッセージがデッドレターキューに移動する仕組み

メッセージは、再処理ポリシーを使用してデッドレターキューに移動されます。再処理ポリシーは、デッドレターキューの ARN を参照する JSON オブジェクトです。deadLetterTargetArn 属性は ARN を指定します。ARN は、Amazon SNS サブスクリプションと同じ およびリージョンにある Amazon SQS Amazon SNS キューを指す必要があります。AWS アカウント 詳細については、「[サブスクリプションの Amazon SNS デッドレターキューを設定する](#)」を参照してください。

次の JSON オブジェクトは、SNS サブスクリプションに添付された再処理ポリシーのサンプルです。

```
{
  "deadLetterTargetArn": "arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue"
}
```

メッセージをデッドレターキューから移動する方法

メッセージをデッドレターキューから移動するには、次の 2 つの方法があります。

- Amazon SQS コンシューマロジックの作成を避ける - デッドレターキューをイベントソースとして Lambda 関数に設定して、デッドレターキューを吸い出します。
- Amazon SQS コンシューマーロジックの書き込み - Amazon SQS API、AWS SDK、またはを使用して、デッドレターキュー内のメッセージをポーリング、処理、削除するためのカスタムコンシューマーロジックを AWS CLI 書き込みます。

デッドレターキューのモニタリングとログ記録方法

Amazon CloudWatch メトリクスを使用して、Amazon SNS サブスクリプションに関連付けられたデッドレターキューをモニタリングできます。すべての Amazon SQS キューは 1 分間隔で CloudWatch メトリクスを送信します。詳細については、「[Amazon Simple Queue Service デベロッパーガイド](#)」の「[Amazon SQS で使用できる CloudWatch メトリクス](#)」を参照してください。デッドレターキューを持つすべての Amazon SNS サブスクリプションも、CloudWatch メトリクスを出力します。詳細については、「[CloudWatch を使用した Amazon SNS のモニタリング](#)」を参照してください。

デッドレターキューのアクティビティを通知するには、CloudWatch メトリクスとアラームを使用できます。NumberOfMessagesSent メトリクスのアラームを設定するのは適切ではありません。このメトリクスは、処理の試行が失敗した場合に DLQ に送信されるメッセージをキャプチャしないためです。ここでは、ApproximateNumberOfMessagesVisible メトリクスを使用します。このメトリクスは、処理の失敗のために移動されたメッセージも含め、DLQ で現在利用可能なすべてのメッセージをキャプチャします。

CloudWatch アラーム設定の例

1. **ApproximateNumberOfMessagesVisible** メトリクスの [CloudWatch アラーム](#) を作成します。
2. アラームしきい値を 1 (または期待と DLQ トラフィックに基づく別の適切な値) に設定します。
3. アラームが作動したときに通知される Amazon SNS トピックを指定します。この Amazon SNS トピックは、任意のエンドポイントタイプ (E メールアドレス、電話番号、モバイルポケットベルアプリなど) にアラーム通知を配信できます。

CloudWatch Logs を使用して、Amazon SNS 配信が失敗する原因となる例外や、デッドレターキューに送信されるメッセージを調査できます。Amazon SNS は、配信の成功と失敗の両方を CloudWatch に記録できます。詳細については、「[Amazon SNS モバイルアプリの属性](#)」を参照してください。

サブスクリプションの Amazon SNS デッドレターキューを設定する

デッドレターキューは、Amazon SNS サブスクリプションが受信者に正常に配信できないメッセージの送信先としての Amazon SQS キューです。クライアントエラーまたはサーバーエラーが原因で配信できないメッセージは、詳細な分析や再処理のためにデッドレターキューに保持されます。詳細については、「[Amazon SNS デッドレターキュー](#)」および「[Amazon SNS メッセージ配信の再試行](#)」を参照してください。

このページでは AWS マネジメントコンソール、AWS SDK、AWS CLI、および CloudFormation を使用して、Amazon SNS サブスクリプションのデッドレターキューを設定する方法を示します。

Note

[FIFO トピック](#)では、Amazon SNS サブスクリプションのデッドレターキューとして Amazon SQS キューを使用できます。FIFO トピックサブスクリプションでは FIFO キューを使用し、標準トピックサブスクリプションでは標準キューを使用します。

前提条件

デッドレターキューを設定するには、次の前提条件を満たしている必要があります。

1. MyTopic という名前で [Amazon SNS トピックを作成する](#)。
2. MyEndpoint という名前で [Amazon SQS キューを作成し](#)、Amazon SNS サブスクリプションのエンドポイントとして使用します。
3. (スキップ CloudFormation) [キューをトピックにサブスクライブします](#)。
4. MyDeadLetterQueue という名前で [Amazon SQS キューを作成し](#)、Amazon SNS サブスクリプションのデッドレターキューとして使用します。
5. Amazon SQS API アクションへのアクセスを Amazon SNS プリンシパルに付与するには、MyDeadLetterQueue に次のキューポリシーを設定します。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "SQS:SendMessage",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:MyTopic"
        }
      }
    }
  ]
}
```

```
}  
}  
}  
]  
}
```

を使用して Amazon SNS サブスクリプションのデッドレターキューを設定するには AWS マネジメントコンソール

このチュートリアルを開始する前に、次の[前提条件](#)を完了してください。

1. [Amazon SQS コンソール](#)にサインインします。
2. [Amazon SQS キューを作成する](#)か、既存のキューを使用して、キューの [詳細] タブでキューの ARN を確認します。次に例を示します。

```
arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue
```

3. [Amazon SNS コンソール](#)にサインインします。
4. ナビゲーションパネルで、[サブスクリプション] を選択します。
5. [サブスクリプション] ページで、既存のサブスクリプションを選択し、[編集] を選択します。
6. [Edit **1234a567-bc89-012d-3e45-6fg7h890123i**] ページで、[再処理ポリシー (デッドレターキュー)] セクションを展開し、次の操作を行います。
 - a. [有効] を選択します。
 - b. Amazon SQS キューの ARN を指定します。
7. [変更を保存] をクリックします。

デッドレターキューを使用するようにサブスクリプションが設定されます。

AWS SDK を使用して Amazon SNS サブスクリプションのデッドレターキューを設定するには

この例を実行する前に、必ず[前提条件](#)を完了してください。

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、「AWS SDK とツールのリファレンスガイド」の「[共有設定ファイルと認証情報ファイル](#)」を参照してください。

次の例は、SetSubscriptionAttributesRedrivePolicy を使用する方法を説明しています。

Java

SDK for Java 1.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。

```
// Specify the ARN of the Amazon SNS subscription.
String subscriptionArn =
    "arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-
bc89-012d-3e45-6fg7h890123i";

// Specify the ARN of the Amazon SQS queue to use as a dead-letter queue.
String redrivePolicy =
    "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-2:123456789012:MyDeadLetterQueue\"}";

// Set the specified Amazon SQS queue as a dead-letter queue
// of the specified Amazon SNS subscription by setting the RedrivePolicy
// attribute.
SetSubscriptionAttributesRequest request = new SetSubscriptionAttributesRequest()
    .withSubscriptionArn(subscriptionArn)
    .withAttributeName("RedrivePolicy")
    .withAttributeValue(redrivePolicy);
sns.setSubscriptionAttributes(request);
```

を使用して Amazon SNS サブスクリプションのデッドレターキューを設定するには
AWS CLI

このチュートリアルを開始する前に、次の[前提条件](#)を完了してください。

1. AWS CLIをインストールして設定します。詳細については、「[AWS Command Line Interface ユーザーガイド](#)」を参照してください。
2. 次のコマンドを使用します。

```
aws sns set-subscription-attributes \  
--subscription-arn arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-  
bc89-012d-3e45-6fg7h890123i  
--attribute-name RedrivePolicy  
--attribute-value "{\"deadLetterTargetArn\": \"arn:aws:sqs:us-  
east-2:123456789012:MyDeadLetterQueue\"}"
```

を使用して Amazon SNS サブスクリプションのデッドレターキューを設定するには CloudFormation

このチュートリアルを開始する前に、次の[前提条件](#)を完了してください。

1. 次のJSON コードをMyDeadLetterQueue.jsonという名前のファイルにコピーします。

```
{  
  "Resources": {  
    "mySubscription": {  
      "Type" : "AWS::SNS::Subscription",  
      "Properties" : {  
        "Protocol": "sqs",  
        "Endpoint": "arn:aws:sqs:us-east-2:123456789012:MyEndpoint",  
        "TopicArn": "arn:aws:sns:us-east-2:123456789012:MyTopic",  
        "RedrivePolicy": {  
          "deadLetterTargetArn":  
            "arn:aws:sqs:us-east-2:123456789012:MyDeadLetterQueue"  
        }  
      }  
    }  
  }  
}
```

2. [CloudFormation コンソール](#) にサインインします。
3. [テンプレートの選択] ページで、[テンプレートを Amazon S3 にアップロード] を選択した後、MyDeadLetterQueue.json ファイルを選択し、次に [次へ] を選びます。
4. [詳細の指定] ページで、[スタック名] に MyDeadLetterQueue を入力してから、[次へ] を選択します。
5. [オプション] ページで、[次へ] を選択します。
6. [Review] ページで、[作成] を選択します。

CloudFormation はMyDeadLetterQueueスタックの作成を開始し、CREATE_IN_PROGRESS ステータスを表示します。プロセスが完了すると、 は CREATE_COMPLETE ステータス CloudFormation を表示します。

Amazon SNS メッセージのアーカイブ、リプレイ、および分析

Amazon SNS 標準トピックは、メッセージのアーカイブをサポートします。Firehose 配信ストリームに通知をファンアウトできます。これにより、Amazon Simple Storage Service (Amazon S3)、Amazon Redshift など、Firehose がサポートするストレージおよび分析送信先に通知を送信できます。

Amazon SNS FIFO トピックは、トピック所有者がトピックに発行されたメッセージを最大 365 日間保存 (またはアーカイブ) できる、ノーコードのインプレースメッセージアーカイブをサポートしています。アクティブ ArchivePolicy を使用するトピックの場合、サブスクライバーは ReplayPolicy を作成して、アーカイブされたメッセージをサブスクライブされたエンドポイントに対してリトリート (またはリプレイ) できます。この機能の詳細は、「[FIFO トピックの Amazon SNS メッセージのアーカイブとリプレイ](#)」を参照してください。

機能	標準トピック	FIFO トピック
メッセージのアーカイブ	Firehose 配信ストリームへのファンアウト	FIFO トピック所有者の Amazon SNS メッセージアーカイブ
メッセージのリプレイ	標準トピックのリプレイは組み込み機能ではありません。多くのお客様は、メッセージアーカイブに基づいて独自の機能を構築しています。	FIFO トピックサブスクライバーの Amazon SNS メッセージリプレイ

Amazon SNS でのリソース管理と最適化

このトピックでは、最適なパフォーマンスを確保し、不要なコストを削減して、適切に整理されたリソースを維持し、Amazon SNS の可能性を最大限に活用する方法について説明します。

トピック

- [Amazon SNS トピックのタグ付け](#)

Amazon SNS トピックのタグ付け

Amazon SNS では、Amazon SNS トピックのタグ付けがサポートされています。これにより、トピックに関連するコストを追跡および管理し、AWS Identity and Access Management (IAM) ポリシーでセキュリティを強化し、数千のトピックを簡単に検索またはフィルタリングできます。タグ付けにより、AWS Resource Groups を使用して Amazon SNS トピックを管理できます。Resource Groups の詳細については、[「AWS Resource Groups ユーザーガイド」](#)を参照してください。

コスト割り当てのタグ付け

コスト割り当てのために Amazon SNS トピックを整理および識別するには、トピックの目的を識別するタグを追加することができます。これは特に、多くのトピックを持っている場合に便利です。コスト配分タグを使用して、独自のコスト構造を反映するように AWS 請求書を整理できます。これを行うには、サインアップして AWS アカウント請求書を取得し、タグキーと値を含めます。詳細については、[「AWS Billing and Cost Management User Guide」](#)の[「Setting Up a Monthly Cost Allocation Report」](#)を参照してください。

たとえば、次のように Amazon SNS のトピックのコストセンターや目的を表すタグを追加することができます。

[リソース]	キー	値
トピック 1	コストセンター	43289
	アプリケーション	注文処理
トピック 2	コストセンター	43289
	アプリケーション	お支払い方法

[リソース]	キー	値
トピック 3	コストセンター	76585
	アプリケーション	アーカイブ

このタグ付けスキームでは、関連するタスクを実行している 2 つのトピックを同じコストセンターにグループ化する一方で、関連性のないアクティビティには別のコスト割り当てタグでタグ付けすることができます。

アクセス制御のタグ付け

AWS Identity and Access Management は、タグに基づくリソースへのアクセスの制御をサポートしています。リソースにタグ付け後、IAM ポリシーの条件要素でリソース タグに関する情報を提供し、タグベースのアクセスを管理します。[Amazon SNS console](#) (Amazon SNS コンソール) または [AWS SDK](#) を使用してリソースにタグ付けする方法については、「[Configuring tags](#)」(タグを設定する) を参照してください。

IAM アイデンティティのアクセスを制限できます。たとえば、キー `environment` および値 `production` のタグを含むすべての Amazon SNS トピックへの `Publish` および `PublishBatch` のアクセスを制限し、他のすべての Amazon SNS トピックへのアクセスを許可することができます。次の例では、ポリシーにより、`production` でタグ付けされたトピックにメッセージを公開する機能が制限され、`development` でタグ付けされた トピックにメッセージを公開できるようになります。詳細については、IAM ユーザーガイドの [\[Controlling Access Using Tags\]](#) (タグを使用したアクセス制御) を参照してください。

Note

`Publish` の IAM の許可を設定すると、`Publish` および `PublishBatch` の両方に権限が設定されます。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
```

```
    "Action": [
      "sns:Publish"
    ],
    "Resource": "arn:aws:sns:*:*:*:",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/environment": "production"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "sns:Publish"
    ],
    "Resource": "arn:aws:sns:*:*:*:",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/environment": "development"
      }
    }
  }
}]
}
```

リソース検索およびフィルタリングのタグ付け

AWS アカウントには、数万の Amazon SNS トピックを含めることができます (詳細については、[Amazon SNS クォータ](#) を参照してください)。トピックにタグ付けすることで、トピックの検索やフィルタリングのプロセスを簡素化することができます。

たとえば、本番環境に関連するトピックが数百個ある場合があります。これらのトピックを手動で検索するのではなく、指定されたタグを持つすべてのトピックを照会することができます。

```
import com.amazonaws.services.resourcegroups.AWSResourceGroups;
import com.amazonaws.services.resourcegroups.AWSResourceGroupsClientBuilder;
import com.amazonaws.services.resourcegroups.model.QueryType;
import com.amazonaws.services.resourcegroups.model.ResourceQuery;
import com.amazonaws.services.resourcegroups.model.SearchResourcesRequest;
import com.amazonaws.services.resourcegroups.model.SearchResourcesResult;

public class Example {
```

```
public static void main(String[] args) {
    // Query Amazon SNS Topics with tag "keyA" as "valueA"
    final String QUERY = "{\"ResourceTypeFilters\":[\"AWS::SNS::Topic\"],\n\"TagFilters\": [{\"Key\": \"keyA\", \"Values\": [\"valueA\"]}]}\"";

    // Initialize ResourceGroup client
    AWSResourceGroups awsResourceGroups = AWSResourceGroupsClientBuilder
        .standard()
        .build();

    // Query all resources with certain tags from ResourceGroups
    SearchResourcesResult result = awsResourceGroups.searchResources(
        new SearchResourcesRequest().withResourceQuery(
            new ResourceQuery()
                .withType(QueryType.TAG_FILTERS_1_0)
                .withQuery(QUERY)
        ));
    System.out.println("SNS Topics with certain tags are " +
        result.getResourceIdentifiers());
}
```

Amazon SNS トピックタグを設定

このトピックでは、AWS SDK AWS マネジメントコンソール、または [Amazon SNS トピック](#) を使用して [Amazon SNS トピック](#) のタグを設定する方法について説明します AWS CLI。

Important

個人情報 (PII) などの機密情報や秘匿性の高い情報はタグに追加しないようにします。タグは、他の Amazon Web Services のサービス (請求など) からアクセスできます。タグは、プライベートデータまたは機密データに使用することを目的としたものではありません。

を使用した Amazon SNS トピックのタグの一覧表示、追加、削除 AWS マネジメントコンソール

1. [Amazon SNS コンソール](#) にサインインします。
2. ナビゲーションパネルで、[トピック] を選択します。
3. [トピック] ページで、トピックを選択して [編集] を選択します。

4. [タグ] セクションを展開します。

トピックに追加されているタグが一覧表示されます。

5. トピックのタグを変更します。

- タグを追加するには、[Add tag] (タグの追加) を選択し、[Key] (キー) と [Value] (値) (オプション) を入力します。
- タグを削除するには、キー値ペアの横にある [タグの削除] を選択します。

6. [Save changes] (変更の保存) をクリックします。

AWS SDK を使用してトピックにタグを追加する

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、「AWS SDK とツールのリファレンスガイド」の「[共有設定ファイルと認証情報ファイル](#)」を参照してください。

次のサンプルコードは、TagResource を使用する方法を説明しています。

CLI

AWS CLI

トピックにタグを追加するには

次の tag-resource の例では、指定した Amazon SNS トピックにメタデータタグを追加します。

```
aws sns tag-resource \  
  --resource-arn arn:aws:sns:us-west-2:123456789012:MyTopic \  
  --tags Key=Team,Value=Alpha
```

このコマンドでは何も出力されません。

- API の詳細については、AWS CLI コマンドリファレンスの「[TagResource](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.Tag;
import software.amazon.awssdk.services.sns.model.TagResourceRequest;
import java.util.ArrayList;
import java.util.List;

/**
 * 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
 */
public class AddTags {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicArn>

            Where:
                topicArn - The ARN of the topic to which tags are added.

            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
    }
}
```

```
String topicArn = args[0];
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();

addTopicTags(snsClient, topicArn);
snsClient.close();
}

public static void addTopicTags(SnsClient snsClient, String topicArn) {
    try {
        Tag tag = Tag.builder()
            .key("Team")
            .value("Development")
            .build();

        Tag tag2 = Tag.builder()
            .key("Environment")
            .value("Gamma")
            .build();

        List<Tag> tagList = new ArrayList<>();
        tagList.add(tag);
        tagList.add(tag2);

        TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
            .resourceArn(topicArn)
            .tags(tagList)
            .build();

        snsClient.tagResource(tagResourceRequest);
        System.out.println("Tags have been added to " + topicArn);

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```


- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[TagResource](#)」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 addTopicTags(topicArn: String) {
    val tag =
        Tag {
            key = "Team"
            value = "Development"
        }

    val tag2 =
        Tag {
            key = "Environment"
            value = "Gamma"
        }

    val tagList = mutableListOf<Tag>()
    tagList.add(tag)
    tagList.add(tag2)

    val request =
        TagResourceRequest {
            resourceArn = topicArn
            tags = tagList
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.tagResource(request)
        println("Tags have been added to $topicArn")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[TagResource](#)」を参照してください。

Amazon SNS API アクションによるタグの管理

Amazon SNS API を使ってタグを管理するには、以下の API アクションを使用します。

- [ListTagsForResource](#)
- [TagResource](#)
- [UntagResource](#)

ABAC をサポートする API アクション

以下に、属性ベースのアクセスコントロール (ABAC) をサポートする API アクションの一覧を示します。ABAC の詳細については、IAM ユーザーガイドの「[ABAC とは AWS](#)」を参照してください。

- [AddPermission](#)
- [ConfirmSubscription](#)
- [DeleteTopic](#)
- [GetDataProtectionPolicy](#)
- [GetSubscriptionAttributes](#)
- [GetTopicAttributes](#)
- [ListSubscriptionsByTopic](#)
- [ListTagsForResource](#)
- [Publish](#)
- [PublishBatch](#)
- [PutDataProtectionPolicy](#)
- [RemovePermission](#)
- [SetSubscriptionAttributes](#)
- [SetTopicAttributes](#)
- [Subscribe](#)
- [TagResource](#)

- [Unsubscribe](#)
- [UntagResource](#)

Amazon SNS イベントのソースと送信先

Amazon SNS は、イベント駆動型通知をルーティングすることで、AWS のサービス と外部システムを接続します。Amazon SNS は AWS のサービス、データパイプラインの更新、Amazon EC2 スケーリングアクション、セキュリティアラートなど、さまざまな からイベントを受信し、これらのイベントを Amazon SNS トピックに発行します。これらのトピックは、指定された送信先に通知を送信します。

Amazon SNS は、[Application-to-Application \(A2A\)](#) と [Application-to-Person \(A2P\)](#) の 2 つの主なタイプの送信先をサポートします。A2A メッセージングでは、Amazon SNS は Lambda にイベントを送信してカスタムビジネスロジックをトリガーしたり、Amazon SQS にイベントを送信してメッセージをキューイングしたり、Amazon Data Firehose にイベントを送信してデータをストレージサービスや分析サービスにストリーミングしたりできます。A2P メッセージングでは、Amazon SNS は SMS、E メール、プッシュ通知を介してモバイルデバイスに通知を送信でき、ユーザーやチームがタイムリーにアラートを受信できます。

中央ハブとして機能することで、Amazon SNS は通知を適切な場所にルーティングし、AWS インフラストラクチャをより効果的に自動化および管理できるようにします。この設定により、サービス間のシームレスな統合と、ユーザーやシステムとの信頼性の高い通信を実現できます。

Amazon SNS イベントソース

Amazon SNS は、さまざまなカテゴリ AWS のサービス にわたる幅広い と統合されるため、これらのサービスは Amazon SNS トピックにイベントを発行できます。この統合により、インフラストラクチャの変更、アプリケーションパフォーマンス、コスト管理などの、重要なイベントをリアルタイムで通知できます。

Note

2020 年 10 月に Amazon SNS は [FIFO トピック](#)を導入しました。現在、ほとんどの AWS サービスは標準トピックへのイベントの送信のみをサポートしています。

分析サービス

次の表は、Amazon SNS を Athena、Amazon Redshift などの AWS 分析サービスと統合して AWS Data Pipeline、コントロール制限違反、パイプラインステータスの更新、データウェアハウスアクティビティなど、主要なイベントをリアルタイムで通知する方法を示しています。

これらの統合を活用して、レスポンスを自動化し、データオペレーションを効果的に監視できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon Athena - 標準 SQL を使用した Amazon S3 内のデータの分析ができます。	制御制限を超えたときに通知を受信します。詳細については、『Amazon Athena ユーザーガイド』の「 データ使用量管理制限の設定 」を参照してください。
AWS Data Pipeline - データの移動と変換を自動化するのに役立ちます。	パイプラインコンポーネントのステータスに関する通知を受信します。詳細については、『AWS Data Pipeline デベロッパーガイド』の「 SnsAlarm 」を参照してください。
Amazon Redshift - データウェアハウスを設定、操作、および拡張するためのすべての作業を管理します。	Amazon Redshift イベントの通知を受信します。詳細については、「Amazon Redshift 管理ガイド」の「 Amazon Redshift イベント通知 」を参照してください。

アプリケーション統合サービス

次の表はAmazon SNS が EventBridge や などのアプリケーション統合サービスと統合し AWS Step Functions、ビジネスクリティカルなアプリケーションのリアルタイムのデータルーティングと通知を有効にする方法を示しています。

これらの統合を活用して、EventBridge イベントからアラートを受信したり、Step Functions を使用してワークフローをオーケストレーションしたり、アプリケーションの自動化と応答性を強化したりできます。

AWS のサービス	Amazon SNS を使用する利点
Amazon EventBridge – 独自のアプリケーション、software-as-a-service (SaaS) アプリケーション、AWS サービスからリアルタイムデータのストリームを配信し、そのデータを Amazon SNS などのターゲットにルーティン	EventBridge イベントの通知を受信します。詳細については、『Amazon EventBridge ユーザーガイド』の「 Amazon EventBridge ターゲット 」を参照してください。

AWS のサービス	Amazon SNS を使用する利点
グします。EventBridge は、以前は CloudWatch Events と呼ばれていました。	
AWS Step Functions – AWS Lambda 関数やその他の AWS サービスを組み合わせ、ビジネスクリティカルなアプリケーションを構築できます。	Step Functions イベントの通知を受信します。詳細については、『AWS Step Functions デベロッパーガイド』の「Step Functions で Amazon SNS を呼び出す」を参照してください。

請求情報とコスト管理サービス

次の表は、を Amazon SNS と AWS Billing and Cost Management 統合して、予算、価格変更、コスト異常の通知を提供する方法を示しています。

この統合を活用して Amazon SNS トピックをセットアップし、AWS 支出に関するリアルタイムのアラートを受け取ることができます。これにより、コストをモニタリングし、予想しない料金に効率的に対応できます。

AWS のサービス	Amazon SNS を使用する利点
AWS Billing and Cost Management - コストと請求書の支払いのモニタリングを支援する機能を提供します。	予算通知、料金変更通知、異常アラートを受信します。詳細については、「AWS Billing ユーザーガイド」の次のページを参照してください。 • 予算の通知に関する Amazon SNS トピックを作成する • 通知の設定 • を使用した異常な支出の検出 AWS Cost Anomaly Detection

ビジネスアプリケーションサービス

次の表は、Amazon Chime と Amazon SNS の統合によって、重要な会議イベントの通知を送信する方法を示しています。これにより、常にコミュニケーションとスケジュールの最新情報を取得できます。

この統合を活用し、Amazon Chime SDK イベント通知を活用して、組織内外のコラボレーションツールを強化できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon Chime - 組織の内外を問わず、会い、チャットをし、仕事の電話をかけられます。	重要な会議イベント通知を受信します。詳細については、『Amazon Chime デベロッパーガイド』の「 Amazon Chime SDK イベント 」を参照してください。

コンピューティングサービス

次の表は、Amazon SNS とさまざまな AWS コンピューティングサービスとの統合を示しています。これにより、自動スケーリングアクション、EC2 Image Builder の完了、Elastic Beanstalk 環境の変更、Lambda 関数出力、Lightsail メトリクスしきい値など、主要なイベントの通知を受信できます。

これらの統合を活用して、AWS のサービス全体の重要な更新やアクションに関する情報を取得すると、アプリケーションとリソースを効率的に管理できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon EC2 Auto Scaling - アプリケーションの負荷を処理する正しい数の Amazon Elastic Compute Cloud (Amazon EC2) インスタンスを確保できます。	Auto Scaling が Auto Scaling グループの Amazon EC2 インスタンスを起動または終了するときに、通知を受信します。詳細については、『Amazon EC2 Auto Scaling ユーザーガイド』の「 Auto Scaling グループのスケーリング時の Amazon SNS 通知の取得 」を参照してください。

AWS のサービス	Amazon SNS を使用する利点
EC2 Image Builder - カスタマイズされたセキュアな最新のサーバーイメージの作成、管理、デプロイを自動化するのに役立ちます。最新のサーバーイメージには、特定の IT 標準を満たすソフトウェアおよび設定が事前にインストールおよび定義されています。	構築が完了すると通知を受信します。詳細については、『AWS コンピューティングのブログ』の「EC2 Image Builder パイプラインでの最新のサーバーイメージの追跡」を参照してください。
AWS Elastic Beanstalk - アプリケーションのキャパシティープロビジョニング、負荷分散、スケーリング、およびアプリケーションの状態のモニタリングといった詳細を処理します。	アプリケーションに影響を与える重要なイベントの通知を受信します。詳細については、『AWS Elastic Beanstalk デベロッパーガイド』の「Amazon SNS を使用した Elastic Beanstalk 環境通知」を参照してください。
AWS Lambda - サーバーをプロビジョニングまたは管理しなくてもコードを実行できます。	SNS トピックを Lambda デッドレターキューまたは Lambda の送信先として設定することで、関数出力データを受信します。詳細については、『AWS Lambda デベロッパーガイド』の「非同期の呼び出し」を参照してください。
Amazon Lightsail - 開発者が AWS を使用してウェブサイトやウェブアプリケーションを構築するのに役立ちます。	インスタンス、データベース、ロードバランサーの 1 つのメトリクスが指定したしきい値を超えた場合に通知を受信します。詳細については、「Amazon Lightsail デベロッパーガイド」の「Amazon Lightsail での通知連絡先の追加」を参照してください。

コンテナサービス

次の表は Amazon SNS と Amazon EKS Distro や Amazon ECS などの AWS コンテナサービスとの統合方法を示しています。これにより、Amazon EKS クラスターの更新とセキュリティパッチを追跡し、新しい ECS 最適化 AMI リリースの通知を受け取ることができます。

これらの統合を活用すると、重要な更新や変更について常に情報を取得して、コンテナデプロイのセキュリティと効率を維持できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon EKS Distro - アプリケーションがデプロイされている場所に関係なく、信頼性の高い安全なクラスターを作成できます。	Amazon EKS Distro で作成されたクラスターの更新とセキュリティパッチを追跡します。詳細については、「Amazon EKS Distro - Amazon EKS で使用されるオープンソースの Kubernetes ディストリビューション」を参照してください。
Amazon Elastic Container Service (Amazon ECS) - クラスターでコンテナを実行、停止、管理できます。	新しい Amazon ECS 最適化 AMI が利用可能になったときに通知を受信します。詳細については、『Amazon Elastic Container Service デベロッパーガイド』の「Amazon ECS に最適化された AMI 更新通知にサブスクライブする」を参照してください。

カスタマーエンゲージメントサービス

次の表は、Amazon Connect、Amazon Simple Email Service (SES) と統合することで Amazon SNS がカスタマーエンゲージメントサービスを強化する方法を示しています。これにより、アラートと検証の受信、双方向 SMS メッセージングの設定、バウンス、苦情、配信に関する E メール通知のモニタリングが可能になります。Amazon Connect AWS End User Messaging SMS

これらの統合は、複数チャネルにわたるカスタマーコミュニケーションの管理に役立ちます。

AWS のサービス	Amazon SNS を使用する利点
Amazon Connect - オムニチャネルクラウドコンタクトセンターをセットアップして、顧客とやり取りすることができます。	アラートと検証を受信します。詳細については、Amazon Connect 管理者ガイドの「Amazon Connect AWS での能力」を参照してください。Amazon Connect
AWS End User Messaging SMS - E メール、SMS、音声メッセージ、プッシュ通知を送信することで、顧客のエンゲージメントを支援します。	双方向 SMS を構成しています。これにより、顧客からメッセージを受信することができます。詳細については、「AWS End User Messaging SMS ユーザーガイド」の「双方向 SMS メッセージング」を参照してください。

AWS のサービス	Amazon SNS を使用する利点
Amazon Simple Email Service (Amazon SES) - ユーザー自身の E メールアドレスとドメインを使用して E メールを送受信するための、費用対効果の高い方法を提供します。	バウンス、苦情および配信に関する通知を受信します。詳細については、『Amazon Simple Email Service デベロッパーガイド』の、「 Amazon SES のための Amazon SNS 通知を設定する 」を参照してください。

データベースサービス

次の表は Amazon SNS が AWS Database Migration Service (DMS)、Amazon DynamoDB、Amazon ElastiCache、Amazon Neptune、Amazon Redshift、Amazon Relational Database Service (RDS) などの AWS データベースサービスと統合して、データ移行、メンテナンスアクティビティ、キャッシュ更新、データベース変更などの重要なイベントに関する通知を送信する方法を示しています。

これらの統合により、主要な運用イベントに関するタイムリーなアラートを提供でき、データベース環境をより効果的にモニタリングおよび管理できます。

AWS のサービス	Amazon SNS を使用する利点
AWS Database Migration Service - オンプレミスのデータベースから AWS クラウドへデータを移行します。	レプリケーションインスタンスの作成または削除など、AWS DMS イベントが発生したときに通知を受け取ります。詳細については、『AWS Database Migration Service ユーザーガイド』の「 AWS Database Migration Service のイベントと通知を使用する 」を参照してください。
Amazon DynamoDB - シームレスな拡張性により、フルマネージドの NoSQL データベースサービスの高速で予測可能なパフォーマンスを提供します。	メンテナンスイベントが発生したときに通知を受信します。詳細については、『Amazon DynamoDB デベロッパーガイド』の「 DAX クラスターの設定をカスタマイズする 」を参照してください。
Amazon ElastiCache - 高パフォーマンスで費用対効果に優れ、さらにサイズ変更も可能なメモリ内キャッシュを提供しながら、分散キャッ	重要なイベントが発生したときに通知を受信します。詳細については、「Amazon ElastiCache (Memcached) ユーザーガイド」の「 イベ

AWS のサービス	Amazon SNS を使用する利点
シユ環境のデプロイと管理に伴う複雑性を排除します。	ント通知と Amazon SNS 」を参照してください。
Amazon Neptune - 高度に接続されたデータセットを扱うアプリケーションを構築し実行できます。	Neptune イベントが発生したときに通知を受信します。詳細については、『Neptune ユーザーガイド』の「 Neptune イベント通知を使用する 」を参照してください。
Amazon Redshift - データウェアハウスを設定、操作、およびスケーリングするためのすべての作業を管理します。	Amazon Redshift イベントの通知を受信します。詳細については、「Amazon Redshift 管理ガイド」の「 Amazon Redshift イベント通知 」を参照してください。
Amazon Relational Database Service - AWS クラウドでリレーショナルデータベースを簡単にセットアップ、運用、スケーリングできます。	Amazon RDS イベントの通知を受信します。詳細については、『Amazon RDS ユーザーガイド』の「 Amazon RDS イベント通知を使用する 」を参照してください。

デベロッパー用ツールサービス

次の表は AWS CodeBuild、Amazon SNS が Amazon CodeGuru などの AWS 開発者ツールサービスと統合 AWS CodePipeline して、ビルドステータスの変更、リポジトリの更新 AWS CodeCommit AWS CodeDeploy、デプロイの進行状況、パフォーマンスの異常、パイプラインアクションなどの重要なイベントの通知を提供する方法を示しています。

これらの統合により、重要なイベントに関するアラートをタイムリーに受信して、ソフトウェア開発ワークフローを効率的にモニタリングおよび管理できます。

AWS のサービス	Amazon SNS を使用する利点
AWS CodeBuild - ソースコードをコンパイルし、単体テストを実行して、すぐにデプロイできるアーティファクトを生成します。	構築が成功したとき、失敗したとき、または構築フェーズから別の構築フェーズに移動したときに通知を受信します。詳細については、『AWS CodeBuild ユーザーガイド』

AWS のサービス	Amazon SNS を使用する利点
	の「CodeBuildのための通知サンプルを構築する」を参照してください。
AWS CodeCommit - クラウドでアセットを個人的に保存および管理するためのバージョン管理を提供します。	CodeCommit リポジトリのイベントに関する通知を受信します。詳細については、AWS CodeCommit ユーザーガイドの「例: Amazon SNS トピックの AWS CodeCommit トリガーを作成する」を参照してください。
AWS CodeDeploy - Amazon EC2 インスタンス、オンプレミスインスタンス、サーバーレス Lambda 関数、または Amazon ECS サービスに対するアプリケーションのデプロイを自動化します。	CodeDeploy デプロイまたはインスタンスイベントの通知を受信します。詳細については、『AWS CodeDeploy ユーザーガイド』の「CodeDeploy イベントのためのトリガーを作成する」を参照してください。
Amazon CodeGuru - ライブアプリケーションからランタイムパフォーマンスデータを収集し、アプリケーションのパフォーマンスを微調整するのに役立つレコメンデーションを提供します。	異常が発生したときに通知を受信します。詳細については、『Amazon CodeGuru ユーザーガイド』の「異常レポートとレコメンデーションレポートを使用する」を参照してください。
AWS CodePipeline - ソフトウェアの変更を継続的にリリースするために必要なステップを自動化します。	承認アクションに関する通知を受信します。詳細については、『AWS CodePipeline ユーザーガイド』の「CodePipeline での承認アクションの管理」を参照してください。

フロントエンドウェブおよびモバイルサービス

次の表は、Amazon SNS が と統合 AWS End User Messaging SMS して E メール、SMS、音声メッセージ、プッシュ通知を送信することでカスタマーエンゲージメントを強化する方法を示しています。これには、カスタマーメッセージを受信するために双方向 SMS を設定する機能も含まれます。

この統合により、さまざまな通信チャネルにわたって顧客とより効果的にやり取りできます。

AWS のサービス	Amazon SNS を使用する利点
AWS End User Messaging SMS - E メール、SMS、音声メッセージ、プッシュ通知を送信することで、顧客のエンゲージメントを支援します。	双方向 SMS を構成しています。これにより、顧客からメッセージを受信することができます。詳細については、「AWS End User Messaging SMS ユーザーガイド」の「 双方向 SMS メッセージング 」を参照してください。

ゲーム開発サービス

次の表は、Amazon SNS と Amazon GameLift Servers との統合により、セッションベースのマルチプレイヤーゲームサーバーでマッチメイキングイベントとキューイベントに関する通知を提供する方法を示しています。

この統合により、ゲーム開発者は、ゲームサーバーのデプロイ、運用、スケーリングを自動化およびモニタリングして、シームレスなゲームエクスペリエンスを確保できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon GameLift Servers – ゲームサーバーのデプロイ、操作、スケーリングに必要な完全マネージド型サービスなど、クラウドでセッションベースのマルチプレイヤーゲームサーバーをホストするためのソリューションを提供します。	マッチメイキングとキューイベント通知を受信します。詳細については、次のページを参照してください。 • マッチメイキング通知については、「Amazon GameLift Servers FlexMatch デベロッパーガイド」の「FlexMatch イベント通知の設定」を参照してください。 • キュー通知については、「Amazon GameLift Servers デベロッパーガイド」の「ゲームセッションの配置のイベント通知を設定する」を参照してください。

IoT サービス

次の表は、Amazon SNS が、AWS IoT Core AWS IoT Device Defender、AWS IoT Eventsなどの AWS IoT サービスと統合して AWS IoT Greengrass IoT イベントやアラートの通知を提供する方法を示しています。

これらの統合により、デバイスの動作を効果的にモニタリングしたり、異常なアクティビティのアラートを受信したり、リアルタイムの更新とアクションを使用して IoT デバイスを管理したりできます。

AWS のサービス	Amazon SNS を使用する利点
AWS IoT Core – IoT デバイスを他のデバイスやサービスに接続するクラウドサービスを提供します AWS クラウド。	AWS IoT Core イベントの通知を受信します。詳細については、『AWS IoT Amazon SNS デベロッパーガイド』の「 トピックの作成 」を参照してください。
AWS IoT Device Defender - デバイスの設定の監査、異常動作の検出を目的としたコネクテッドデバイスのモニタリング、セキュリティリスクの軽減ができます。	デバイスが動作に違反したときにアラームを受信します。詳細については、「AWS IoT デベロッパーガイド」の AWS IoT Device Defender 「検出の使用法」 を参照してください。
AWS IoT Events - 機器やデバイスフリートの障害や動作の変化をモニタリングし、そのようなイベントが発生したときにアクションをトリガーできます。	AWS IoT Events イベントの通知を受信します。詳細については、『AWS IoT Events デベロッパーガイド』の「 Amazon Simple Notification Service 」を参照してください。
AWS IoT Greengrass – AWS 物理デバイスに拡張して、生成したデータに対してローカルで動作できるようにしながら、管理、分析、耐久性のあるストレージにクラウドを使用します。	AWS IoT Greengrass イベントの通知を受信します。詳細については、『AWS IoT Greengrass Version 1 デベロッパーガイド』の「 SNS コネクタ 」を参照してください。

機械学習サービス

次の表は、Amazon SNS が Amazon CodeGuru、Amazon DevOps Guru、Amazon Lookout for Metrics、Amazon Rekognition、Amazon SageMaker AI などの AWS 機械学習サービスと統合して、異常、運用上のインサイト、データラベリングアクティビティの通知を提供する方法を示しています。

これらの統合によって、アプリケーションのパフォーマンスをモニタリングしたり、データ異常に関するアラートを受信したり、リアルタイム更新を使用して機械学習モデルのデプロイを効率化したりできます。

AWS のサービス	Amazon SNS を使用する利点
Amazon CodeGuru - ライブアプリケーションからランタイムパフォーマンスデータを収集し、アプリケーションのパフォーマンスを微調整するのに役立つレコメンデーションを提供します。	異常が発生したときに通知を受信します。詳細については、『Amazon CodeGuru ユーザーガイド』の「 異常レポートとレコメンデーションレポートを使用する 」を参照してください。
Amazon DevOps Guru - 機械学習を使用して運用上のインサイトを生成し、運用アプリケーションのパフォーマンスを向上させるのに役立ちます。	インサイトと確認を転送します。詳細については、「AWS 管理およびガバナンスのブログ」の「 Amazon DevOps Guru で PagerDuty を使用して、機械学習を活用した運用上のインサイトをオンコールチームに提供する 」を参照してください。
Amazon Lookout for Metrics - データ内の異常を検出し、その根本原因を特定し、迅速な対応を可能にします。	異常の通知を受信します。詳細については、『 Lookout for Metrics で Amazon SNS を使用する 』の「Amazon Lookout for Metrics デベロッパーガイド」を参照してください。
Amazon Rekognition - イメージ分析とビデオ分析をアプリケーションに追加できます。	リクエスト結果の通知を受信します。詳細については、『Amazon Rekognition デベロッパーガイド』の「 リファレンス: ビデオ分析結果の通知 」を参照してください。
Amazon SageMaker AI - データサイエンティストや開発者は、機械学習モデルを構築してトレーニングし、それらのモデルを本番稼働準備が整ったホスト環境に直接デプロイできます。	データオブジェクトがラベル付けされると、通知を受信します。詳細については、「Amazon SageMaker AI デベロッパーガイド」の「 ストリーミングラベリングジョブを作成する 」を参照してください。

管理とガバナンスサービス

次の表はAmazon SNS がチャットアプリケーション、CloudTrail CloudFormation、CloudWatch AWS Config、AWS Control Tower AWS License Manager、などの Amazon Q Developer などの AWS 管理およびガバナンスサービスと統合し AWS Systems Manager、インフラストラクチャの変更 AWS Service Catalog、コンプライアンスアラート、運用上のインサイトなどの主要なイベントの通知を提供する方法を示しています。

これらの統合により、関連するチームやシステムにタイムリーなアラートや更新を提供して、AWS 環境を効率的にモニタリングおよび管理できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon Q Developer in chat applications – DevOps やソフトウェア開発チームが Amazon Chime や Slack のチャットルームを使用して、AWS クラウド内の運用イベントをモニタリングして対応できるようにします。	チャットルームに通知を配信します。詳細については、「Amazon Q Developer in chat applications 管理者ガイド」の「 Amazon Q Developer in chat applications の設定 」を参照してください。
AWS CloudFormation – AWS インフラストラクチャのデプロイを予測どおりに繰り返し作成およびプロビジョニングできます。	スタックが作成され、更新されたときに通知を受信します。詳細については、『AWS CloudFormation ユーザーガイド』の「 CloudFormation スタックオプションの設定をする 」を参照してください。
AWS CloudTrail - お客様の AWS アカウント アクティビティイベント履歴を提供します。	CloudTrail が新しいログファイルを Amazon S3 バケットに発行するときに通知を受信します。詳細については、『ユーザーガイド』の「 AWS CloudTrail CloudTrail 用の Amazon SNS 通知の構成 」を参照してください。
Amazon CloudWatch – AWS リソースとで実行しているアプリケーションを AWS リアルタイムでモニタリングします。	アラームの変化状態に応じて通知を受信します。詳細については、『Amazon CloudWatch ユーザーガイド』の「 Amazon CloudWatch アラームの使用 」を参照してください。
AWS Config – 内のリソースの設定 AWS の詳細ビューを提供します AWS アカウント。	リソースが更新されたとき、または AWS Config がリソースに対しカスタムルールまた

AWS のサービス	Amazon SNS を使用する利点
	はマネージドルールを評価したときに通知を受信します。詳細については、「デベロッパーガイド」の「が SNS トピック AWS Config に送信する通知」および「設定項目の変更通知の例」を参照してください。AWS Config
AWS Control Tower – 安全で準拠したマルチアカウント AWS 環境を設定および管理できます。	アラートを使用すると、ランディングゾーン内のドリフトを防止し、コンプライアンス通知を受信できます。詳細については、『AWS Control Tower ユーザーガイド』の「Amazon Simple Notification Service でアラートを追跡する」を参照してください。
AWS License Manager – ソフトウェアベンダーからのソフトウェアライセンスを AWS とオンプレミス環境全体で一元管理できます。	License Manager の通知とアラートを受信します。詳細については、「License Manager ユーザーガイド」の「License Manager の設定」および AWS 「管理とガバナンスブログ」の「通知用の AWS License Manager ServiceNow インシデントの作成」を参照してください。
AWS Service Catalog - IT 管理者が承認された製品のポートフォリオを作成および管理し、そのポートフォリオをエンドユーザーに配布できます。エンドユーザーは、パーソナライズされたポータルから必要な製品にアクセスできます。	スタックイベントに関する通知を受信します。詳細については、「Service Catalog 管理者ガイド」の「AWS Service Catalog 通知の制約」を参照してください。
AWS Systems Manager – インフラストラクチャを表示および制御できます AWS。	コマンドのステータスに関する通知を受け取ります。詳細については、『AWS Systems Manager ユーザーガイド』の「Amazon SNS 通知を使用した Systems Manager のステータス変更のモニタリング」を参照してください。

メディアサービス

次の表は、Amazon SNS と、Amazon Elastic Transcoder との統合により、メディアトランスコーディングジョブのステータスが変化したときに通知を送信する方法について説明しています。これにより、Amazon S3 に保存されているメディアファイルの、コンシューマー再生デバイスに適した形式への変換を、効率的にモニタリングおよび管理できます。

この統合により、ジョブステータスに関するリアルタイムのアラートを提供して、メディア処理ワークフローを効率化できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon Elastic Transcoder - Amazon S3 に保存したメディアファイルをユーザーの再生デバイスに対応した形式のメディアファイルに変換できます。	ジョブのステータスが変更されたときに通知を受信します。詳細については、『Amazon Elastic Transcoder デベロッパーガイド』の「 ジョブステータスの通知 」を参照してください。

移行と転送のサービス

次の表は、Amazon SNS を AWS Database Migration Service (DMS) などの AWS 移行および転送サービスと統合して AWS Snowball Edge、サーバーデータ収集 AWS Application Discovery Service、データベース移行アクティビティ、データ転送ジョブなどのイベントの通知を提供する方法を示しています。

これらの統合により、重要な移行タスクに関するリアルタイムのアラートと更新を提供して、クラウド移行プロセスを効果的に管理およびモニタリングできます。

AWS のサービス	Amazon SNS を使用する利点
AWS Application Discovery Service – オンプレミスサーバーに関する使用状況と設定データを収集 AWS クラウド することで、への移行を計画するのに役立ちます。	を介してイベントの通知を受信します AWS CloudTrail。詳細については、『Application Discovery Service ユーザーガイド』の「 AWS CloudTrailを使用したApplication Discovery Service API コールのログ記録 」を参照してください。

AWS のサービス	Amazon SNS を使用する利点
AWS Database Migration Service - オンプレミスのデータベースから AWS クラウドへデータを移行します。	レプリケーションインスタンスの作成または削除など、AWS DMS イベントが発生したときに通知を受け取ります。詳細については、『AWS Database Migration Service ユーザーガイド』の「 AWS Database Migration Service のイベントと通知を使用する 」を参照してください。
AWS Snowball Edge - 物理ストレージデバイスを使用して、Amazon S3 とオンサイトのデータストレージロケーション間で、インターネットよりも高速に大量のデータを転送します。	Snowball Edge ジョブの通知を受信します。詳細については、「AWS Snowball Edge ユーザーガイド」の「 Snow Family デバイスの通知 」を参照してください。

ネットワーキングとコンテンツ配信サービス

次の表は、Amazon SNS が Amazon API Gateway、Amazon CloudFront、Elastic Load Balancing Direct Connect、Amazon Route 53、Amazon VPC などの AWS ネットワークおよびコンテンツ配信サービスと統合して、API メッセージ、CloudFront メトリクスアラーム、接続状態の変更、ロードバランサーイベント、ヘルスチェックステータス、VPC エンドポイントアクティビティなどのイベントの通知を送信する方法を示しています。

これらの統合により、タイムリーなアラートと更新を提供して、ネットワークとコンテンツの配信オペレーションをモニタリングおよび管理できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon API Gateway - 規模を問わず、独自の REST および WebSocket API を作成してデプロイできます。	API Gateway エンドポイントに投稿されたメッセージを受信します。詳細については、「 API Gateway デベロッパーガイド 」の「 チュートリアル: AWS 統合を使用して API Gateway REST API を構築する 」を参照してください。
Amazon CloudFront - 静的および動的なウェブコンテンツ (.html、.css、.php、イメージ、メディアファイルなど) を高速に配信できます。	指定した CloudFront メトリクスに基づくアラームが発生すると、通知を受信します。詳細については、『Amazon CloudFront デベロッ

AWS のサービス	Amazon SNS を使用する利点
Direct Connect – 内部ネットワークを、標準のイーサネット光ファイバケーブルを介して Direct Connect ロケーションにリンクします。	パーガイド』の「通知受信するようにアラームを設定する」を参照してください。 Direct Connect 接続変更状態をモニタリングするアラームが発生すると通知を受信します。詳細については、Direct Connect 「ユーザーガイド」のDirect Connect 「接続をモニタリングする CloudWatch アラームの作成」を参照してください。
Elastic Load Balancing - 受信したトラフィックを複数のアベイラビリティゾーンの複数のターゲット (Amazon EC2 インスタンス、コンテナ、IP アドレスなど) に自動的に分散させます。	ロードバランサーイベント用に作成したアラームの通知を受信します。詳細については、『Classic Load Balancer 向けユーザーガイド』の「ロードバランサー向け CloudWatch アラームを作成する」を参照してください。
Amazon Route 53 - ドメイン登録、DNS ルーティング、ヘルスチェックを提供します。	ヘルスチェックのステータスに不具合がある場合に通知を受信します。詳細については、『Amazon Route 53 デベロッパーガイド』の「ヘルスチェックのステータス (コンソール) に不具合がある場合に Amazon SNS 通知を受信するには」を参照してください。
Amazon Virtual Private Cloud (Amazon VPC) – 定義した仮想ネットワークに AWS リソースを起動できます。	インターフェイスエンドポイントで発生する特定のイベントに関する通知を受信します。詳細については、『Amazon VPC User Guide』の「エンドポイントサービスの通知を作成および管理する」を参照してください。

セキュリティ、アイデンティティ、コンプライアンスサービス

次の表は、Amazon SNS が Amazon GuardDuty、Amazon Inspector Directory Service、などの AWS セキュリティ、アイデンティティ、コンプライアンスサービスと統合して、ディレクトリステータスの変更 AWS Security Hub CSPM、セキュリティ検出結果、Inspector イベント、セキュリティハブの発表に関する通知を提供する方法を示しています。

これらの統合により、セキュリティとコンプライアンスイベントに関するタイムリーなアラートと更新を提供して、堅牢なセキュリティプラクティスを維持できます。

AWS のサービス	Amazon SNS を使用する利点
AWS Directory Service – 他のもので Microsoft Active Directory (AD) を使用する複数の方法を提供します AWS のサービス。	ディレクトリのステータスが変更されたときにメールまたはテキストメッセージ (SMS) を受け取ります。詳細については、『AWS Directory Service 管理ガイド』の「ディレクトリステータス通知の構成」を参照してください。
Amazon GuardDuty – AWS 環境内の予期しないアクティビティや潜在的に不正なアクティビティ、悪意のあるアクティビティを特定するのに役立つ継続的なセキュリティモニタリングを提供します。	新しくリリースされた結果タイプ、既存の結果タイプに関する更新などの機能の変更に係る通知を受信します。詳細については、『Amazon GuardDuty ユーザーガイド』の「GuardDuty お知らせの SNS トピックをサブスクライブする」を参照してください。
Amazon Inspector - Amazon EC2 インスタンスのネットワークアクセシビリティとそれらのインスタンスで実行されるアプリケーションのセキュリティ状態をテストします。	Amazon Inspector イベントの通知を受信します。詳細については、『Amazon Inspector ユーザーガイド』の「Amazon Inspector 通知用の SNS トピックのセットアップ」を参照してください。
AWS Security Hub CSPM — AWS のセキュリティチェックを自動化し、セキュリティアラートを一元化します。	追加、編集、または廃止された AWS Security Hub CSPM コントロールまたは標準に関する通知を含む、AWS Security Hub CSPM 発表に関する通知を受け取ります。詳細については、Amazon SNSでの AWS Security Hub CSPM お知らせのサブスクライブ」を参照してください。

サーバーレスサービス

次の表は、Amazon SNS と、Amazon DynamoDB、Amazon EventBridge、Lambda などのサービスとの統合により、メンテナンス更新、リアルタイムデータストリーム、Lambda 関数出力などの重要なイベントの通知を送信する方法を示しています。

これらの統合により、重要なオペレーションに関するアラートをタイムリーに受信し、これらのイベントへのレスポンスを自動化して、アプリケーションを効率的にモニタリングおよび管理できます。

AWS のサービス	Amazon SNS を使用する利点
Amazon DynamoDB - フルマネージドの NoSQL データベースサービスでシームレスなスケーラビリティの高速で予測可能なパフォーマンスを提供します。	メンテナンスイベントが発生したときに通知を受信します。詳細については、『Amazon DynamoDB デベロッパーガイド』の「 DAX クラスターの設定をカスタマイズする 」を参照してください。
Amazon EventBridge – 独自のアプリケーション、software-as-a-service (SaaS) アプリケーションからリアルタイムデータのストリームを配信 AWS のサービスし、そのデータを Amazon SNS などのターゲットにルーティングします。EventBridge は、以前は CloudWatch Events と呼ばれていました。	EventBridge イベントの通知を受信します。詳細については、『Amazon EventBridge ユーザーガイド』の「 Amazon EventBridge ターゲット 」を参照してください。
AWS Lambda - サーバーをプロビジョニングまたは管理しなくてもコードを実行できます。	SNS トピックを Lambda デッドレターキューまたは Lambda の送信先として設定することで、関数出力データを受信します。詳細については、『AWS Lambda デベロッパーガイド』の「 非同期呼び出し 」を参照してください。

ストレージサービス

次の表は、Amazon SNS を Amazon Elastic File System (EFS) AWS Backup、Amazon Glacier、Amazon S3、などの AWS ストレージサービスと統合 AWS Snowball Edge して、バックアップアクティビティ、ファイルシステムアラーム、データ取り出しジョブ、バケットの変更、データ転送オペレーションなどのさまざまなイベントの通知を提供する方法を示しています。

これらの統合により、重要なストレージイベントに関するアラートをタイムリーに受信して、ストレージソリューションを効率的にモニタリングおよび管理できます。

AWS のサービス	Amazon SNS を使用する利点
AWS Backup - クラウド内およびオンプレミスの AWS のサービス にわたってデータのバックアップを一元化および自動化するのに役立ちます。	AWS Backup イベントの通知を受信します。詳細については、「AWS Backup デベロッパーガイドAmazon SNS を使用して AWS Backup イベントを追跡する」を参照してください。
Amazon Elastic File System - Amazon EC2 インスタンスのファイルストレージを提供します。	Amazon EFS イベント用に作成したアラームの通知を受信します。詳細については、『Amazon Elastic File System User Guide』の「自動モニタリングツール」を参照してください。
Amazon Glacier - 使用頻度の低いデータ用のストレージを提供します。	ポールの通知設定で、ジョブが完了するとメッセージが SNS トピックに送信されるように設定します。詳細については、「Amazon Glacier デベロッパーガイド」の「Amazon Glacier でのポール通知の設定」を参照してください。
Amazon Simple Storage Service (Amazon S3) - オブジェクトストレージを提供します。	オブジェクトがレプリケート先リージョンにレプリケートされない場合、Amazon S3 バケットまたはまれなインスタンスに変更が生じると通知を受信します。詳細については、『Amazon Simple Storage Service ユーザーガイド』の「チュートリアル: 通知 (SNS トピックまたは SQS キュー) 用のバケットを設定する」および「レプリケーションメトリクスと Amazon S3 イベント通知による進捗状況のモニタリング」を参照してください。
AWS Snowball Edge - 物理ストレージデバイスを使用して、Amazon S3 とオンサイトのデータストレージロケーション間で、インターネットよりも高速に大量のデータを転送します。	Snowball Edge ジョブの通知を受信します。詳細については、「AWS Snowball Edge ユーザーガイド」の「Snow Family デバイスの通知」を参照してください。

追加のイベントソース

次の表は、Amazon SNS を使用して AWS、毎日の機能の更新と AWS IP アドレス範囲の変更に関する通知をタイムリーに受信し、最新の AWS サービスリリース、インスタンスタイプ、VPC エンドポイント、パブリック IP アドレスの変更について通知を受ける方法を示しています。

これらの統合により、AWS インフラストラクチャの変更 up-to-date に保つことができ、リソースを効果的に管理できます。

ソース	Amazon SNS を使用する利点
AWS 毎日の機能の更新	Amazon SNS トピックを通じて、AWS のリリースや更新に関する詳細情報をタイムリーに受け取ることができます。これらのリリースには AWS リージョン、AWS のサービス、AWS Service Quotas と統合された Amazon VPC エンドポイント AWS のサービス、Amazon EC2 インスタンスタイプ、Amazon SageMaker AI インスタンスタイプ、Amazon Nimble Studio インスタンスタイプ、Amazon RDS データベースエンジンバージョン、Amazon MSK Apache Kafka バージョンが含まれます。詳細については、ニュースブログの Amazon SNS 経由で AWS 毎日の機能更新をサブスクライブ する」を参照してください。AWS
AWS IP アドレス範囲	Amazon SNS トピックを介して AWS IP 範囲の変更の通知を受信します。詳細については、の AWS 「IP アドレス範囲通知」 Amazon Web Services 全般のリファレンス」および ニュースブログの Amazon SNS 経由で AWS パブリック IP アドレスの変更をサブスクライブ する」を参照してください。AWS

イベント駆動型のコンピューティングの詳細については、次のソースを参照してください。

- [イベント駆動型のアーキテクチャとは](#)
- [コンピューティングブログのAmazon SNS と AWS コンピューティング、ストレージ、データベース、ネットワーキングサービスによるイベント駆動型コンピューティングAWS](#)」
- [AWS コンピューティングブログの「Event Fork Pipelines による AWS イベント駆動型アーキテクチャの強化」](#)

Amazon SNS イベントの送信先

このトピックには、[Application-to-Application \(A2A\) メッセージング](#)および [Application-to-Person \(A2P\) 通知](#)でグループ分けされた、すべてのイベントの送信先が一覧表示されています。

Note

Amazon SNS は 2020 年 10 月、[FIFO トピック](#)を導入しました。現在、ほとんどの AWS サービスは、SNS 標準トピックからのイベントの受信のみをサポートしています。Amazon SQS は、SNS 標準トピックと FIFO トピックの両方からのイベントの受信をサポートしています。

A2A 送信先

次の表は、Amazon SNS が、Lambda、Amazon SQS、AWS Event Fork Pipelines、HTTP/S エンドポイントなど、さまざまな Application-to-Application (A2A) の送信先にイベントを配信する方法を示しています。

これらの統合により、データのアーカイブと分析、カスタムビジネスロジックのトリガー、アプリケーション統合の促進、外部ウェブフックへのイベントのルーティングが可能になり、イベント駆動型アーキテクチャの効率性と柔軟性が向上します。

イベントの発行先	Amazon SNS を使用する利点
https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html	アーカイブおよび分析の目的で、イベントを配信ストリームに配信します。配信ストリームを通じて、イベントを Amazon Simple Storage Service (Amazon S3)、Amazon Redshift、Amazon OpenSearch Service (OpenSearch Service)、または Datadog、New Relic、Mon

イベントの発行先	Amazon SNS を使用する利点
	goDB、Splunk などのサードパーティーの AWS 送信先に配信できます。詳細については、「 Firehose 配信ストリームへのファンアウト 」を参照してください。
AWS Lambda	カスタムビジネスロジックの実行をトリガーする関数にイベントを配信します。詳細については、「 Amazon SNS 通知の Lambda 関数へのファンアウトと自動処理 」を参照してください。
Amazon Simple Queue Service	アプリケーション統合の目的でイベントをキューに配信します。詳細については、「 Amazon SNS 通知の Amazon SQS キューへのファンアウトと非同期処理 」を参照してください。
AWS Event Fork Pipelines	イベントのバックアップとストレージ、イベントの検索と分析、またはイベント再生パイプラインにイベントを配信します。詳細については、「 Amazon SNS イベントを AWS Event Fork Pipelines にファンアウトする 」を参照してください。
HTTP/S	外部 webhook にイベントを配信します。詳細については、「 Amazon SNS 通知の HTTPS エンドポイントへのファンアウト 」を参照してください。

A2Pの送信先

次の表は、Amazon SNS が、携帯電話 (SMS およびネイティブプッシュ通知)、E メール受信箱、Amazon Chime チャットルーム、Slack チャンネル、オンコールチーム (PagerDuty を介した運用上のインサイト) など、さまざまな送信先に Application-to-Person (A2P) 通知を配信する方法を示しています。

これらの統合によって、複数のプラットフォームと通信チャネルにわたってリアルタイムのアラートと更新を行うことができ、通信と運用効率が向上します。

イベントの発行先	Amazon SNS を使用する利点
SMS	イベントをテキストメッセージとして携帯電話に配信します。詳細については、「 Amazon SNS を使用したモバイルテキストメッセージ 」を参照してください。
E メール	イベントを E メールメッセージとして受信トレイに配信します。詳細については、「 Amazon SNS E メールサブスクリプションの設定と管理 」を参照してください。
プラットフォームエンドポイント	ネイティブプッシュ通知として携帯電話にイベントを配信します。詳細については、「 Amazon SNS でモバイルプッシュ通知を送信する 」を参照してください。
Amazon Q Developer in chat applications	Amazon Chime チャットルームや Slack チャンネルにイベントを配信します。詳細については、「Amazon Q Developer in chat applications 管理者ガイド」の次のページを参照してください。 • Amazon Chime を使用した Amazon Q Developer in chat applications の設定 • Slack を使用した Amazon Q Developer in chat applications の設定 • Amazon Q Developer in chat applications を他の AWS サービスと併用する
PagerDuty	オンコールチームにオペレーション上のインサイトを配信します。詳細については、『AWS 管理とガバナンスのブログ』の「 Amazon

イベントの発行先	Amazon SNS を使用する利点
	DevOps Guru で PagerDuty を介して、ML を活用したオペレーション上のインサイトをオンコールチームに配信する』 を参照してください。

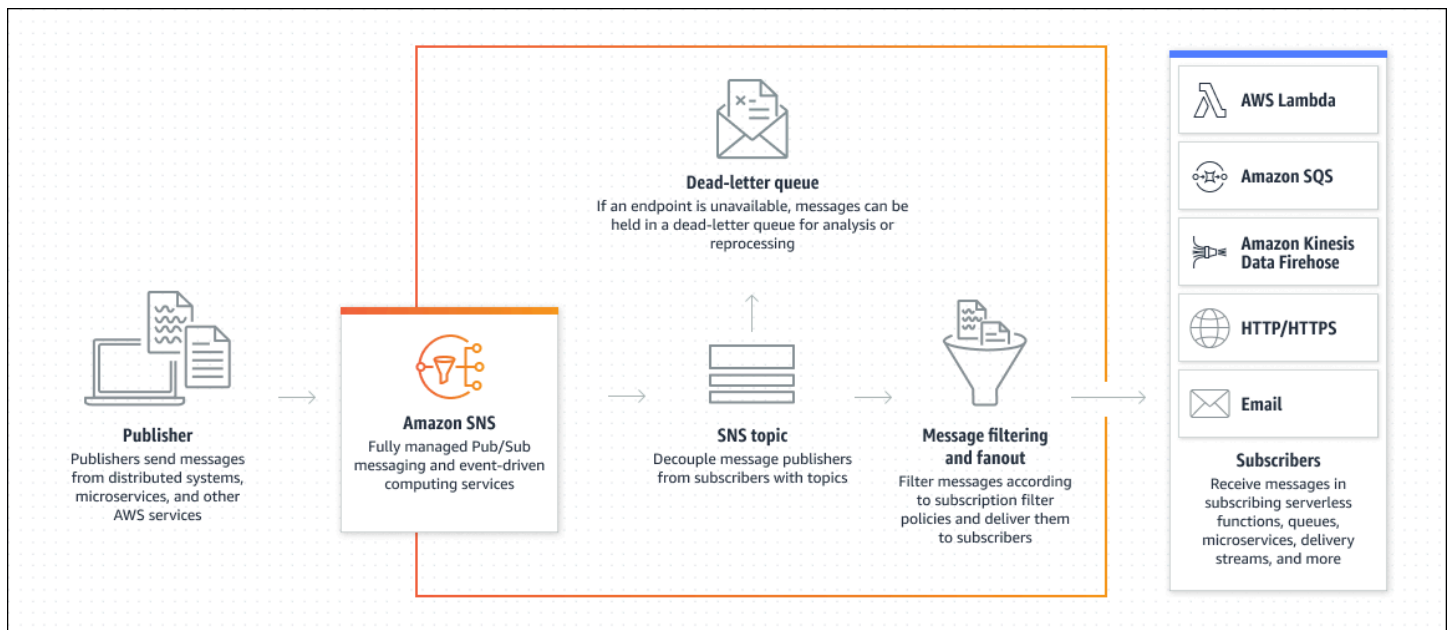
Note

ネイティブ AWS イベントとカスタムイベントの両方をチャットアプリケーションに配信することができます。

- ネイティブ AWS イベント – Amazon Q Developer in chat applications を使用してネイティブ AWS イベントを Amazon SNS トピック経由で Amazon Chime と Slack に送信できます。サポートされているネイティブ AWS イベントのセットには、AWS Billing and Cost Management、AWS Health、AWS CloudFormation、Amazon CloudWatch などからのイベントが含まれます。詳細については、「Amazon Q Developer in chat applications 管理者ガイド」の「[Amazon Q Developer in chat applications を他のサービスと併用する](#)」を参照してください。
- カスタムイベント – Amazon SNS トピックを通じて、Amazon Chime、Slack、Microsoft Teams にカスタムイベントを送信することもできます。これを行うには、カスタムイベントを SNS トピックに発行し、サブスクライブされた Lambda 関数にイベントを配信します。Lambda 関数は、チャットアプリケーションの webhook を使用して、受信者にイベントを配信します。詳細については、「[webhook を使用して Amazon SNS メッセージを Amazon Chime、Slack、または Microsoft Teams に送信するには](#)」を参照してください。

Amazon SNS を使用した Application-to-Application メッセージング

Amazon SNS は、サブスクライバーからパブリッシャーを分離して、Application-to-Application (A2A) メッセージングを簡素化し、マイクロサービス、分散システム、サーバーレスアプリケーションをサポートします。メッセージは Amazon SNS トピックに送信され、フィルタリングされて、Lambda、Amazon SQS、HTTP エンドポイントなどのサブスクライバーに配信されます。配信に失敗すると、メッセージはデッドレターキューに保存され、さらに分析や再処理が行われます。



Firehose 配信ストリームへのファンアウト

[配信ストリーム](#)を Amazon SNS トピックにサブスクライブできます。これにより、追加のストレージおよび分析エンドポイントに通知を送信できます。Amazon SNS トピックに発行されたメッセージは、サブスクライブした Firehose 配信ストリームに送信され、Firehose で設定した配信先に配信されます。サブスクリプション所有者は、Amazon SNS トピックに最大 5 つの Firehose 配信ストリームをサブスクライブできます。Firehose の各配信ストリームには、1 秒あたりのリクエスト数とスループットの[デフォルトクォータ](#)があります。この制限により、メッセージの配信数 (アウトバウンドトラフィック) よりもメッセージの発行数 (インバウンドトラフィック) が多くなる場合があります。インバウンドトラフィックがアウトバウンドトラフィックよりも多いと、サブスクリプションによって大きなメッセージバックログが蓄積され、メッセージ配信のレイテンシーが高くなる場合があります。ワークロードへの悪影響を避けるため、発行レートに基づいて[クォータの引き上げ](#)をリクエストできます。

Firehose 配信ストリームを通じて、Amazon SNS 通知を Amazon Simple Storage Service (Amazon S3)、Amazon Redshift、Amazon OpenSearch Service (OpenSearch Service)、および Datadog、New Relic、MongoDB、Splunk などのサードパーティーサービスプロバイダーにダウンロードできます。

例えば、この機能を使用して、コンプライアンス、アーカイブ、またはその他の目的で Amazon S3 バケットにトピックに送信されたメッセージを永続的に保存できます。これを行うには、Amazon S3 バケットを送信先とする Firehose 配信ストリームを作成し、その配信ストリームを Amazon SNS トピックにサブスクライブします。別の例として、Amazon SNS トピックに送信されたメッセージの分析を実行するには、OpenSearch Service インデックスを送信先とする配信ストリームを作成します。次に、Firehose 配信ストリームを Amazon SNS トピックにサブスクライブします。

Amazon SNS では、Firehose エンドポイントに送信される通知のメッセージ配信ステータスログもサポートします。詳細については、「[Amazon SNS メッセージ配信ステータス](#)」を参照してください。

Firehose 配信ストリームを Amazon SNS トピックにサブスクライブするための前提条件

配信ストリームを SNS トピックにサブスクライブするには、に次のもの AWS アカウント が必要です。

- スタンダード SNS トピック 詳細については、「[Amazon SNS トピックを作成する](#)」を参照してください。
- Firehose 配信ストリーム。詳細については、「Amazon Data Firehose デベロッパーガイド」の「[配信ストリームの作成](#)」および「[Firehose リソースへのアクセス権をアプリケーションに付与する](#)」を参照してください。
- Amazon SNS サービスプリンシパルを信頼し、配信ストリームに書き込むアクセス許可を持つ AWS Identity and Access Management (IAM) ロール。サブスクリプションを作成するときに、このロールの Amazon リソースネーム (ARN) を SubscriptionRoleARN として入力します。Amazon SNS はこのロールを引き受け、Amazon SNS が Firehose 配信ストリームにレコードを配置できるようにします。

次のポリシーの例は、推奨されるアクセス権限を示しています。

JSON

```
{  
  "Version": "2012-10-17",
```

```
"Statement": [
  {
    "Action": [
      "firehose:DescribeDeliveryStream",
      "firehose:ListDeliveryStreams",
      "firehose:ListTagsForDeliveryStream",
      "firehose:PutRecord",
      "firehose:PutRecordBatch"
    ],
    "Resource": [
      "arn:aws:firehose:us-east-1:111111111111:deliverystream/firehose-sns-delivery-stream"
    ],
    "Effect": "Allow"
  }
]
```

Firehose を使用するための完全なアクセス許可を付与するには、AWS 管理ポリシー を使用することもできますAmazonKinesisFirehoseFullAccess。または、Firehose を使用するためのより厳格なアクセス権限を付与するために、独自のポリシーを作成することもできます。少なくとも、ポリシーは、特定の配信ストリームで PutRecord オペレーションを実行するアクセス権限を与える必要があります。

いずれの場合も、信頼関係を編集して、Amazon SNS サービスプリンシパルを含める必要があります。例えば、次のようになります。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```


ロールの作成の詳細については、IAM ユーザーガイドの「[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。

これらの要件を満たしたら、[配信ストリームを SNS トピックへサブスクライブ](#)できます。

Amazon SNS トピックへ Firehose 配信ストリームをサブスクライブする

Amazon SNS 通知を[配信ストリーム](#)に配信するには、まず、すべての[前提条件](#)を満たしていることを確認してください。サポートされているエンドポイントのリストについては、「Amazon Web Services 全般のリファレンス」の「[エンドポイントとクォータ](#)」を参照してください。

トピックへ Firehose 配信ストリームをサブスクライブするには

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションペインで [サブスクリプション] を選択します。
3. [サブスクリプション] ページで [サブスクリプションの作成] を選択します。
4. [サブスクリプションの作成] ページで [詳細] セクションで、以下を実行します。
 - a. [トピック ARN] は、標準トピックの Amazon リソースネーム (ARN) を選択します。
 - b. [プロトコル] で、[Firehose] を選択します。
 - c. [エンドポイント] で、Amazon SNS からの通知を受信できる Firehose 配信ストリームの ARN を選択します。
 - d. [サブスクリプションロール ARN] で、Firehose 配信ストリームに書き込むために作成した AWS Identity and Access Management IAM ロールの ARN を指定します。詳細については、「[Firehose 配信ストリームを Amazon SNS トピックにサブスクライブするための前提条件](#)」を参照してください。
 - e. (オプション) 発行されたメッセージから Amazon SNS メタデータを削除するには、[Enable raw message delivery] を選択します。詳細については、「[Amazon SNS raw メッセージの配信](#)」を参照してください。
5. (オプション) フィルターポリシーを設定するには、[サブスクリプションのフィルターポリシー] セクションを展開します。詳細については、「[Amazon SNS サブスクリプションフィルターポリシー](#)」を参照してください。
6. (オプション) サブスクリプションのデッドレターキューを設定するには、Redrive ポリシー (デッドレターキュー)を展開します。詳細については、「[Amazon SNS デッドレターキュー](#)」を参照してください。

7. [サブスクリプションの作成] を選択します。

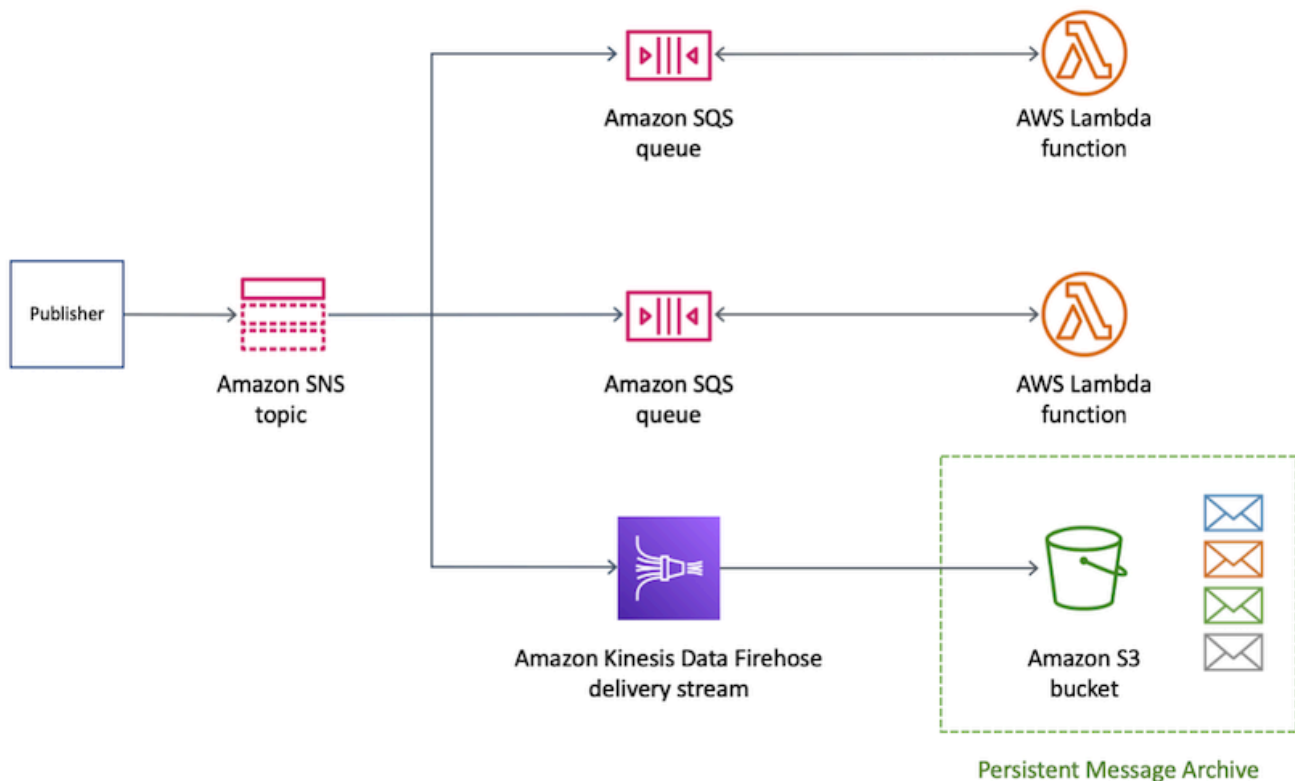
コンソールがサブスクリプションを作成し、サブスクリプションの [詳細] ページを開きます。

複数の配信ストリーム送信先にわたる Amazon SNS メッセージを管理する

[配信ストリーム](#)を使用すると、複数の送信先にまたがる Amazon SNS メッセージを管理できます。また、Amazon S3、Amazon OpenSearch Service、Amazon Redshift、HTTP エンドポイントと統合することで、ストレージ、インデックス作成、分析が可能になります。メッセージのフォーマットと配信を適切に設定することで、Amazon SNS 通知を Amazon S3 に保存して後で処理したり、Amazon Athena を使用して構造化されたメッセージデータを分析したりすることができます。また、OpenSearch でメッセージのインデックスを作成してリアルタイム検索や視覚化を行ったり、Amazon Redshift でアーカイブを構造化して高度なクエリを実行したりすることもできます。

Amazon S3 送信先に Amazon SNS メッセージを保存して分析する

このトピックでは、配信ストリームが Amazon Simple Storage Service (Amazon S3) にデータを発行する方法について説明します。



トピック

- [Amazon S3 送信先でのストレージの Amazon SNS 通知のフォーマット](#)
- [Athena を使用して Amazon S3 に保存された Amazon SNS メッセージを分析する](#)

Amazon S3 送信先でのストレージの Amazon SNS 通知のフォーマット

次の例は、読みやすくするためにインデントを使用して Amazon Simple Storage Service (Amazon S3) バケットに送信された Amazon SNS 通知を示しています。

Note

この例では、raw メッセージ配信は、発行されたメッセージに対して無効になっています。raw メッセージ配信が無効になっている場合、Amazon SNS は次のプロパティを含む JSON メタデータをメッセージに追加します。

- Type
- MessageId
- TopicArn
- Subject
- Timestamp
- UnsubscribeURL
- MessageAttributes

raw 配信の詳細については、「[Amazon SNS raw メッセージの配信](#)」を参照してください。

```
{
  "Type": "Notification",
  "MessageId": "719a6bbf-f51b-5320-920f-3385b5e9aa56",
  "TopicArn": "arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic",
  "Subject": "My 1st subject",
  "Message": "My 1st body",
  "Timestamp": "2020-11-26T23:48:02.032Z",
  "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:333333333333:my-kinesis-test-
topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5",
  "MessageAttributes": {
    "myKey1": {
```

```

        "Type": "String",
        "Value": "myValue1"
    },
    "myKey2": {
        "Type": "String",
        "Value": "myValue2"
    }
}
}

```

次の例は、配信ストリームを介して同じ Amazon S3 バケットに送信された 3 つの SNS メッセージを示しています。バッファリングが適用され、各メッセージは改行で区切られます。

```

{"Type":"Notification","MessageId":"d7d2513e-6126-5d77-bbe2-09042bd0a03a","TopicArn":"arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic","Subject":"My 1st subject","Message":"My 1st body","Timestamp":"2020-11-27T00:30:46.100Z","UnsubscribeURL":"https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:313276652360:my-kinesis-test-topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5","MessageAttributes":{"myKey1":{"Type":"String","Value":"myValue1"},"myKey2":{"Type":"String","Value":"myValue2"}}}
{"Type":"Notification","MessageId":"0c0696ab-7733-5bfb-b6db-ce913c294d56","TopicArn":"arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic","Subject":"My 2nd subject","Message":"My 2nd body","Timestamp":"2020-11-27T00:31:22.151Z","UnsubscribeURL":"https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:313276652360:my-kinesis-test-topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5","MessageAttributes":{"myKey1":{"Type":"String","Value":"myValue1"}}}
{"Type":"Notification","MessageId":"816cd54d-8cfa-58ad-91c9-8d77c7d173aa","TopicArn":"arn:aws:sns:us-east-1:333333333333:my-kinesis-test-topic","Subject":"My 3rd subject","Message":"My 3rd body","Timestamp":"2020-11-27T00:31:39.755Z","UnsubscribeURL":"https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:313276652360:my-kinesis-test-topic:0b410f3c-ee5e-49d8-b59b-3b4aa6d8fcf5"}

```

Athena を使用して Amazon S3 に保存された Amazon SNS メッセージを分析する

このページでは、配信ストリームを介して Amazon Simple Storage Service (Amazon S3) の送信先に送信された Amazon SNS メッセージを分析する方法について説明します。

Firehose 配信ストリームを介して Amazon S3 送信先に送信された SNS メッセージを分析するには

1. Amazon S3 リソースを設定します。手順については、『Amazon Simple Storage Service ユーザーガイド』の「[バケットの作成](#)」および『Amazon Simple Storage Service ユーザーガイド』の「[Amazon S3 バケットの使用](#)」を参照してください。
2. 配信ストリームを設定します。手順については、「Amazon Data Firehose デベロッパーガイド」の「[配信先の Amazon S3 の選択](#)」を参照してください。
3. [Amazon Athena](#) を使用して、標準 SQL を使用した Amazon S3 オブジェクトのクエリを実行します。詳細については、『Amazon Athena ユーザーガイド』の「[開始方法](#)」を参照してください。

クエリの例

このクエリの例では、次のことを前提としています。

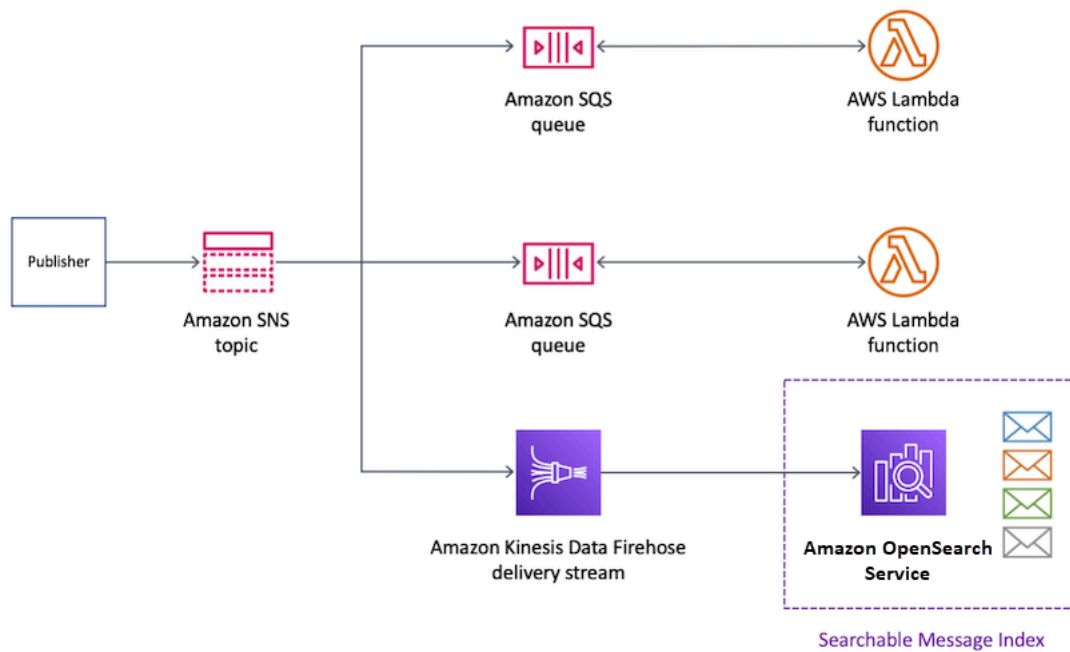
- メッセージは default スキーマの notifications テーブルに保存されます。
- notifications テーブルは string のタイプの timestamp カラムを含みます。

次のクエリは、指定された日付の範囲内で受信されたすべての SNS メッセージを返します。

```
SELECT *
FROM default.notifications
WHERE from_iso8601_timestamp(timestamp) BETWEEN TIMESTAMP '2020-12-01 00:00:00' AND
TIMESTAMP '2020-12-02 00:00:00';
```

Amazon SNS メッセージと Amazon OpenSearch Service の送信先との統合

このセクションでは、配信ストリームが Amazon OpenSearch Service (OpenSearch Service) にデータを発行する方法について説明します。



トピック

- [OpenSearch Service インデックスでの Amazon SNS 通知の保存とフォーマット](#)
- [OpenSearch Service の宛先の Amazon SNS メッセージを分析する](#)

OpenSearch Service インデックスでの Amazon SNS 通知の保存とフォーマット

次の例は、my-index という Amazon OpenSearch Service (OpenSearch Service) インデックスに送信された Amazon SNS 通知を示しています。このインデックスには、Timestamp フィールドにタイムフィルターフィールドがあります。SNS 通知はペイロードの `_source` プロパティにあります。

Note

この例では、raw メッセージ配信は、発行されたメッセージに対して無効になっています。raw メッセージ配信が無効になっている場合、Amazon SNS は次のプロパティを含む JSON メタデータをメッセージに追加します。

- Type
- MessageId
- TopicArn
- Subject

- Timestamp
- UnsubscribeURL
- MessageAttributes

raw 配信の詳細については、「[Amazon SNS raw メッセージの配信](#)」を参照してください。

```
{
  "_index": "my-index",
  "_type": "_doc",
  "_id": "49613100963111323203250405402193283794773886550985932802.0",
  "_version": 1,
  "_score": null,
  "_source": {
    "Type": "Notification",
    "MessageId": "bf32e294-46e3-5dd5-a6b3-bad65162e136",
    "TopicArn": "arn:aws:sns:us-east-1:111111111111:my-topic",
    "Subject": "Sample subject",
    "Message": "Sample message",
    "Timestamp": "2020-12-02T22:29:21.189Z",
    "UnsubscribeURL": "https://sns.us-east-1.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-
topic:b5aa9bc1-9c3d-452b-b402-aca2cefc63c9",
    "MessageAttributes": {
      "my_attribute": {
        "Type": "String",
        "Value": "my_value"
      }
    }
  },
  "fields": {
    "Timestamp": [
      "2020-12-02T22:29:21.189Z"
    ]
  },
  "sort": [
    1606948161189
  ]
}
```

OpenSearch Service の宛先の Amazon SNS メッセージを分析する

このトピックでは、配信ストリームを介して Amazon OpenSearch Service (OpenSearch Service) の送信先に送信される Amazon SNS メッセージを分析する方法について説明します。

Firehose 配信ストリームを介して OpenSearch Service の送信先に送信される SNS メッセージを分析するには

1. OpenSearch Service リソースを設定します。手順については、Amazon OpenSearch Service デベロッパーガイドの [Amazon OpenSearch Service の開始方法](#) を参照してください。
2. 配信ストリームを設定します。手順については、「Amazon Data Firehose デベロッパーガイド」の「[送信先に OpenSearch Service を選択する](#)」を参照してください。
3. OpenSearch Service クエリと Kibana を使用してクエリを実行します。詳細については、Amazon OpenSearch Service デベロッパーガイドの [ステップ 3: OpenSearch Service ドメインでドキュメントを検索する](#) および [Kibana](#) を参照してください。

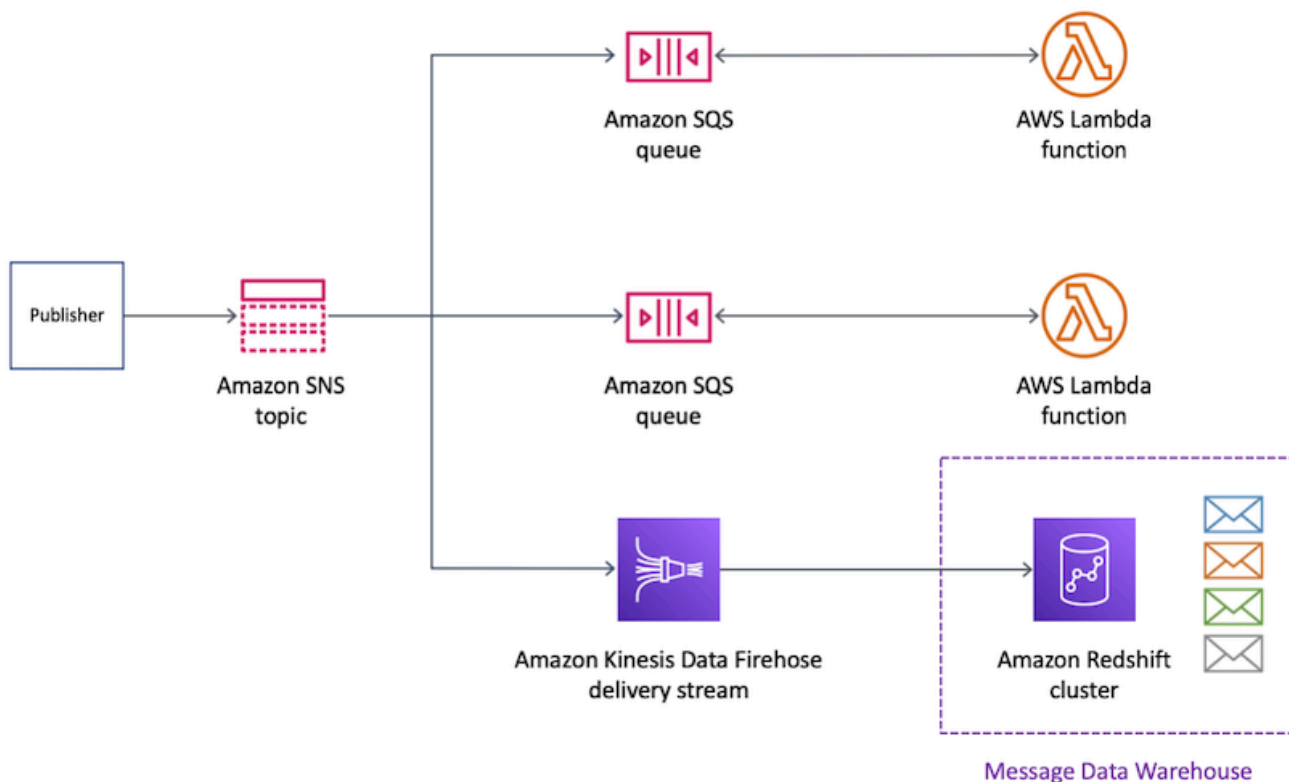
クエリの例

次の例では、指定された日付の範囲内で受信したすべての SNS メッセージの my-index インデックスのクエリを示しています。

```
POST https://search-my-domain.us-east-1.es.amazonaws.com/my-index/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "range": {
            "Timestamp": {
              "gte": "2020-12-08T00:00:00.000Z",
              "lte": "2020-12-09T00:00:00.000Z",
              "format": "strict_date_optional_time"
            }
          }
        }
      ]
    }
  }
}
```

Amazon Redshift 送信先での Amazon SNS メッセージ配信と分析を設定する

このトピックでは、Amazon SNS 通知を配信ストリームにファンアウトし、データを Amazon Redshift に発行する方法について説明します。この設定では、Amazon Redshift データベースに接続し、SQL クエリツールを使用して、特定の条件を満たす Amazon SNS メッセージを取得できます。



トピック

- [Amazon Redshift テーブルでの Amazon SNS メッセージアーカイブの構築](#)
- [Amazon Redshift の送信先に保存された Amazon SNS メッセージを分析する](#)

Amazon Redshift テーブルでの Amazon SNS メッセージアーカイブの構築

Amazon Redshift エンドポイントの場合、Amazon SNS メッセージは、テーブル内の列としてアーカイブされます。データの保存方法の例を次に示します。

Note

この例では、raw メッセージ配信は、発行されたメッセージに対して無効になっています。raw メッセージ配信が無効になっている場合、Amazon SNS は次のプロパティを含む JSON メタデータをメッセージに追加します。

- Type
- MessageId
- TopicArn
- Subject
- Message
- Timestamp
- UnsubscribeURL
- MessageAttributes

raw 配信の詳細については、「[Amazon SNS raw メッセージの配信](#)」を参照してください。Amazon SNS は、このリストに表示される頭文字を大文字の単語でメッセージにプロパティを追加しますが、Amazon Redshift テーブルではすべて小文字で表示されます。Amazon Redshift エンドポイントの JSON メタデータを変換するには、SQL の COPY コマンドを使用できます。詳細については、『Amazon Redshift データベースデベロッパーガイド』の「[JSON からのコピーの例](#)」および「['auto ignorecase' オプションを使用した JSON データからのロード](#)」を参照してください。

型	messageId	topicarn	subject	message	timestamp	unsubscribeurl	messageattributes
通知	ea544832-a0d8-581d-9275-108243c46103	arn:aws:sns:us-east-1:111111111111:my-topic	サンプル 件名	サンプル メッセー ジ	2020-12-02T00:33:32.272Z	https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-	{"my_attribute\":"Type\":"String","\Value\":"my_value"}}

型	messageId	topicArn	subject	message	timestamp	unsubscribeUrl	messageAttributes
						t-1:1111111111111111:m y-topic:3 26deeeb-c bf4-45da-b92b-ca77 a247813b	
Notification	ab124832-a0d8-581d-9275-108243c46114	arn:aws:sns:us-east-1:1111111111111111:m y-topic	サンプル 件名 2	サンプル メッセージ 2	2020-12-03T00:18:11.129Z	https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:1111111111111111:m y-topic:3 26deeeb-c bf4-45da-b92b-ca77 a247813b	{"my_attribute2":{"Type":"String","Value":"my_value"}}

型	messageId	topicArn	subject	message	timestamp	unsubscribeurl	messageAttributes
Notification	ce644832-a0d8-581d-9275-108243c46125	arn:aws:sns:us-east-1:111111111111:my-topic	サンプル件名 3	サンプルメッセージ 3	2020-12-09T00:08:44.405Z	https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-1:111111111111:my-topic:326deeeb-cbf4-45da-b92b-ca77a247813b	{"my_attribute3":{"Type":"String","Value":"my_value"}}

Amazon Redshift エンドポイントへの通知のファンアウトの詳細については、「[Amazon Redshift 送信先での Amazon SNS メッセージ配信と分析を設定する](#)」を参照してください。

Amazon Redshift の送信先に保存された Amazon SNS メッセージを分析する

このトピックでは、配信ストリームを介して Amazon Redshift の送信先に送信される Amazon SNS メッセージを分析する方法について説明します。

Firehose 配信ストリームを介して Amazon Redshift の送信先に送信される SNS メッセージを分析するには

1. Amazon Redshift リソースを設定します。手順については、『Amazon Redshift 使用開始ガイド』の「[Amazon Redshift の開始方法](#)」を参照してください。
2. 配信ストリームを設定します。手順については、「Amazon Data Firehose デベロッパーガイド」の「[配信先の Amazon Redshift の選択](#)」を参照してください。
3. クエリを実行する 詳細については、「Amazon Redshift 管理ガイド」の「[クエリエディタを使用してデータベースのクエリを実行する](#)」を参照してください。

クエリの例

このクエリの例では、次のことを前提としています。

- メッセージはデフォルト public スキーマの notifications テーブルに保存されます。
- SNS メッセージからの Timestamp プロパティは、timestampz のタイプのコラムデータを持つテーブルの timestamp カラムに保存されます。

Note

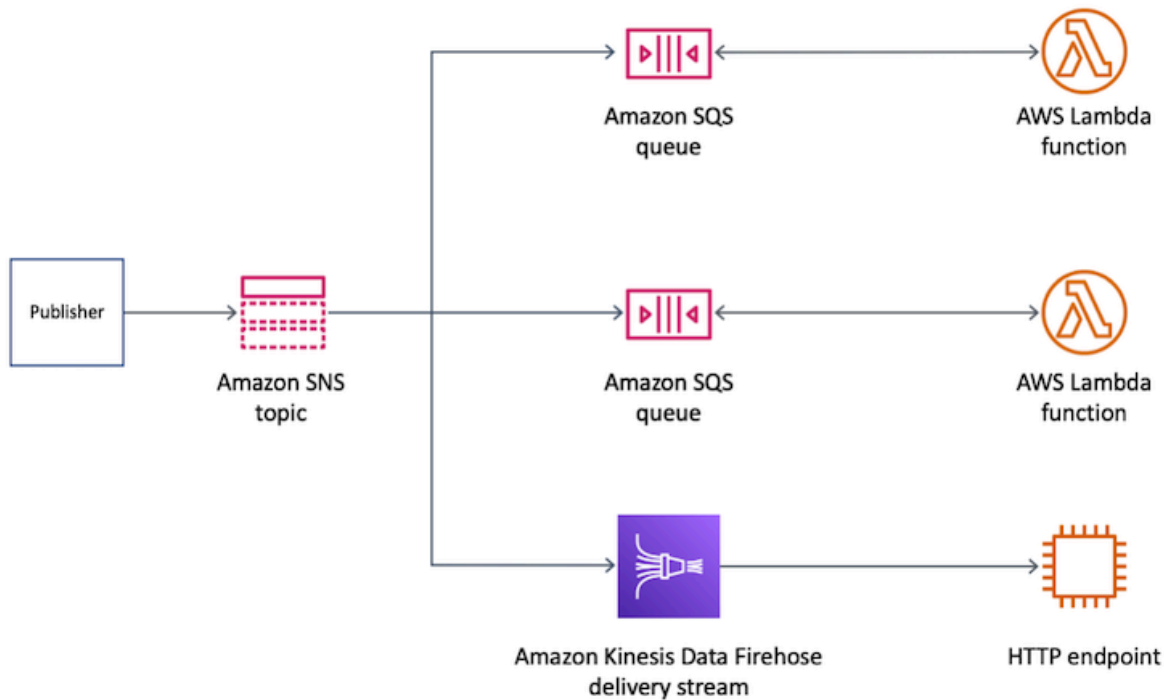
Amazon Redshift エンドポイントの JSON メタデータを変換するには、SQL COPY コマンドを使用できます。詳細については、『Amazon Redshift データベースデベロッパーガイド』の「[JSON からのコピーの例](#)」および「['auto ignorecase' オプションを使用した JSON データからのロード](#)」を参照してください。

次のクエリは、指定された日付の範囲内で受信されたすべての SNS メッセージを返します。

```
SELECT *
FROM public.notifications
WHERE timestamp > '2020-12-01T09:00:00.000Z' AND timestamp <
'2020-12-02T09:00:00.000Z';
```

HTTP 送信先への Amazon SNS メッセージ配信を設定する

このトピックでは、配信ストリームが HTTP エンドポイントにデータを発行する方法について説明します。



トピック

- [HTTP 送信先に配信するための Amazon SNS 通知フォーマット](#)

HTTP 送信先に配信するための Amazon SNS 通知フォーマット

Amazon SNS から配信ストリームを介して HTTP エンドポイントに送信される HTTP POST リクエスト本文の例を次に示します。Amazon SNS 通知は、records プロパティ内で base64 ペイロードとしてエンコードされます。

Note

この例では、raw メッセージ配信は、発行されたメッセージに対して無効になっています。raw 配信の詳細については、「[Amazon SNS raw メッセージの配信](#)」を参照してください。

```
"body": {
  "requestId": "ebc9e8b2-fce3-4aef-a8f1-71698bf8175f",
  "timestamp": 1606255960435,
```

```
"records": [  
  {  
    "data":  
      "eyJUeXB1IjoiTm90aWZpY2F0aW9uIiwidWVzc2FnZUlkIjoiMjFkMmUzOGQtMmNhYi01ZjYxLTliYTItYmJiYWZhYzg0M"  
    }  
  ]  
}
```

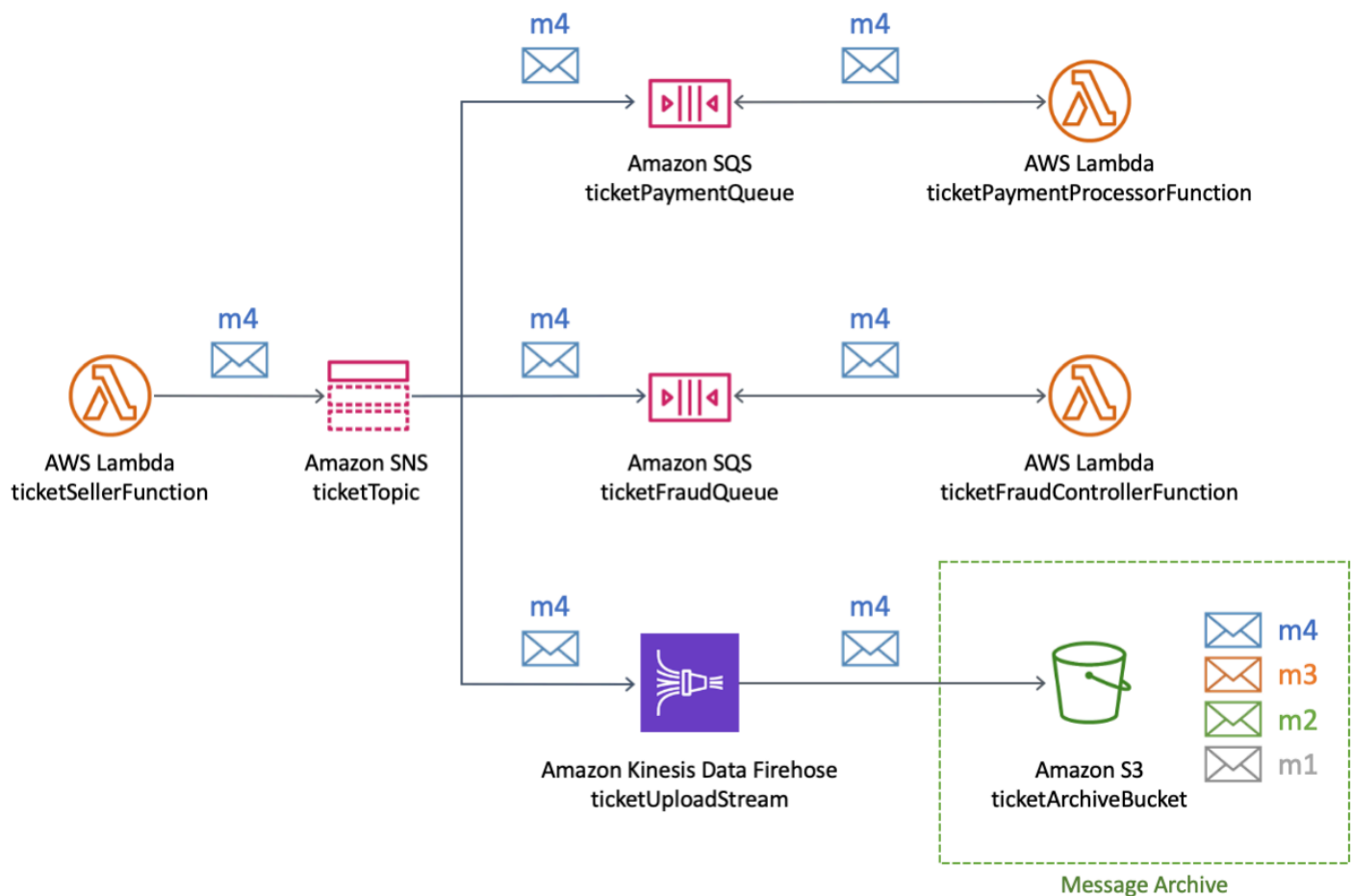
Amazon SNS メッセージのアーカイブと分析: 航空会社の発券プラットフォームのユースケース例

このトピックでは、Amazon SNS メッセージをアーカイブおよび分析する一般的なユースケースに関するチュートリアルを紹介します。

このユースケースの設定は、規制された環境で動作する航空会社の発券プラットフォームです。

1. このプラットフォームは、少なくとも5年間、すべてのチケット販売をアーカイブする必要があるコンプライアンスフレームワークの対象となります。
2. データ保持に関するコンプライアンス目標を達成するために、既存の Amazon SNS トピックに配信ストリームをサブスクライブしています。
3. 配信ストリームの送信先は、Amazon Simple Storage Service (Amazon S3) バケットです。この設定では、SNS トピックに発行されたすべてのイベントが Amazon S3 バケットにアーカイブされます。

次の図は、この設定のアーキテクチャを示しています。



分析を実行し、チケット販売に関する洞察を得るために、同社は Amazon Athena を使用して SQL クエリを実行します。例えば、同社は、最も人気のある目的地や最も頻繁に搭乗する人についてクエリを実行できます。

このユースケースの AWS リソースを作成するには、AWS マネジメントコンソール または CloudFormation テンプレートを使用できます。

トピック

- [Amazon SNS メッセージアーカイブと分析の初期 AWS リソースの設定](#)
- [Amazon SNS メッセージアーカイブ用の Amazon Data Firehose 配信ストリームの設定](#)
- [Amazon SNS トピックへ Firehose 配信ストリームをサブスクライブする](#)
- [効果的なデータ管理のための Amazon SNS 設定のテストとクエリ](#)
- [CloudFormation テンプレートを使用した Amazon SNS メッセージアーカイブの自動化](#)

Amazon SNS メッセージアーカイブと分析の初期 AWS リソースの設定

このトピックでは、[メッセージのアーカイブと分析の例を示すユースケース](#)で必要となるリソースの作成方法について説明します。

- 1 つの Amazon Simple Storage Service (Amazon S3) バケット
- 2 つの Amazon Simple Queue Service (Amazon SQS) キュー
- 1 つの Amazon SNS トピック
- Amazon SNS トピックへの 2 つの Amazon SQS キューサブスクリプション

初期リソースを作成するには

1. Amazon S3 バケットを作成する

- a. [Amazon S3 コンソール](#)を開きます。
- b. [バケットを作成する] を選択します。
- c. [バケット名] に、一意の名前を入力します。他のフィールドはデフォルトのままにします。
- d. [バケットの作成] をクリックします。

Amazon S3 バケットについては、『Amazon Simple Storage Service ユーザーガイド』の「[バケットの作成](#)」および『Amazon Simple Storage Service ユーザーガイド』の「[Amazon S3 バケットの使用](#)」を参照してください。

2. 2 つの Amazon SQS キューを作成します。

- a. [Amazon SQS コンソール](#)を開きます。
- b. [キューの作成] を選択します。
- c. タイプ] で、[標準] を選択します。
- d. [名前] に **ticketPaymentQueue** と入力します。
- e. [アクセスポリシー] の [Choose method] で、[Advanced] を選択します。
- f. [JSON ポリシー] ボックスに、以下のポリシーをペーストします。

JSON

```
{  
  "Version": "2012-10-17",
```



```
"Statement": [  
  {  
    "Effect": "Allow",  
    "Principal": {  
      "Service": "sns.amazonaws.com"  
    },  
    "Action": "sqs:SendMessage",  
    "Resource": "*",  
    "Condition": {  
      "ArnEquals": {  
        "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:ticketTopic"  
      }  
    }  
  }  
]
```

このアクセスポリシーでは、AWS アカウント 番号 (**123456789012**) を独自のものに置き換え、それに応じて AWS リージョン (**us-east-1**) を変更します。

- g. [キューの作成] を選択します。
- h. 上記のステップを繰り返して、**ticketFraudQueue** という 2 番目の SQS キューを作成します。

SQS キューの作成の詳細については、『Amazon Simple Queue Service デベロッパーガイド』の「[Amazon SQS キューの作成 \(コンソール\)](#)」を参照してください。

3. SNS トピックを作成する

- a. Amazon SNS コンソールの [\[トピック\] ページ](#)を開きます。
- b. [トピックの作成] を選択します。
- c. [詳細] の [タイプ] で、[標準] を選択します。
- d. [名前] に **ticketTopic** と入力します。
- e. [トピックを作成] を選択します。

SNS トピックの作成の詳細については、「[Amazon SNS トピックを作成する](#)」を参照してください。

4. SQS キューを SNS トピックにサブスクライブする

- a. [Amazon SNS コンソール](#)で、[ticketTopic] トピック詳細ページの [サブスクリプションの作成] を選択します。
- b. [詳細] で、[プロトコル] は [Amazon SQS] を選択します。
- c. [エンドポイント] で、ticketPaymentQueue キューの Amazon リソースネーム (ARN) を選択します。
- d. [サブスクリプションの作成] を選択します。
- e. 上記のステップを繰り返して、ticketFraudQueue キューの ARN を使用して 2 番目のサブスクリプションを作成します。

SNS トピックへのサブスクライブに関する詳細については、「[Amazon SNS トピックのサブスクリプションの作成](#)」を参照してください。また、Amazon SQS コンソールから SNS トピックに SQS キューをサブスクライブすることもできます。詳細については、『Amazon Simple Queue Service デベロッパーガイド』の「[Amazon SNS トピックへの Amazon SQS キューのサブスクライブ \(コンソール\)](#)」を参照してください。

これで、このユースケース例の初期リソースが作成できました。続行するには、「[Amazon SNS メッセージアーカイブ用の Amazon Data Firehose 配信ストリームの設定](#)」を参照してください。

Amazon SNS メッセージアーカイブ用の Amazon Data Firehose 配信ストリームの設定

このトピックでは、[メッセージのアーカイブと分析の例を示すユースケース](#)に必要な Amazon Data Firehose 配信ストリームの作成方法について説明します。

Amazon Data Firehose 配信ストリームを作成するには

1. [Amazon Data Firehose サービスコンソール](#)を開きます。
2. Firehose、配信ストリームの作成の順に選択します。
3. [New delivery stream] ページで、[Delivery stream name] に、**ticketUploadStream** を入力して、次に [Next] をクリックします。
4. [Recipients] ページで、[Next] をクリックします。
5. [Choose a Destination] ページで、次の作業を行います。
 - a. 送信先で、[Amazon S3] を選択します。
 - b. [S3 送信先] の [S3 バケット] で、[最初に作成した](#) S3 バケットを選択します。

- c. [Next] をクリックします。
6. [設定] ページの [S3 buffer conditions] で、次の作業を行います。
 - [Buffer size] に **1** を入力します。
 - [Buffer interval] に **60** を入力します。

Amazon S3 バッファにこれらの値を使用すると、設定をすばやくテストできます。最初に満たされたバッファ条件によって、S3 バケットへのデータの送信がトリガーされます。

7. アクセス許可の設定ページで、必要なアクセス許可が自動的に割り当てられた AWS Identity and Access Management (IAM) ロールを作成することを選択します。続いて、[Next] をクリックします。
8. [Review] ページで、[配信ストリームの作成] を選択します。。
9. [Amazon Data Firehose 配信ストリームページ] で、作成した配信ストリーム (ticketUploadStream) を選択します。[詳細] タブで、後でストリームの Amazon リソースネーム (ARN) を記録します。

配信ストリーム作成について詳しくは、「Amazon Data Firehose デベロッパーガイド」の「[Amazon Data Firehose 配信ストリームの作成](#)」を参照してください。IAM ロールの作成の詳細については、IAM ユーザーガイドの「[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。

これで、必要なアクセス許可を持つ Firehose 配信ストリームを作成しました。続行するには、「[Amazon SNS トピックへ Firehose 配信ストリームをサブスクライブする](#)」を参照してください。

Amazon SNS トピックへ Firehose 配信ストリームをサブスクライブする

このトピックでは、[メッセージのアーカイブと分析の例を示すユースケース](#)で必要となるリソースの作成方法について説明します。

- Amazon SNS サブスクリプションが配信ストリームにレコードを配置できるようにする AWS Identity and Access Management (IAM) ロール。
- Amazon SNS トピックへの Firehose 配信ストリームサブスクリプション。

Amazon SNS サブスクリプションの IAM ロールを作成するには

1. IAM コンソールの [\[ロール\] ページ](#)を開きます。

2. [ロールの作成] を選択します。
3. [信頼されたエンティティのタイプの選択] で、[AWS のサービス] を選択します。
4. [ユースケースの選択] で、[SNS] を選択します。接ぎ木、[Next: Permissions] 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. [Review] ページの [Role name] に、**ticketUploadStreamSubscriptionRole** を入力します。次に、[ロールの作成] を選択します。
8. ロールが作成されたら、ロールの名前 (TicketUploadStreamSubscriptionRole) を選択します。
9. ロールの [Summary] ページで、[Add inline policy] を選択します。
10. [ポリシーの作成] ページの [JSON] タブを選択し、次のポリシーをテキストボックスにペーストします。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:DescribeDeliveryStream",
        "firehose:ListDeliveryStreams",
        "firehose:ListTagsForDeliveryStream",
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Resource": [
        "arn:aws:firehose:us-east-1:123456789012:deliverystream/
ticketUploadStream"
      ],
      "Effect": "Allow"
    }
  ]
}
```

このポリシーでは、AWS アカウント 番号 (**123456789012**) を独自のものに置き換え、それに
応じて AWS リージョン (**us-east-1**) を変更します。

11. [ポリシーの確認] を選択します。
12. [ポリシーの確認] ページで [名前] に **FirehoseSnsPolicy** と入力します。次に、[ポリシーの作成] を選択します。
13. ロールの [Summary] ページで、後で [ロール ARN] を記録します。

IAM ロールの作成の詳細については、IAM ユーザーガイドの「[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。

SNS トピックへ Firehose 配信ストリームをサブスクライブするには

1. Amazon SNS コンソールの [\[トピック\] ページ](#)を開きます。
2. [サブスクリプション] タブで [サブスクリプションの作成] を選択します。
3. [詳細] で、[プロトコル] に対して [Amazon SQS] を選択します。
4. [エンドポイント] で、以前に作成した TicketUploadStream 配信ストリームの Amazon リソースネーム (ARN) を入力します。例えば、「**arn:aws:firehose:us-east-1:123456789012:deliverystream/ticketUploadStream**」と入力します。
5. [Subscription role ARN] で、以前に作成した ticketUploadStreamSubscriptionRole IAM ロールの ARN を入力します。たとえば、**arn:aws:iam::123456789012:role/ticketUploadStreamSubscriptionRole** と入力します。
6. [Enable raw message delivery] チェックボックスをオンにします。
7. [Create subscription] を選択してください。

これで、IAM ロールと SNS トピックのサブスクリプションを作成しました。続行するには、「[効果的なデータ管理のための Amazon SNS 設定のテストとクエリ](#)」を参照してください。

効果的なデータ管理のための Amazon SNS 設定のテストとクエリ

このトピックでは、Amazon SNS トピックにメッセージを発行することにより、[メッセージのアーカイブと分析の例を示すユースケース](#)をテストする方法について説明します。手順には、実行してお客様独自のニーズに適應できるクエリの例が含まれています。

設定をテストするには

1. Amazon SNS コンソールの [\[トピック\] ページ](#)を開きます。
2. [**ticketTopic** トピック] を選択します。
3. [メッセージの発行] を選択します。

4. [Publish message to topic] ページで、メッセージ本文に次のように入力します。メッセージの末尾に改行文字を追加します。

```
{"BookingDate":"2020-12-15","BookingTime":"2020-12-15  
04:15:05","Destination":"Miami","FlyingFrom":"Vancouver","TicketNumber":"abcd1234"}
```

他のすべてのオプションはデフォルト設定のままにします。

5. [メッセージの発行] を選択します。

公開の詳細については、「[Amazon SNS メッセージの発行](#)」を参照してください。

6. 配信ストリームの間隔が 60 秒になったら、[はじめに作成した Amazon Simple Storage Service \(Amazon S3\) コンソール](#)と Amazon S3 バケットを選択します。

バケットには、発行されたメッセージが表示されます。

データをクエリするには

1. [Amazon Athena コンソール](#)を開きます。
2. クエリを実行する

例えば、default スキーマの notifications テーブルには、以下のデータが含まれていることを前提とします。

```
{"BookingDate":"2020-12-15","BookingTime":"2020-12-15  
04:15:05","Destination":"Miami","FlyingFrom":"Vancouver","TicketNumber":"abcd1234"}  
{"BookingDate":"2020-12-15","BookingTime":"2020-12-15  
11:30:15","Destination":"Miami","FlyingFrom":"Omaha","TicketNumber":"efgh5678"}  
{"BookingDate":"2020-12-15","BookingTime":"2020-12-15  
3:30:10","Destination":"Miami","FlyingFrom":"NewYork","TicketNumber":"ijkl9012"}  
{"BookingDate":"2020-12-15","BookingTime":"2020-12-15  
12:30:05","Destination":"Delhi","FlyingFrom":"Omaha","TicketNumber":"mnop3456"}
```

トップの送信先を確認するには、以下のクエリを実行します。

```
SELECT destination  
FROM default.notifications  
GROUP BY destination  
ORDER BY count(*) desc  
LIMIT 1;
```

特定の日付と時刻の範囲内で販売されたチケットをクエリするには、以下のようなクエリを実行します。

```
SELECT *
FROM default.notifications
WHERE bookingtime
    BETWEEN TIMESTAMP '2020-12-15 10:00:00'
    AND TIMESTAMP '2020-12-15 12:00:00';
```

両方のサンプルクエリをお客様独自のニーズに合わせて調整できます。Athena を使用してクエリを実行する方法の詳細については、『Amazon Athena ユーザーガイド』の「[開始方法](#)」を参照してください。

クリーンアップ

テスト完了後に利用料金が発生しないようにするには、チュートリアル中に作成した以下のリソースを削除します。

- Amazon SNS サブスクリプション
- Amazon SNS トピック
- Amazon Simple Queue Service (Amazon SQS) キュー
- Amazon S3 バケット
- 配信ストリーム
- AWS Identity and Access Management (IAM) ロールとポリシー

CloudFormation テンプレートを使用した Amazon SNS メッセージアーカイブの自動化

以下の YAML テンプレートを使用して、[メッセージのアーカイブと分析例のユースケース](#)の Amazon SNS のデプロイを自動化できます。

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: Template for creating an SNS archiving use case
Resources:
  ticketUploadStream:
    DependsOn:
```

```

- ticketUploadStreamRolePolicy
Type: AWS::KinesisFirehose::DeliveryStream
Properties:
  S3DestinationConfiguration:
    BucketARN: !Sub 'arn:${AWS::Partition}:s3:::${ticketArchiveBucket}'
    BufferingHints:
      IntervalInSeconds: 60
      SizeInMBs: 1
    CompressionFormat: UNCOMPRESSED
    RoleARN: !GetAtt ticketUploadStreamRole.Arn
ticketArchiveBucket:
  Type: AWS::S3::Bucket
ticketTopic:
  Type: AWS::SNS::Topic
ticketPaymentQueue:
  Type: AWS::SQS::Queue
ticketFraudQueue:
  Type: AWS::SQS::Queue
ticketQueuePolicy:
  Type: AWS::SQS::QueuePolicy
Properties:
  PolicyDocument:
    Statement:
      Effect: Allow
      Principal:
        Service: sns.amazonaws.com
      Action:
        - sqs:SendMessage
      Resource: '*'
      Condition:
        ArnEquals:
          aws:SourceArn: !Ref ticketTopic
Queues:
  - !Ref ticketPaymentQueue
  - !Ref ticketFraudQueue
ticketUploadStreamSubscription:
  Type: AWS::SNS::Subscription
Properties:
  TopicArn: !Ref ticketTopic
  Endpoint: !GetAtt ticketUploadStream.Arn
  Protocol: firehose
  SubscriptionRoleArn: !GetAtt ticketUploadStreamSubscriptionRole.Arn
ticketPaymentQueueSubscription:
  Type: AWS::SNS::Subscription

```



```
Properties:
  TopicArn: !Ref ticketTopic
  Endpoint: !GetAtt ticketPaymentQueue.Arn
  Protocol: sqs
ticketFraudQueueSubscription:
  Type: AWS::SNS::Subscription
  Properties:
    TopicArn: !Ref ticketTopic
    Endpoint: !GetAtt ticketFraudQueue.Arn
    Protocol: sqs
ticketUploadStreamRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Sid: ''
          Effect: Allow
          Principal:
            Service: firehose.amazonaws.com
          Action: sts:AssumeRole
ticketUploadStreamRolePolicy:
  Type: AWS::IAM::Policy
  Properties:
    PolicyName: FirehoseTicketUploadStreamRolePolicy
    PolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Action:
            - s3:AbortMultipartUpload
            - s3:GetBucketLocation
            - s3:GetObject
            - s3:ListBucket
            - s3:ListBucketMultipartUploads
            - s3:PutObject
          Resource:
            - !Sub 'arn:aws:s3:::${ticketArchiveBucket}'
            - !Sub 'arn:aws:s3:::${ticketArchiveBucket}/*'
    Roles:
      - !Ref ticketUploadStreamRole
ticketUploadStreamSubscriptionRole:
  Type: AWS::IAM::Role
  Properties:
```

```
AssumeRolePolicyDocument:
  Version: '2012-10-17'
  Statement:
    - Effect: Allow
      Principal:
        Service:
          - sns.amazonaws.com
      Action:
        - sts:AssumeRole
  Policies:
    - PolicyName: SNSKinesisFirehoseAccessPolicy
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Action:
              - firehose:DescribeDeliveryStream
              - firehose:ListDeliveryStreams
              - firehose:ListTagsForDeliveryStream
              - firehose:PutRecord
              - firehose:PutRecordBatch
            Effect: Allow
            Resource:
              - !GetAtt ticketUploadStream.Arn
```

Amazon SNS 通知の Lambda 関数へのファンアウトと自動処理

Amazon SNS はと統合されているため AWS Lambda、Amazon SNS 通知に応答して Lambda 関数をトリガーできます。Lambda 関数がサブスクライブしている SNS トピックにメッセージが発行されると、発行されたメッセージのペイロードで Lambda 関数が呼び出されます。Lambda 関数は、メッセージペイロードを入力パラメータとして受け取り、メッセージの情報を操作したり、メッセージを他の SNS トピックに発行したり、メッセージを他の AWS サービスに送信したりできます。

さらに、Amazon SNS は、Lambda エンドポイントに送信されたメッセージ通知のメッセージ配信ステータス属性もサポートします。詳細については、「[Amazon SNS メッセージ配信ステータス](#)」を参照してください。

トピック

- [リージョン間で Amazon SNS と Lambda 関数を統合するための前提条件](#)
- [Lambda 関数の Amazon SNS トピックへのサブスクライブ](#)

リージョン間で Amazon SNS と Lambda 関数を統合するための前提条件

Amazon SNS 通知を使用して Lambda 関数を呼び出すには、以下が必要になります。

- Lambda 関数
- 1 つの Amazon SNS トピック

Amazon SNS で使用する Lambda 関数の作成の詳細については、「[Using Lambda with Amazon SNS](#)」を参照してください。Amazon SNS トピックの作成の詳細については、「[トピックの作成](#)」を参照してください。

Amazon SNS を使用してオプトインリージョンからデフォルトで有効になっているリージョンにメッセージを配信する場合、プリンシパル `sns.amazonaws.com` を `sns.<opt-in-region>.amazonaws.com` に置き換えて、AWS Lambda 関数で作成されたポリシーを変更する必要があります。

例えば、米国東部 (バージニア北部) の Lambda 関数をアジアパシフィック (香港) の SNS トピックにサブスクライブする場合は、AWS Lambda 関数ポリシーのプリンシパルを `sns.ap-east-1.amazonaws.com` に変更します。オプトインリージョンには、2019 年 3 月 20 日以降に開始されたリージョンが含まれます。これには、アジアパシフィック (香港)、中東 (バーレーン)、欧州 (ミラノ)、アフリカ (ケープタウン) が含まれます。2019 年 3 月 20 日以前に開始されたリージョンは、デフォルトで有効になっています。

Note

AWS は、デフォルトで有効になっているリージョンからオプトインリージョンへの Lambda へのクロスリージョン配信をサポートしていません。オプトインリージョンから他のオプトインリージョンへの SNS メッセージのクロスリージョン転送もサポートされていません。

Lambda 関数の Amazon SNS トピックへのサブスクライブ

このトピックでは、Lambda 関数を Amazon SNS トピックにサブスクライブする方法について説明します。これにより、発行されたメッセージによって関数をトリガーすることができます。

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[トピック] を選択します。
3. [トピック] ページでトピックを選択します。

4. [サブスクリプション] セクションで、[サブスクリプションの作成] を選択します。
5. [サブスクリプションの作成] ページの [詳細] セクションで、以下を実行します。
 - a. 選択した [トピック ARN] を確認します。
 - b. Protocol で を選択します AWS Lambda。
 - c. [エンドポイント] に関数の ARN を入力します。
 - d. [サブスクリプションの作成] を選択します。

Lambda 関数がサブスクライブしている SNS トピックにメッセージが発行されると、発行されたメッセージのペイロードで Lambda 関数が呼び出されます。チュートリアルを含む Amazon SNS AWS Lambda で を使用する方法については、[Amazon SNS AWS Lambda で を使用する](#)」を参照してください。

Amazon SNS 通知の Amazon SQS キューへのファンアウトと非同期処理

[Amazon SNS](#) は、Amazon Simple Queue Service (Amazon SQS) と密接に連携します。これらのサービスは、デベロッパーにとって異なる利点を提供します。Amazon SNS を使用すると、アプリケーションは「プッシュ」メカニズムを使用して、複数の受信者にタイムクリティカルなメッセージを送信することができます。そのため、更新を定期的に確認または「ポーリング」する必要がありません。Amazon SQS は、ポーリングモデルを通してメッセージを交換するために分散型アプリケーションによって使用されるメッセージキューサービスであり、このサービスを使用すると、送信および受信コンポーネントを切り離すことができます。この場合、各コンポーネントが同時に使用できなくなることはありません。Amazon SNS と Amazon SQS を組み合わせて使用することで、イベントの即時通知を必要とするアプリケーションにメッセージを配信できるだけでなく、他のアプリケーションのメッセージは、後で処理できるように Amazon SQS キューに保持することもできます。

Amazon SNS トピックへの Amazon SQS キューをサブスクライブすると、そのトピックにメッセージを発行することができ、Amazon SNS は Amazon SQS メッセージをサブスクライブされたキューに送信します。Amazon SQS メッセージには、トピックに発行された件名とメッセージに加え、JSON ドキュメントのメッセージに関するメタデータが含まれます。Amazon SQS メッセージは、以下の JSON ドキュメントのようになります。

```
{
  "Type" : "Notification",
  "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
```

```
{
  "Subject" : "Testing publish to subscribed queues",
  "Message" : "Hello world!",
  "Timestamp" : "2012-03-29T05:12:16.901Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEnTrFPa3...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:123456789012:MyTopic:c7fe3a54-
ab0e-4ec2-88e0-db410a0f2bee"
}
```

Amazon SNS トピックへ Amazon SQS キューをサブスクライブする

Amazon SNS トピックから Amazon SQS キューにメッセージを送信できるようにするには、次のいずれかを行います。

- プロセスを簡素化するため、[Amazon SQS コンソール](#)を使用します。詳細については、『Amazon Simple Queue Service デベロッパーガイド』の「[Amazon SNS トピックへの Amazon SQS キューのサブスクライブ](#)」を参照してください。
- 以下のステップを使用します。
 1. [メッセージの送信先にするキューとキューをサブスクライブするトピックの Amazon Resource Name \(ARN\) を取得します。](#)
 2. [Amazon SNS トピックに sqs:SendMessage 許可を付与し、キューにメッセージを送信できるようにします。](#)
 3. [キューを Amazon SNS トピックへサブスクライブします。](#)
 4. [IAM ユーザーまたは AWS アカウント に、Amazon SNS トピックに発行し、Amazon SQS キューからのメッセージを読むための許可を付与します。](#)
 5. [トピックにメッセージを発行し、キューからのメッセージを読むことでテストします。](#)

別の AWS アカウントにあるキューにメッセージを送信するようにトピックをセットアップする方法については、「[別のアカウントの Amazon SQS キューへ Amazon SNS メッセージを送信する](#)」を参照してください。

2 つのキューにメッセージを送信するトピックを作成する CloudFormation テンプレートを確認するには、「[」を参照してくださいを使用して Amazon SNS から Amazon SQS へのメッセージングを自動化する AWS CloudFormation](#)。

ステップ 1: キューとトピックの ARN を取得する

トピックにキューをサブスクライブするときは、キューの ARN のコピーが必要です。同様に、トピックがキューにメッセージを送ることを許可するには、トピックの ARN のコピーが必要です。

キューの ARN を取得するには、Amazon SQS コンソールまたは [GetQueueAttributes](#) API アクションを使用できます。

キューの ARN を Amazon SQS コンソールから取得するには

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/sqs/> で Amazon SQS コンソールを開きます。
2. ARN を取得するキューのチェックボックスをオンにします。
3. [詳細] タブから、Amazon SNS トピックへのサブスクライブに使用できるように ARN 値をコピーします。

トピックの ARN を取得するには、Amazon SNS コンソール、[sns-get-topic-attributes](#) コマンド、または [GetQueueAttributes](#) API アクションを使用できます。

トピックの ARN を Amazon SNS コンソールから取得するには

1. [Amazon SNS コンソール](#) にサインインします。
2. ナビゲーションパネルで、ARN を取得するトピックを選択します。
3. 詳細セクションで、ARN 値を使用して Amazon SNS トピックがメッセージを送信する許可を取得できるようにします。

ステップ 2: Amazon SQS キューにメッセージを送信する許可を Amazon SNS トピックに付与する

Amazon SNS トピックがキューにメッセージを送信できるようにするには、Amazon SNS トピックに `sqs:SendMessage` アクションの実行を許可するようにキューのポリシーを設定する必要があります。

トピックにキューをサブスクライブする前に、トピックとキューが必要です。トピックやキューをまだ作成していない場合は、ここで作成します。詳細については、「[トピックの作成](#)」と、「Amazon Simple Queue Service デベロッパーガイド」の「[キューの作成](#)」を参照してください。

キューにポリシーを設定するには、Amazon SQS コンソールまたは [SetQueueAttributes](#) API アクションを使用できます。開始する前に、キューにメッセージを送信できるようにするトピックの ARN があることを確認してください。複数のトピックにキューをサブスクライブする場合、ポリシーにはトピックごとに 1 つの Statement 要素が含まれている必要があります。

Amazon SQS コンソールを使用してキューに SendMessage ポリシーを設定するには

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/sqs/> で Amazon SQS コンソールを開きます。
2. ポリシーを設定するキューのチェックボックスをオンにし、[アクセスポリシー] タブを選択してから、[編集] を選択します。
3. [アクセスポリシー] セクションで、キューにアクセスできるユーザーを定義します。
 - トピックにアクションを許可する条件を追加します。
 - 以下の例に示すように、Amazon SNS サービスを Principal に設定します。
 - [aws:SourceArn](#) または [aws:SourceAccount](#) グローバル条件キーを使用して、[混乱した代理シナリオ](#)から保護します。これらの条件キーを使用するには、値をトピックの ARN に設定します。キューを複数のトピックにサブスクライブしている場合は、代わりに [aws:SourceAccount](#) を使用できます。

このポリシーは、MyTopic が MyQueue にメッセージを送ることを許可します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sns.amazonaws.com"
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:sns:us-east-2:123456789012:MyTopic"
        }
      }
    }
  ]
}
```

ステップ 3: キューを Amazon SNS トピックへサブスクライブします。

トピックを介してキューにメッセージを送信するには、キューを Amazon SNS トピックにサブスクライブする必要があります。キューは ARN で指定します。トピックにサブスクライブするには、Amazon SNS コンソール、[sns-subscribe](#) CLI コマンド、または [Subscribe](#) API アクションを使用できます。開始する前に、サブスクライブするキューの ARN があることを確認してください。

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[トピック] を選択します。
3. [トピック] ページでトピックを選択します。
4. **[MyTopic]** ページの [サブスクリプション] ページで [サブスクリプションの作成] を選択します。
5. [サブスクリプションの作成] ページの [詳細] セクションで、以下を実行します。
 - a. [トピックの ARN] を検証します。
 - b. [プロトコル] で、[Amazon SQS] を選択します。
 - c. [エンドポイント] に、Amazon SQS キューの ARN を入力します。
 - d. [サブスクリプションの作成] を選択します。

サブスクリプションが確認されると、新しいサブスクリプションの [サブスクリプション ID] にサブスクリプション ID が表示されます。キューの所有者がサブスクリプションを作成する場合、サブスクリプションは自動的に確認され、ほぼ瞬時にアクティブになります。

通常は、自分のアカウントの自分のトピックに自分のキューをサブスクライブします。ただし、別のアカウントのキューを自分のトピックにサブスクライブすることもできます。サブスクリプションを作成するユーザーがキューの所有者ではない場合 (例えば、アカウント A のユーザーがアカウント A のトピックにアカウント B のキューをサブスクライブする場合)、サブスクリプションの確認が必要です。別のアカウントのキューをサブスクライブし、サブスクリプションを確認する方法の詳細については、「[別のアカウントの Amazon SQS キューへ Amazon SNS メッセージを送信する](#)」を参照してください。

ステップ 4: 適切なトピックおよびキューアクションに対するアクセス権限をユーザーに付与する

AWS Identity and Access Management (IAM) を使用して、適切なユーザーのみが Amazon SNS トピックに発行し、Amazon SQS キューからのメッセージの読み取り/削除を許可する必要があります。IAM ユーザーに対するトピックおよびキューに対するアクションの制御の詳細については、「[Amazon SNS でのアイデンティティベースのポリシーを使用する](#)」および『Amazon Simple Queue Service デベロッパーガイド』の「[Amazon SQS での Identity and Access Management](#)」を参照してください。

トピックまたはキューへのアクセスは、以下の 2 つの方法で制御します。

- [IAM ユーザーまたはグループにポリシーを追加する](#)。ユーザーにトピックやキューへのアクセス権限を付与する最も簡単な方法として、グループを作成し、そのグループに適切なポリシーとユーザーを追加することができます。個々のユーザーに設定するポリシーを継続的に追跡するよりも、グループに対してユーザーを追加または削除する方がはるかに簡単です。
- [トピックまたはキューにポリシーを追加する](#)。トピックまたはキューへのアクセス許可を別の AWS アカウントに付与する場合は、アクセス許可を付与 AWS アカウント するプリンシパルとして を持つポリシーを追加することしかできません。

ほとんどの場合は、最初の方法 (ポリシーをグループに適用し、適切なユーザーをグループに追加または削除することでアクセス権限を管理する) を使用します。別のアカウントのユーザーにアクセス権限を付与する場合は、2 番目の方法を使用する必要があります。

IAM ユーザーまたはグループにポリシーを追加する

IAM ユーザーまたはグループに次のポリシーを追加した場合、そのユーザーまたはそのグループのメンバーに、MyTopic トピックで `sns:Publish` アクションを実行する許可が付与されます。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
    }
  ]
}
```

IAM ユーザーまたはグループに次のポリシーを追加した場合、ユーザーまたはそのグループのメンバーに、キューの「MyQueue 1」と「MyQueue 2」に対して `sqs:ReceiveMessage` および `sqs:DeleteMessage` アクションを実行する許可が付与されます。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage"
      ],
      "Resource": [
        "arn:aws:sqs:us-east-2:123456789012:MyQueue1",
        "arn:aws:sqs:us-east-2:123456789012:MyQueue2"
      ]
    }
  ]
}
```

トピックまたはキューにポリシーを追加する

以下のサンプルポリシーは、トピックとキューに別のアカウントのアクセス権限を付与する方法を示しています。

 Note

アカウントのリソースへの別の AWS アカウント アクセスを許可すると、そのリソースへの管理者レベルのアクセス (ワイルドカードアクセス) アクセス許可を持つ IAM ユーザーにも付与されます。他のアカウントの他のすべての IAM ユーザーは、自動的にリソースへのアクセスが拒否されます。その AWS アカウント アクセスの特定の IAM ユーザーにリソースへのアクセス権を付与する場合、管理者レベルアクセス権を持っているアカウントまたは IAM ユーザーは、そのリソースのアクセス権限をそれらの IAM ユーザーに委任する必要があります。クロスアカウントの委任の詳細については、『IAM ガイドの使用』の「[Enabling Cross-Account Access](#)」を参照してください。

アカウント 123456789012 の「マイトピック」トピックに次のポリシーを追加した場合、そのトピックで `sns:Publish` アクションを実行するアクセス権限をアカウント 111122223333 に付与したことになります。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "111122223333"
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
    }
  ]
}
```

アカウント 123456789012 のキュー「マイキュー」に以下のポリシーを追加した場合、そのキューで `sqs:DeleteMessage` および `sqs:ReceiveMessage` アクションを実行する許可をアカウント 111122223333 に付与したことになります。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "111122223333"
      },
      "Action": [
        "sqs:DeleteMessage",
        "sqs:ReceiveMessage"
      ],
      "Resource": [
        "arn:aws:sqs:us-east-2:123456789012:MyQueue"
      ]
    }
  ]
}
```

ステップ 5: トピックのキューサブスクリプションをテストする

トピックのキューサブスクリプションは、トピックに発行し、トピックがキューに送信したメッセージを表示することでテストできます。

Amazon SNS コンソールを使用してトピックに発行するには

1. トピックに発行するアクセス許可を持つ AWS アカウント または IAM ユーザーの認証情報を使用して、 にサインイン AWS マネジメントコンソール し、 <https://console.aws.amazon.com/sns/> で Amazon SNS コンソールを開きます。
2. ナビゲーションパネルで、トピックを選択し、[トピックに発行] を選択します。
3. [件名] ボックスに件名 (「**Testing publish to queue**」など) を入力し、[メッセージ] ボックスに任意のテキスト (「**Hello world!**」など) を入力して、[メッセージの発行] を選択します。「Your message has been successfully published.」というメッセージが表示されます。

Amazon SQS コンソールを使用してトピックからのメッセージを表示するには

1. キュー内のメッセージを表示するアクセス許可を持つ AWS アカウント または IAM ユーザーの認証情報を使用して、 にサインイン AWS マネジメントコンソール し、 <https://console.aws.amazon.com/sqs/> で Amazon SQS コンソールを開きます。
2. トピックにサブスクライブされている [queue] (キュー) を選択します。
3. [Send and receive messages] (メッセージの送受信) を選択し、[Poll for messages] (メッセージのポーリング) を選択します。タイプが [通知] のメッセージが表示されます。
4. [本文] カラムで、[詳細] を選択します。[メッセージ詳細] ボックスに、トピックに発行した件名とメッセージを含む JSON ドキュメントが表示されます。メッセージは、以下の JSON ドキュメントのようになります。

```
{
  "Type" : "Notification",
  "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "Testing publish to subscribed queues",
  "Message" : "Hello world!",
  "Timestamp" : "2012-03-29T05:12:16.901Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEnTrFPa3...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/
SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
west-2:123456789012:MyTopic:c7fe3a54-ab0e-4ec2-88e0-db410a0f2bee"
}
```

5. [閉じる] を選択します。キューに通知メッセージを送信するトピックへの発行は、正常に終了しました。

を使用して Amazon SNS から Amazon SQS へのメッセージングを自動化する AWS CloudFormation

CloudFormation では、テンプレートファイルを使用して、リソースの AWS コレクションを 1 つのユニットとしてまとめて作成および設定できます。このセクションでは、キューに発行するトピックのデプロイを容易にするサンプルテンプレートを使用します。このテンプレートは、2 つのキューの作成、キューにサブスクリプションを持つトピックの作成、トピックがキューにメッセージを送信できるようにするポリシーのキューへの追加、これらのリソースへのアクセスを制御する IAM ユーザーとグループの作成を行って、セットアップ手順を代行します。

CloudFormation テンプレートを使用した AWS リソースのデプロイの詳細については、[「ユーザーガイド」の「開始方法」](#)を参照してください。CloudFormation

CloudFormation テンプレートを使用して 内のトピックとキューを設定する AWS アカウント

このサンプルテンプレートでは、1 つは IAM グループのメンバーがトピックに発行するため、もう 1 つはキューからのメッセージを読み取るために、それぞれ適切なアクセス権限を付与された 2 つの Amazon SQS キューにメッセージを送信できる Amazon SNS トピックを作成します。このテンプレートは、各グループに追加される IAM ユーザーも作成します。

テンプレートの内容をファイルにコピーします。[\[AWS CloudFormation テンプレート\] ページ](#)からテンプレートをダウンロードすることもできます。テンプレートページで、AWS サービス別にサンプルテンプレートを参照を選択し、Amazon Simple Queue Service を選択します。

MySNSTopic は 2 つのサブスクライブされたエンドポイント (MyQueue1 と MyQueue2 という 2 つの Amazon SQS キュー) に発行するようにセットアップされます。MyPublishTopicGroup は、メンバーが API アクションの [\[Publish\]](#) または [\[sns-publish\]](#) コマンドを使用して MySNSTopic に発行する許可を持つ IAM グループです。テンプレートは、MyPublishUser と MyQueueUser という IAM ユーザーを作成し、このユーザーにログインプロファイルとアクセスキーを付与します。このテンプレートを使用してスタックを作成するユーザーは、入力パラメータとしてログインプロファイル用のパスワードを指定します。テンプレートは、2 人の IAM ユーザー用に MyPublishUserKey と MyQueueUserKey というアクセスキーを作成します。AddUserToMyPublishTopicGroup は MyPublishTopicGroup に MyPublishUser を追加して、グループに割り当てられたアクセス権限がユーザーに付与されるようにします。

MyRDMessageQueueGroup はメンバーが [ReceiveMessage](#) および [DeleteMessage](#) API アクションを使用して、2 つの Amazon SQS キューからメッセージを読み取り/削除するためのアクセス権限を持つ IAM グループです。AddUserToMyQueueGroup は MyQueueUser を MyRDMessageQueueGroup に追加して、グループに割り当てられたアクセス権限がユーザーに付与されるようにします。MyQueuePolicy は、MySNSTopic が 2 つのキューに通知を発行する許可を割り当てます。

次のリストは、CloudFormation テンプレートの内容を示しています。

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",

  "Description" : "AWS CloudFormation Sample Template SNSToSQS: This Template creates
an SNS topic that can send messages to
two SQS queues with appropriate permissions for one IAM user to publish to the topic
and another to read messages from the queues.
MySNSTopic is set up to publish to two subscribed endpoints, which are two SQS queues
(MyQueue1 and MyQueue2). MyPublishUser is an IAM user
that can publish to MySNSTopic using the Publish API. MyTopicPolicy assigns that
permission to MyPublishUser. MyQueueUser is an IAM user
that can read messages from the two SQS queues. MyQueuePolicy assigns those
permissions to MyQueueUser. It also assigns permission for
MySNSTopic to publish its notifications to the two queues. The template creates
access keys for the two IAM users with MyPublishUserKey
and MyQueueUserKey. ***Warning*** you will be billed for the AWS resources used if
you create a stack from this template.",

  "Parameters": {
    "MyPublishUserPassword": {
      "NoEcho": "true",
      "Type": "String",
      "Description": "Password for the IAM user MyPublishUser",
      "MinLength": "1",
      "MaxLength": "41",
      "AllowedPattern": "[a-zA-Z0-9]*",
      "ConstraintDescription": "must contain only alphanumeric characters."
    },
    "MyQueueUserPassword": {
      "NoEcho": "true",
      "Type": "String",
      "Description": "Password for the IAM user MyQueueUser",
      "MinLength": "1",
      "MaxLength": "41",
```

```
    "AllowedPattern": "[a-zA-Z0-9]*",
    "ConstraintDescription": "must contain only alphanumeric characters."
  }
},

"Resources": {
  "MySNSTopic": {
    "Type": "AWS::SNS::Topic",
    "Properties": {
      "Subscription": [{
        "Endpoint": {
          "Fn::GetAtt": ["MyQueue1", "Arn"]
        },
        "Protocol": "sqs"
      },
      {
        "Endpoint": {
          "Fn::GetAtt": ["MyQueue2", "Arn"]
        },
        "Protocol": "sqs"
      }
    ]
  },
  "MyQueue1": {
    "Type": "AWS::SQS::Queue"
  },
  "MyQueue2": {
    "Type": "AWS::SQS::Queue"
  },
  "MyPublishUser": {
    "Type": "AWS::IAM::User",
    "Properties": {
      "LoginProfile": {
        "Password": {
          "Ref": "MyPublishUserPassword"
        }
      }
    }
  },
  "MyPublishUserKey": {
    "Type": "AWS::IAM::AccessKey",
    "Properties": {
```

```
    "UserName": {
      "Ref": "MyPublishUser"
    }
  },
  "MyPublishTopicGroup": {
    "Type": "AWS::IAM::Group",
    "Properties": {
      "Policies": [{
        "PolicyName": "MyTopicGroupPolicy",
        "PolicyDocument": {
          "Statement": [{
            "Effect": "Allow",
            "Action": [
              "sns:Publish"
            ],
            "Resource": {
              "Ref": "MySNSTopic"
            }
          }]
        }
      }]
    }
  },
  "AddUserToMyPublishTopicGroup": {
    "Type": "AWS::IAM::UserToGroupAddition",
    "Properties": {
      "GroupName": {
        "Ref": "MyPublishTopicGroup"
      },
      "Users": [{
        "Ref": "MyPublishUser"
      }]
    }
  },
  "MyQueueUser": {
    "Type": "AWS::IAM::User",
    "Properties": {
      "LoginProfile": {
        "Password": {
          "Ref": "MyQueueUserPassword"
        }
      }
    }
  }
}
```



```
    },
    "MyQueueUserKey": {
      "Type": "AWS::IAM::AccessKey",
      "Properties": {
        "UserName": {
          "Ref": "MyQueueUser"
        }
      }
    },
  },
  "MyRDMessageQueueGroup": {
    "Type": "AWS::IAM::Group",
    "Properties": {
      "Policies": [{
        "PolicyName": "MyQueueGroupPolicy",
        "PolicyDocument": {
          "Statement": [{
            "Effect": "Allow",
            "Action": [
              "sqs:DeleteMessage",
              "sqs:ReceiveMessage"
            ]
          }],
          "Resource": [{
            "Fn::GetAtt": ["MyQueue1", "Arn"]
          },
          {
            "Fn::GetAtt": ["MyQueue2", "Arn"]
          }
        ]
      }]
    }
  },
  "AddUserToMyQueueGroup": {
    "Type": "AWS::IAM::UserToGroupAddition",
    "Properties": {
      "GroupName": {
        "Ref": "MyRDMessageQueueGroup"
      },
      "Users": [{
        "Ref": "MyQueueUser"
      }]
    }
  },
},
```

```

    "MyQueuePolicy": {
      "Type": "AWS::SQS::QueuePolicy",
      "Properties": {
        "PolicyDocument": {
          "Statement": [{
            "Effect": "Allow",
            "Principal": {
              "Service": "sns.amazonaws.com"
            },
            "Action": ["sqs:SendMessage"],
            "Resource": "*",
            "Condition": {
              "ArnEquals": {
                "aws:SourceArn": {
                  "Ref": "MySNSTopic"
                }
              }
            }
          }]
        },
        "Queues": [{
          "Ref": "MyQueue1"
        }, {
          "Ref": "MyQueue2"
        }]
      }
    },
    "Outputs": {
      "MySNSTopicTopicARN": {
        "Value": {
          "Ref": "MySNSTopic"
        }
      },
      "MyQueue1Info": {
        "Value": {
          "Fn::Join": [
            " ",
            [
              "ARN:",
              {
                "Fn::GetAtt": ["MyQueue1", "Arn"]
              }
            ]
          ],
          "URL:",

```

```
        {
            "Ref": "MyQueue1"
        }
    ]
}
},
"MyQueue2Info": {
    "Value": {
        "Fn::Join": [
            " ",
            [
                "ARN:",
                {
                    "Fn::GetAtt": ["MyQueue2", "Arn"]
                },
                "URL:",
                {
                    "Ref": "MyQueue2"
                }
            ]
        ]
    }
},
"MyPublishUserInfo": {
    "Value": {
        "Fn::Join": [
            " ",
            [
                "ARN:",
                {
                    "Fn::GetAtt": ["MyPublishUser", "Arn"]
                },
                "Access Key:",
                {
                    "Ref": "MyPublishUserKey"
                },
                "Secret Key:",
                {
                    "Fn::GetAtt": ["MyPublishUserKey", "SecretAccessKey"]
                }
            ]
        ]
    }
}
```

```

    },
    "MyQueueUserInfo": {
      "Value": {
        "Fn::Join": [
          " ",
          [
            "ARN:",
            {
              "Fn::GetAtt": ["MyQueueUser", "Arn"]
            },
            "Access Key:",
            {
              "Ref": "MyQueueUserKey"
            },
            "Secret Key:",
            {
              "Fn::GetAtt": ["MyQueueUserKey", "SecretAccessKey"]
            }
          ]
        ]
      }
    }
  }
}

```

Amazon SNS 通知の HTTPS エンドポイントへのファンアウト

Amazon SNS を使用して、1 つ以上の HTTP または HTTPS エンドポイントに通知メッセージを送信できます。エンドポイントをトピックにサブスクライブすると、トピックに通知を発行することができ、Amazon SNS は HTTP POST リクエストを送信し、通知の内容をサブスクライブしたエンドポイントに配信します。エンドポイントをサブスクライブする際は、Amazon SNS が POST リクエストをエンドポイントに送信するときに HTTP を使用するか HTTPS を使用するかを選択します。HTTPS を使用する場合は、以下に対して Amazon SNS のサポートを利用できます。

- **Server Name Indication (SNI)** - これにより、Amazon SNS は複数のドメインをホストするために複数の証明書を必要とするサーバーなど、SNI を必要とする HTTPS エンドポイントをサポートすることができます。SNI の詳細については、「[Server Name Indication](#)」を参照してください。
- **Basic and Digest Access Authentication** - これにより、HTTP POST リクエストで HTTPS URL にユーザーネームとパスワードを指定できます (https://

user:password@domain.com、https://user@domain.com など)。HTTPS を使用する場合は、ユーザーネームとパスワードは確立された SSL 接続で暗号化されます。ドメイン名のみがプレーンテキストで送信されます。Basic and Digest Access Authentication の詳細については、[RFC-2617](#) を参照してください。

 Important

Amazon SNS は現在、プライベート HTTP(S) エンドポイントをサポートしていません。HTTPS URL は API アクセスを許可したプリンシパルの場合、Amazon SNS GetSubscriptionAttributes API アクションからのみ取得できます。

 Note

クライアントサービスは HTTP/1.1 401 Unauthorized ヘッダーレスポンスをサポートできる必要があります。

リクエストには、JSON ドキュメントの通知に関するメタデータとともに、トピックに発行された件名とメッセージが含まれます。リクエストは以下の HTTP POST リクエストのようになります。HTTP ヘッダーおよびリクエストボディの JSON 形式の詳細については、「[HTTP/HTTPS ヘッダー](#)」および「[HTTP/HTTPS 通知の JSON 形式](#)」を参照してください。

 Note

Amazon SNS では、すべての 5XX エラーと 429 エラー (送信されたリクエストが多すぎます) を再試行可能なエラーとして扱います。これらのエラーには、配信ポリシーが適用されます。他のすべてのエラーは永続的なエラーとして扱われ、再試行は行われません。

POST / HTTP/1.1

```
x-amz-sns-message-type: Notification
x-amz-sns-message-id: da41e39f-ea4d-435a-b922-c6aae3915ebe
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 761
Content-Type: text/plain; charset=UTF-8
```

```
Host: ec2-50-17-44-49.compute-1.amazonaws.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent
```

```
{
  "Type" : "Notification",
  "MessageId" : "da41e39f-ea4d-435a-b922-c6aae3915ebe",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "test",
  "Message" : "test message",
  "Timestamp" : "2012-04-25T21:49:25.719Z",
  "SignatureVersion" : "1",
  "Signature" :
    "EXAMPLE1DMXvB8r9R83tGoNn0ecwd5UjllzsvSvbItzfaMpN2nk5HVS7Xn0n/49IkxDKz8Yr1H2qJXj2iZB0Zo2071c4
    "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
    f3ecfb7224c7233fe7bb5f59f96de52f.pem",
    "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
    Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
    west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55"
}
```

Amazon SNS トピックに HTTP/S エンドポイントをサブスクライブする

このトピックでは、HTTP/S エンドポイントを Amazon SNS トピックにサブスクライブする方法について説明します。

トピック

- [ステップ 1: エンドポイントで Amazon SNS メッセージを処理する準備が完了していることを確認する](#)
- [ステップ 2: Amazon SNS トピックに HTTP/HTTPS エンドポイントをサブスクライブする](#)
- [ステップ 3: Amazon SNS サブスクリプションを確認する](#)
- [ステップ 4: Amazon SNS サブスクリプションの配信ポリシーを設定する \(オプション\)](#)
- [ステップ 5: Amazon SNS トピックに発行するアクセス許可をユーザーに付与する \(オプション\)](#)
- [ステップ 6: HTTP/HTTPS エンドポイントに Amazon SNS メッセージを送信する](#)

ステップ 1: エンドポイントで Amazon SNS メッセージを処理する準備が完了していることを確認する

HTTP または HTTPS エンドポイントをトピックにサブスクライブする前に、Amazon SNS がサブスクライブプッシュの確認および通知メッセージの送信に使用する HTTP POST のリクエストを処理する能力が HTTP または HTTPS エンドポイントにあることを確認する必要があります。通常、これは Amazon SNS からの HTTP リクエストを処理するウェブアプリケーション (例えば、エンドポイントホストが Apache と Tomcat で Linux を実行している場合は Java servlet) の作成とデプロイを意味します。HTTP エンドポイントをサブスクライブする場合、Amazon SNS はサブスクリプションの確認リクエストを送信します。サブスクリプションを作成するときに、エンドポイントはこのリクエストを受信して処理する準備ができていなければならない必要があります。Amazon SNS はこの時点でこのリクエストを送信するためです。Amazon SNS は、サブスクリプションが確認されるまでエンドポイントにメッセージを送信しません。Amazon SNS は、サブスクリプションを確認すると、サブスクライブしたトピックで発行アクションが実行されたときにエンドポイントに通知を送信します。

サブスクリプションの確認および通知メッセージを処理するようにエンドポイントを設定するには

1. コードは、Amazon SNS がエンドポイントに送信する HTTP POST リクエストの HTTP ヘッダーを読み取る必要があります。また、コードは Amazon SNS が送信したメッセージのタイプを示すヘッダーフィールド `x-amz-sns-message-type` を探す必要があります。ヘッダーを確認すると、HTTP リクエストの本文を解析することなく、メッセージタイプを判断できます。処理する必要がある 2 つのタイプ (`SubscriptionConfirmation` および `Notification`) があります。UnsubscribeConfirmation メッセージは、サブスクリプションがトピックから削除された場合のみ使用されます。

HTTP ヘッダーの詳細については、「[HTTP/HTTPS ヘッダー](#)」を参照してください。以下の HTTP POST のリクエストはサブスクリプションの確認メッセージの例です。

```
POST / HTTP/1.1
x-amz-sns-message-type: SubscriptionConfirmation
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
Content-Length: 1336
Content-Type: text/plain; charset=UTF-8
Host: example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "SubscriptionConfirmation",
```

```
{
  "MessageId" : "165545c9-2a5c-472c-8df2-7ff2be2b3b1b",
  "Token" : "2336412f37f...",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-west-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL included in this message.",
  "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-west-2:123456789012:MyTopic&Token=2336412f37f...",
  "Timestamp" : "2012-04-26T20:45:04.751Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEpH+...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

2. コードは、Amazon SNS メッセージを構成する名前と値のペアを読み取る HTTP POST リクエストの本文と content-type text/plain で、JSON ドキュメントを解析する必要があります。制御文字のエスケープ表現を ASCII 文字値に戻す変換 (\n を改行文字に変換するなど) を実行する JSON パーサーを使用します。[Jackson JSON プロセッサ](#)などの既存の JSON パーサーを使用するか、または独自のパーサーを記述できます。件名フィールドとメッセージフィールドで有効な JSON としてテキストを送信するためには、Amazon SNS は一部の制御文字を、JSON ドキュメントに含めることができるエスケープ表現に変換する必要があります。エンドポイントに送信される POST リクエストの本文で JSON ドキュメントを受信する場合、トピックに対して発行された元の件名およびメッセージとまったく同じ表現が必要なときは、エスケープした文字を元の文字値に戻す変換を実行する必要があります。これは、署名では署名対象の文字列の一部としてメッセージと件名が元の形式で使われるため、通知の署名を確認する場合に重要です。
3. Amazon SNS によって送信された通知、サブスクリプションの確認、またはサブスクリプション解除の確認メッセージの信頼性を確認する必要があります。エンドポイントは、Amazon SNS メッセージに含まれている情報を使用して署名を再作成し、署名を Amazon SNS がメッセージとともに送信した署名とマッチングすることで、メッセージの内容を確認できるようにします。メッセージの署名の確認の詳細については、「[Amazon SNS メッセージの署名を検証する](#)」を参照してください。
4. コードは、ヘッダーフィールド x-amz-sns-message-type で指定したタイプに基づいて、HTTP リクエストの本文に含まれている JSON ドキュメントを読み取り、メッセージを処理する必要があります。メッセージの 2 つの主要なタイプを処理するためのガイドラインを以下に示します。

SubscriptionConfirmation

SubscribeURL の値を読み取り、その URL にアクセスします。サブスクリプションを確認し、エンドポイントで通知の受信を開始するには、(URL に HTTP GET リクエストを送信するなどして) SubscribeURL URL にアクセスする必要があります。SubscribeURL の例については、前のステップの HTTP リクエストの例を参照してください。SubscriptionConfirmation メッセージの形式の詳細については、「[HTTP/HTTPS サブスクリプションの確認の JSON 形式](#)」を参照してください。URL にアクセスすると、以下の XML ドキュメントのような応答があります。ドキュメントは、ConfirmSubscriptionResult 要素内でエンドポイントのサブスクリプション ARN を返します。

```
<ConfirmSubscriptionResponse xmlns="http://sns.amazonaws.com/doc/2010-03-31/">
  <ConfirmSubscriptionResult>
    <SubscriptionArn>arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55</SubscriptionArn>
  </ConfirmSubscriptionResult>
  <ResponseMetadata>
    <RequestId>075ecce8-8dac-11e1-bf80-f781d96e9307</RequestId>
  </ResponseMetadata>
</ConfirmSubscriptionResponse>
```

SubscribeURL にアクセスする代わりに、[ConfirmSubscription](#) アクションを SubscriptionConfirmation メッセージの対応する値に設定した Token とともに使ってサブスクリプションを確認できます。トピックの所有者とサブスクリプションの所有者がエンドポイントのサブスクリプションを解除できるようにするには、AWS 署名とともに ConfirmSubscription アクションを呼び出します。

通知

Subject および Message の値を読み取り、トピックに発行された通知情報を取得します。

Notification メッセージの形式の詳細については、「[HTTP/HTTPS ヘッダー](#)」を参照してください。以下の HTTP POST リクエストは、エンドポイント example.com に送信される通知メッセージの例です。

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324
```

```
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-
west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96
Content-Length: 773
Content-Type: text/plain; charset=UTF-8
Host: example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "Notification",
  "MessageId" : "22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "My First Message",
  "Message" : "Hello world!",
  "Timestamp" : "2012-05-02T00:54:06.655Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEw6JRN...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/
SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96"
}
```

5. エンドポイントが Amazon SNS からの HTTP POST メッセージに適切なステータスコードで応答することを確認します。接続は約 15 秒でタイムアウトします。接続がタイムアウトする前にエンドポイントが応答しない場合、またはエンドポイントが 200~4xx の範囲外のステータスコードを返す場合、Amazon SNS はメッセージの配信に失敗したとみなします。
6. コードが Amazon SNS からのメッセージ配信の再試行を処理できることを確認します。Amazon SNS は、エンドポイントから正常な応答を受け取らない場合、そのメッセージの配信をもう一度試みます。これはサブスクリプションの確認メッセージを含むすべてのメッセージに適用されます。デフォルトでは、メッセージの最初の配信が失敗した場合、Amazon SNS は試行の失敗の間隔として 20 秒に設定された遅延時間で、最大 3 回再試行を試みます。

 Note

メッセージのリクエストは約 15 秒後にタイムアウトします。つまり、メッセージ配信の失敗がタイムアウトによって起きた場合、Amazon SNS は前回の配信の試行から約 35 秒後に再試行します。エンドポイントに別の配信ポリシーを設定できます。

Amazon SNS は `x-amz-sns-message-id` ヘッダーフィールドを使用して、Amazon SNS トピックにパブリッシュされた各メッセージを一意に識別します。受信メッセージとともに処理したメッセージ ID を比較することで、メッセージが再試行であるかどうかを判断することができます。

7. HTTPS エンドポイントをサブスクライブする場合、エンドポイントに、信頼された認定権限 (CA) からのサーバー証明書があることを確認します。Amazon SNS は、Amazon SNS によって信頼された CA によって署名されたサーバー証明書がある HTTPS エンドポイントにのみメッセージを送信します。
8. Amazon SNS メッセージを受信するために作成したコードをデプロイします。エンドポイントサブスクライブするときに、エンドポイントは少なくともサブスクリプションの確認メッセージを受信する準備ができていなければならない必要があります。

ステップ 2: Amazon SNS トピックに HTTP/HTTPS エンドポイントをサブスクライブする

トピックを通じて HTTP または HTTPS エンドポイントにメッセージを送信するには、エンドポイントを Amazon SNS トピックにサブスクライブする必要があります。エンドポイントを指定するには、その URL を使用します。トピックにサブスクライブするには、Amazon SNS コンソール、[sns-subscribe](#) コマンド、または [Subscribe](#) API アクションを使用できます。開始する前に、サブスクライブするエンドポイントの URL があり、ステップ 1 で説明したように、エンドポイントで確認メッセージや通知メッセージを受信する準備ができていなければならないことを確認します。

Amazon SNS コンソールを使って HTTP または HTTPS エンドポイントをトピックにサブスクライブするには

1. [Amazon SNS コンソール](#) にサインインします。
2. ナビゲーションパネルで、[サブスクリプション] を選択します。
3. [Create subscription] (サブスクリプションの作成) を選択します。
4. [Protocol] ドロップダウンリストで [HTTP] または [HTTPS] を選択します。
5. [エンドポイント] ボックスで、メッセージの送信先となるトピックのエンドポイントの URL をペーストし、[サブスクリプションの作成] を選択します。
6. 確認メッセージが表示されます。[閉じる] を選択してください。

新しいサブスクライプリプションの [サブスク립ション ID] に、PendingConfirmation と表示されます。サブスク립ションを確認すると、[サブスク립ション ID] にサブスク립ション ID が表示されます。

ステップ 3: Amazon SNS サブスク립ションを確認する

Amazon SNS サブスク립ションを確認する際、次の手順に従って、エンドポイントがメッセージを正しく受信できることを確認します。このプロセスでは、受信される確認メッセージを処理するようにエンドポイントを設定し、確認 URL の取得や、サブスク립ションの確認を行います。サブスク립ションは、設定に応じて自動または手動で確認できます。

1. エンドポイントを Amazon SNS トピックにサブスクライブすると、Amazon SNS はそのエンドポイントに確認メッセージを送信します。このメッセージには **SubscribeURL** が含まれており、これを使用してサブスク립ションを確認する必要があります。
2. エンドポイントは、Amazon SNS からの受信メッセージをリッスンするように設定する必要があります。確認メッセージが到着したら、メッセージから **SubscribeURL** を取得します。
3. **SubscribeURL** を取得すると、次の 2 つの方法のいずれかでサブスク립ションを確認できます。

- 自動確認 – エンドポイントでは、HTTP GET リクエストを **SubscribeURL** に送信することでサブスク립ションを自動的に確認できます。

この方法では、手動操作は必要ありません。

- 手動確認 – 自動確認が設定されていない場合は、確認メッセージから **SubscribeURL** をコピーして、ブラウザのアドレスバーに貼り付けます。

これにより、サブスク립ションを手動で確認できます。

4. Amazon SNS コンソールを使用してサブスク립ションステータスを確認することもできます。
 - a. [Amazon SNS コンソール](#) にサインインします。
 - b. ナビゲーションペインで [サブスク립ション] を選択します。
 - c. リストでサブスク립ションを探します。
 - 確認されると、**SubscriptionArn** が表示されます。
 - まだ確認されていない場合は、**PendingConfirmation** と表示されます。

ステップ 4: Amazon SNS サブスクリプションの配信ポリシーを設定する (オプション)

デフォルトでは、メッセージの最初の配信が失敗した場合、Amazon SNS は試行の失敗の間隔として 20 秒に設定された遅延時間で、最大 3 回再試行を試みます。[ステップ 1](#) で説明しているように、エンドポイントには再試行されたメッセージを処理できるコードが必要です。トピックまたはサブスクリプションで配信ポリシーを設定することで、Amazon SNS が失敗したメッセージを再試行する頻度と間隔を制御できます。DeliveryPolicy で HTTP/S 通知のコンテンツタイプを指定することもできます。詳細については、「[HTTP/S 配信ポリシーの作成](#)」を参照してください。

ステップ 5: Amazon SNS トピックに発行するアクセス許可をユーザーに付与する (オプション)

デフォルトでは、トピックの所有者にはトピックを発行するアクセス権限があります。他のユーザーまたはアプリケーションがトピックに発行できるようにするには、AWS Identity and Access Management (IAM) を使用してトピックに発行アクセス許可を付与する必要があります。Amazon SNS アクションに対するアクセス権限を IAM ユーザーに付与する方法の詳細については、「[Amazon SNS でのアイデンティティベースのポリシーを使用する](#)」を参照してください。

トピックへのアクセスは、以下の 2 つの方法でコントロールできます。

- IAM ユーザーまたはグループにポリシーを追加する。トピックへのアクセス権限をユーザーに付与する最も簡単な方法では、グループを作成し、適切なポリシーをグループに追加して、そのグループにユーザーを追加します。個々のユーザーに設定するポリシーを継続的に追跡するよりも、グループに対してユーザーを追加または削除する方がはるかに簡単です。
- トピックにポリシーを追加します。別の AWS アカウントにトピックへのアクセス権限を追加する場合、そのプリンシパルとして、アクセス権限を付与する AWS アカウント を持っているポリシーを追加することが唯一の方法です。

ほとんどの場合は、最初の方法 (ポリシーをグループに適用し、適切なユーザーをグループに追加または削除することでアクセス権限を管理する) を使用します。別のアカウントのユーザーにアクセス権限を付与する必要がある場合は、2 番目の方法を使用します。

IAM ユーザーまたはグループに次のポリシーを追加した場合、そのユーザーまたはそのグループのメンバーに、MyTopic トピックで sns:Publish アクションを実行する許可が付与されます。

```
{
```

```
"Statement": [{
  "Sid": "AllowPublishToMyTopic",
  "Effect": "Allow",
  "Action": "sns:Publish",
  "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
}]
}
```

以下の例のポリシーでは、トピックへのアクセス権限を別のアカウントに付与する方法を示しています。

 Note

アカウントのリソースへの別の AWS アカウント アクセスを許可すると、そのリソースへの管理者レベルのアクセス (ワイルドカードアクセス) アクセス許可を持つ IAM ユーザーにも付与されます。他のアカウントの他のすべての IAM ユーザーは、自動的にリソースへのアクセスが拒否されます。その内の特定の IAM ユーザーにリソース AWS アカウント へのアクセスを許可する場合、アカウントまたは管理者レベルのアクセス権を持つ IAM ユーザーは、リソースのアクセス許可をそれらの IAM ユーザーに委任する必要があります。クロスアカウントの委任の詳細については、『IAM ガイドの使用』の「[Enabling Cross-Account Access](#)」を参照してください。

アカウント 123456789012 の「マイトピック」トピックに次のポリシーを追加した場合、そのトピックで `sns:Publish` アクションを実行するアクセス権限をアカウント 111122223333 に付与したことになります。

```
{
  "Statement": [{
    "Sid": "Allow-publish-to-topic",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic"
  }]
}
```

ステップ 6: HTTP/HTTPS エンドポイントに Amazon SNS メッセージを送信する

トピックに発行することで、トピックのサブスクリプションにメッセージを送信できます。トピックに発行するには、Amazon SNS コンソール、[sns-publish](#) CLI コマンド、または [Publish](#) API アクションを使用できます。

[ステップ 1](#) を実行した場合、エンドポイントでデプロイしたコードで通知を処理できます。

Amazon SNS コンソールを使用してトピックに発行するには

1. トピックに発行するアクセス許可を持つ AWS アカウント または IAM ユーザーの認証情報を使用して、にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/sns/> で Amazon SNS コンソールを開きます。
2. ナビゲーションパネルで、[トピック] を選択し、トピックを選択します。
3. [メッセージの発行] ボタンをクリックします。
4. [件名] ボックスに件名 (「**Testing publish to my endpoint**」など) を入力します。
5. [メッセージ] ボックスにテキスト (「**Hello world!**」など) を入力し、[メッセージの発行] を選択します。

「Your message has been successfully published.」というメッセージが表示されます。

Amazon SNS メッセージの署名を検証する

Amazon SNS はメッセージ署名を使用して、HTTP エンドポイントに送信されたメッセージの信頼性を確認します。メッセージの整合性を確保し、なりすましを防ぐには、Amazon SNS メッセージを処理する前に署名を検証する必要があります。

Amazon SNS 署名の検証が必要になるタイミング

次のシナリオでは、Amazon SNS メッセージの署名を検証する必要があります。

- Amazon SNS が HTTP(S) エンドポイントに通知メッセージを送信するとき。
- Amazon SNS が [Subscribe](#) API 呼び出しや [Unsubscribe](#) API コールにエンドポイントに確認メッセージを送信するとき。

Amazon SNS は次の 2 つの署名バージョンをサポートしています。

- SignatureVersion1 – メッセージの SHA1 ハッシュを使用します。

- SignatureVersion2 – メッセージの SHA256 ハッシュを使用します。このバージョンはセキュリティを強化するため、推奨されるオプションです。

SNS メッセージの署名を正しく検証するには、次のベストプラクティスに従ってください。

- 不正な傍受攻撃を防ぐために、常に HTTPS を使用して署名証明書を取得します。
- 証明書が Amazon SNS によって発行されていることを確認します。
- 証明書の信頼チェーンが有効であることを確認します。
- 証明書は SNS 署名付き URL から取得する必要があります。
- メッセージで提供された証明書を、検証せずに信頼してはいけません。
- なりすましを防ぐために、予期しない TopicArn を含むメッセージは拒否します。
- Amazon SNS AWS SDKs は、組み込みの検証ロジックを提供するため、誤実装のリスクが軽減されます。

Amazon SNS トピックでメッセージ署名バージョンを設定する

Amazon SNS トピックでメッセージ署名バージョンを設定すると、メッセージ検証プロセスのセキュリティと互換性を強化できます。

SignatureVersion1 (SHA1) と SignatureVersion2 (SHA256) のいずれかを選択して、メッセージの署名に使用されるハッシュアルゴリズムを制御します。Amazon SNS トピックのデフォルトは、SignatureVersion1 です。この属性は、[SetTopicAttributes](#) API アクションを使用して設定できます。

次の例では、AWS CLIを使用してトピック属性 SignatureVersion を設定しています。

```
aws sns set-topic-attributes \  
  --topic-arn arn:aws:sns:us-east-2:123456789012:MyTopic \  
  --attribute-name SignatureVersion \  
  --attribute-value 2
```

HTTP クエリベースのリクエストの使用時に Amazon SNS メッセージの署名を確認する

HTTP クエリベースのリクエストを使用する際に Amazon SNS メッセージの署名を検証すると、メッセージの信頼性と整合性が保証されます。このプロセスでは、メッセージが Amazon SNS から

発信され、転送中に改ざんされていないことを確認します。メッセージの解析、署名用の正しい文字列の作成、信頼できるパブリックキーによる署名の検証を行うことで、なりすましや不正なメッセージ変更からシステムを保護できます。

1. Amazon SNS から送信された HTTP POST リクエストの本文で、JSON ドキュメントからキーと値のペアを抽出します。これらのフィールドは、署名文字列を作成するために必要です。

- Message
- Subject (存在する場合)
- MessageId
- Timestamp
- TopicArn
- Type

例えば、次のようになります。

```
MESSAGE_FILE="message.json"
FIELDS=("Message" "MessageId" "Subject" "Timestamp" "TopicArn" "Type")
```

Note

いずれかのフィールドにエスケープ文字 (\n など) が含まれている場合は、それらの文字を元の形式に変換して、正確な一致が確実に行われるようにします。

2. Amazon SNS メッセージで SigningCertURL フィールドを見つけます。この証明書には、メッセージ署名の検証に必要なパブリックキーが含まれています。例えば、次のようになります。

```
SIGNING_CERT_URL=$(jq -r '.SigningCertURL' "$MESSAGE_FILE")
```

3. SigningCertURL が信頼された AWS ドメイン (など) からのものであることを確認します <https://sns.us-east-1.amazonaws.com>。セキュリティ上の理由から、AWS ドメイン外の URLs。
4. 指定された URL から X.509 証明書をダウンロードします。例えば、次のようになります。

```
curl -s "$SIGNING_CERT_URL" -o signing_cert.pem
```

- ダウンロードした X.509 証明書からパブリックキーを抽出します。パブリックキーを使用すると、メッセージの署名を復号化して、その整合性を検証できます。例えば、次のようになります。

```
openssl x509 -pubkey -noout -in signing_cert.pem > public_key.pem
```

- メッセージタイプが異なると、署名文字列に含めるキーと値のペアも異なります。メッセージタイプ (Amazon SNS メッセージ内の Type フィールド) を特定して、含めるキーと値のペアを決定します。
 - 通知メッセージ – Message、MessageId、Subject (存在する場合)、Timestamp、TopicArn、および Type が含まれます。
 - SubscriptionConfirmation または UnsubscribeConfirmation メッセージ – Message、MessageId、SubscribeURL、Timestamp、Token、TopicArn、および Type が含まれます。
- Amazon SNS では、検証を行うために、署名文字列は厳格で固定されたフィールドの順序に従う必要があります。明示的な必須フィールドのみを含める必要があり、追加のフィールドを加えることはできません。Subject などのオプションフィールドは、そのフィールドがメッセージ内に存在する場合にのみ含める必要があり、必須フィールドの順序で定義された正確な位置に含める必要があります。例えば、次のようになります。

```
KeyNameOne\nValueOne\nKeyNameTwo\nValueTwo
```

Important

文字列の末尾に改行文字を追加しないでください。

- キーと値のペアをバイト順でソートし、並べます (キー名に基づくアルファベット順)。
- 次の形式の例を使用して、署名文字列を作成します。

```
STRING_TO_SIGN=""
for FIELD in "${FIELDS[@]}"; do
    VALUE=$(jq -r --arg field "$FIELD" '[$field]' "$MESSAGE_FILE")
    STRING_TO_SIGN+="$FIELD\n$VALUE"
    # Append a newline after each field except the last one
    if [[ "$FIELD" != "Type" ]]; then
        STRING_TO_SIGN+="\n"
    fi
done
```

```
done
```

通知メッセージの例:

```
Message
My Test Message
MessageId
4d4dc071-ddbf-465d-bba8-08f81c89da64
Subject
My subject
Timestamp
2019-01-31T04:37:04.321Z
TopicArn
arn:aws:sns:us-east-2:123456789012:s4-MySNSTopic-1G1WEFC0XTC0P
Type
Notification
```

SubscriptionConfirmation の例:

```
Message
Please confirm your subscription
MessageId
3d891288-136d-417f-bc05-901c108273ee
SubscribeURL
https://sns.us-east-2.amazonaws.com/...
Timestamp
2024-01-01T00:00:00.000Z
Token
abc123...
TopicArn
arn:aws:sns:us-east-2:123456789012:MyTopic
Type
SubscriptionConfirmation
```

10. メッセージ内の Signature フィールドは Base64 でエンコードされています。未加工のバイナリ形式と、派生したハッシュを比較するために、このフィールドをデコードする必要があります。例えば、次のようになります。

```
SIGNATURE=$(jq -r '.Signature' "$MESSAGE_FILE")
echo "$SIGNATURE" | base64 -d > signature.bin
```

11. SignatureVersion フィールドを使用してハッシュアルゴリズムを選択します。

- SignatureVersion1 の場合 SHA1 を使用します (例: -sha1)。
- SignatureVersion2 の場合 SHA256 を使用します (例: -sha256)。

12. Amazon SNS メッセージの信頼性を確認するには、作成された文字列のハッシュを生成し、パブリックキーを使用して署名を検証します。

```
openssl dgst -sha256 -verify public_key.pem -signature signature.bin <<<
"$STRING_TO_SIGN"
```

署名が有効な場合、出力は Verified OK になります。それ以外の場合、出力は Verification Failure になります。

エラー処理を使用したスクリプトの例

次のスクリプト例では、検証プロセスを自動化します。

```
#!/bin/bash

# Path to the local message file
MESSAGE_FILE="message.json"

# Extract the SigningCertURL and Signature from the message
SIGNING_CERT_URL=$(jq -r '.SigningCertURL' "$MESSAGE_FILE")
SIGNATURE=$(jq -r '.Signature' "$MESSAGE_FILE")

# Fetch the X.509 certificate
curl -s "$SIGNING_CERT_URL" -o signing_cert.pem

# Extract the public key from the certificate
openssl x509 -pubkey -noout -in signing_cert.pem > public_key.pem

# Define the fields to include in the string to sign
FIELDS=("Message" "MessageId" "Subject" "Timestamp" "TopicArn" "Type")

# Initialize the string to sign
STRING_TO_SIGN=""

# Iterate over the fields to construct the string to sign
for FIELD in "${FIELDS[@]}; do
    VALUE=$(jq -r --arg field "$FIELD" '[$field]' "$MESSAGE_FILE")
    STRING_TO_SIGN+="$FIELD\n$VALUE"
```

```
# Append a newline after each field except the last one
if [[ "$FIELD" != "Type" ]]; then
    STRING_TO_SIGN+="\n"
fi
done

# Verify the signature
echo -e "$STRING_TO_SIGN" | openssl dgst -sha256 -verify public_key.pem -signature
<(echo "$SIGNATURE" | base64 -d)
```

Amazon SNS メッセージ形式を解析する

Amazon SNS が HTTP/HTTPS エンドポイントにメッセージを送信するとき、HTTP ヘッダーと JSON メッセージ本文の両方が含まれます。これらのメッセージは、メッセージタイプ、トピック ARN、タイムスタンプ、デジタル署名などのメタデータを含む構造化形式に従います。Amazon SNS メッセージを正しく解析することで、メッセージがサブスクリプションの確認、通知、またはサブスクリプション解除の確認であるかどうかを判断し、関連するデータを抽出し、署名検証を使用して真正性を検証できます。

HTTP/HTTPS ヘッダー

Amazon SNS がサブスクリプションの確認、通知、またはサブスクリプション解除の確認メッセージを HTTP/HTTPS エンドポイントに送信するときは、Amazon SNS 固有の多くのヘッダー値とともに POST メッセージを送信します。ヘッダー値を使用すると、JSON メッセージ本文を解析して Type 値を読み取ることなく、メッセージタイプの識別などのタスクを実行できます。デフォルトでは、Amazon SNS は、Content-Type が text/plain; charset=UTF-8 に設定されている HTTP/S エンドポイントにすべての通知を送信します。text/plain (デフォルト) 以外の Content-Type を選択するには、「[HTTP/S 配信ポリシーの作成](#)」の「headerContentType」を参照してください。

x-amz-sns-message-type

メッセージのタイプ。指定できる値は、SubscriptionConfirmation、Notification および UnsubscribeConfirmation です。

x-amz-sns-message-id

発行するメッセージごとの汎用一意識別子 (UUID)。再試行間に Amazon SNS が再送信する通知の場合、元のメッセージのメッセージ ID が使用されます。

x-amz-sns-topic-arn

このメッセージの発行先トピックの Amazon リソースネーム (ARN)。

x-amz-sns-subscription-arn

このエンドポイントへのサブスクリプションの ARN。

次の HTTP POST ヘッダーは、HTTP エンドポイントへの Notification メッセージのヘッダーの例です。

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 1336
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent
```

HTTP/HTTPS サブスクリプションの確認の JSON 形式

HTTP/HTTPS エンドポイントをサブスクライブすると、Amazon SNS は HTTP/HTTPS エンドポイントにサブスクリプションの確認メッセージを送信します。このメッセージには、サブスクリプションを確認するためにアクセスする必要がある `SubscribeURL` 値が含まれています (または、Token 値を [ConfirmSubscription](#) で使用できます)。

Note

サブスクリプションが確認されるまで、Amazon SNS はこのエンドポイントに通知を送信しません。

サブスクリプションの確認メッセージは、以下の名前と値のペアを持つ JSON ドキュメントを含むメッセージ本文がある POST メッセージです。

Type

メッセージのタイプ。サブスクリプションの確認の場合、タイプは `SubscriptionConfirmation` です。

MessageId

発行するメッセージごとの汎用一意識別子 (UUID)。再試行間に Amazon SNS が再送信するメッセージの場合、元のメッセージのメッセージ ID が使用されます。

Token

サブスクリプションを確認するために [ConfirmSubscription](#) アクションで利用できる値。または、SubscribeURL にアクセスしても構いません。

TopicArn

このエンドポイントがサブスクライブしているトピックの Amazon リソースネーム (ARN)。

Message

メッセージについて説明する文字列。サブスクリプションの確認の場合、この文字列は以下のようになります。

```
You have chosen to subscribe to the topic arn:aws:sns:us-east-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL included in this message.
```

SubscribeURL

サブスクリプションを確認するためにアクセスする必要がある URL。または、代わりに Token を [ConfirmSubscription](#) アクションで使用してサブスクリプションを確認することもできます。

Timestamp

サブスクリプションの確認が送信された日時 (GMT)

SignatureVersion

使用される Amazon SNS 署名のバージョン

- SignatureVersion が 1 である場合、Signature は Message、MessageId、Type、Timestamp、TopicArn の各値の Base64 でエンコードされた SHA1withRSA シグネチャです。

- `SignatureVersion` が 2 である場合、`Signature` は `Message`、`MessageId`、`Type`、`Timestamp`、`TopicArn` の各値の Base64 でエンコードされた SHA256withRSA シグネチャです。

Signature

`Message`、`MessageId`、`Type`、`Timestamp`、`TopicArn` の各値の Base64 でエンコードされた SHA1withRSA または SHA256withRSA シグネチャ。

SigningCertURL

メッセージの署名に使用された証明書の URL

次の HTTP POST メッセージは、HTTP エンドポイントへの `SubscriptionConfirmation` メッセージの例です。

```
POST / HTTP/1.1
x-amz-sns-message-type: SubscriptionConfirmation
x-amz-sns-message-id: 165545c9-2a5c-472c-8df2-7ff2be2b3b1b
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
Content-Length: 1336
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "SubscriptionConfirmation",
  "MessageId" : "165545c9-2a5c-472c-8df2-7ff2be2b3b1b",
  "Token" : "2336412f37...",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Message" : "You have chosen to subscribe to the topic arn:aws:sns:us-
west-2:123456789012:MyTopic.\nTo confirm the subscription, visit the SubscribeURL
included in this message.",
  "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-
west-2:123456789012:MyTopic&Token=2336412f37...",
  "Timestamp" : "2012-04-26T20:45:04.751Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEpH
+DcEwjAPg809mY8dReBSwksfg2S7WKQcickNKWLQjwu6A4VbeS0QHVCkhRS7fUQvi2egU3N858fiTDN6bkk0xYDVrY0Ad8L
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem"
```



```
}
```

HTTP/HTTPS 通知の JSON 形式

Amazon SNS がサブスクライブしている HTTP または HTTPS エンドポイントに通知を送信すると、エンドポイントに送信される POST メッセージには、以下の名前と値のペアを持つ JSON ドキュメントを含むメッセージ本文があります。

Type

メッセージのタイプ。通知の場合、タイプは Notification です。

MessageId

発行するメッセージごとの汎用一意識別子 (UUID)。再試行間に Amazon SNS が再送信する通知の場合、元のメッセージのメッセージ ID が使用されます。

TopicArn

このメッセージの発行先トピックの Amazon リソースネーム (ARN)。

Subject

通知をトピックに発行したときに指定した Subject パラメータ。

Note

このパラメータはオプションです。Subject を指定しなかった場合、この名前と値のペアは、この JSON ドキュメントに表示されません。

Message

通知をトピックに発行したときに指定した Message 値。

Timestamp

通知が公開されたときの日時 (GMT)。

SignatureVersion

使用される Amazon SNS 署名のバージョン

- SignatureVersion が 1 である場合、Signature は Message、MessageId、Subject (存在する場合)、Type、Timestamp、TopicArn の各値の Base64 でエンコードされた SHA1withRSA シグネチャです。
- SignatureVersion が 2 である場合、Signature は Message、MessageId、Subject (存在する場合)、Type、Timestamp、TopicArn の各値の Base64 でエンコードされた SHA256withRSA シグネチャです。

Signature

Message、MessageId、Subject (存在する場合)、Type、Timestamp、TopicArn の各値の Base64 でエンコードされた SHA1withRSA または SHA256withRSA シグネチャ。

SigningCertURL

メッセージの署名に使用された証明書の URL

UnsubscribeURL

このトピックからエンドポイントのサブスクリプションを解除するために使用できる URL。この URL にアクセスすると、Amazon SNS はエンドポイントのサブスクリプションを解除し、このエンドポイントへの通知の送信を停止します。

次の HTTP POST メッセージは、HTTP エンドポイントへの Notification メッセージの例です。

```
POST / HTTP/1.1
x-amz-sns-message-type: Notification
x-amz-sns-message-id: 22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96
Content-Length: 773
Content-Type: text/plain; charset=UTF-8
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "Notification",
  "MessageId" : "22b80b92-fdea-4c2c-8f9d-bdfb0c7bf324",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Subject" : "My First Message",
  "Message" : "Hello world!",
  "Timestamp" : "2012-05-02T00:54:06.655Z",
```

```
{
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEw6JRN...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-
f3ecfb7224c7233fe7bb5f59f96de52f.pem",
  "UnsubscribeURL" : "https://sns.us-west-2.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-
west-2:123456789012:MyTopic:c9135db0-26c4-47ec-8998-413945fb5a96"
}
```

HTTP/HTTPS サブスクリプションの解除の JSON 形式

HTTP/HTTPS エンドポイントのサブスクリプションがトピックから解除されると、Amazon SNS はエンドポイントにサブスクリプションの確認メッセージを送信します。

サブスクリプション解除の確認メッセージは、次の名前と値のペアを持つ JSON ドキュメントを含むメッセージ本文がある POST メッセージです。

Type

メッセージのタイプ。サブスクリプション解除の確認の場合、タイプは `UnsubscribeConfirmation` です。

MessageId

発行するメッセージごとの汎用一意識別子 (UUID)。再試行間に Amazon SNS が再送信するメッセージの場合、元のメッセージのメッセージ ID が使用されます。

Token

サブスクリプションを再確認するために [ConfirmSubscription](#) アクションで使用できる値。または、SubscribeURL にアクセスしても構いません。

TopicArn

このエンドポイントがサブスクリプションを解除されたトピックの Amazon リソースネーム (ARN)。

Message

メッセージについて説明する文字列。サブスクリプション解除の確認の場合、この文字列は次のようになります。

```
You have chosen to deactivate subscription arn:aws:sns:us-
east-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55.\n
To cancel this
```

operation and restore the subscription, visit the SubscribeURL included in this message.

SubscribeURL

サブスクリプションを再確認するためにアクセスする必要がある URL。または、代わりに Token を [ConfirmSubscription](#) アクションで使用してサブスクリプションを再確認することもできます。

Timestamp

サブスクリプション解除の確認が送信された日時 (GMT)。

SignatureVersion

使用される Amazon SNS 署名のバージョン

- SignatureVersion が 1 である場合、Signature は Message、MessageId、Type、Timestamp、TopicArn の各値の Base64 でエンコードされた SHA1withRSA シグネチャです。
- SignatureVersion が 2 である場合、Signature は Message、MessageId、Type、Timestamp、TopicArn の各値の Base64 でエンコードされた SHA256withRSA シグネチャです。

Signature

Message、MessageId、Type、Timestamp、TopicArn の各値の Base64 でエンコードされた SHA1withRSA または SHA256withRSA シグネチャ。

SigningCertURL

メッセージの署名に使用された証明書の URL

次の HTTP POST メッセージは、HTTP エンドポイントへの UnsubscribeConfirmation メッセージの例です。

```
POST / HTTP/1.1
x-amz-sns-message-type: UnsubscribeConfirmation
x-amz-sns-message-id: 47138184-6831-46b8-8f7c-afc488602d7d
x-amz-sns-topic-arn: arn:aws:sns:us-west-2:123456789012:MyTopic
x-amz-sns-subscription-arn: arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55
Content-Length: 1399
Content-Type: text/plain; charset=UTF-8
```

```
Host: myhost.example.com
Connection: Keep-Alive
User-Agent: Amazon Simple Notification Service Agent

{
  "Type" : "UnsubscribeConfirmation",
  "MessageId" : "47138184-6831-46b8-8f7c-afc488602d7d",
  "Token" : "2336412f37...",
  "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
  "Message" : "You have chosen to deactivate subscription arn:aws:sns:us-west-2:123456789012:MyTopic:2bcfbf39-05c3-41de-beaa-fcfcc21c8f55.\nTo cancel this operation and restore the subscription, visit the SubscribeURL included in this message.",
  "SubscribeURL" : "https://sns.us-west-2.amazonaws.com/?Action=ConfirmSubscription&TopicArn=arn:aws:sns:us-west-2:123456789012:MyTopic&Token=2336412f37fb6...",
  "Timestamp" : "2012-04-26T20:06:41.581Z",
  "SignatureVersion" : "1",
  "Signature" : "EXAMPLEHXgJm...",
  "SigningCertURL" : "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-f3ecfb7224c7233fe7bb5f59f96de52f.pem"
}
```

SetSubscriptionAttributes 配信ポリシーの JSON 形式

SetSubscriptionAttributes アクションにリクエストを送信し、AttributeName パラメータを DeliveryPolicy の値に設定する場合、AttributeValue パラメータの値は有効な JSON オブジェクトである必要があります。例えば、次の例では、配信ポリシーを合計 5 回の再試行に設定します。

```
http://sns.us-east-2.amazonaws.com/
?Action=SetSubscriptionAttributes
&SubscriptionArn=arn%3Aaws%3Asns%3Aus-east-2%3A123456789012%3AMy-Topic%3A80289ba6-0fd4-4079-afb4-ce8c8260f0ca
&AttributeName=DeliveryPolicy
&AttributeValue={"healthyRetryPolicy":{"numRetries":5}}
...
```

AttributeValue パラメータの値には、次の JSON 形式を使用します。

```
{
  "healthyRetryPolicy" : {
```

```

    "minDelayTarget" : int,
    "maxDelayTarget" : int,
    "numRetries" : int,
    "numMaxDelayRetries" : int,
    "backoffFunction" : "linear|arithmetic|geometric|exponential"
  },
  "throttlePolicy" : {
    "maxReceivesPerSecond" : int
  },
  "requestPolicy" : {
    "headerContentType" : "text/plain | application/json | application/xml"
  }
}

```

SetSubscriptionAttribute アクションの詳細については、「Amazon Simple Notification Service API リファレンス」の「[SetSubscriptionAttributes](#)」を参照してください。サポートされている HTTP コンテンツタイプヘッダーの詳細については、「[HTTP/S 配信ポリシーの作成](#)」を参照してください。

SetTopicAttributes 配信ポリシーの JSON 形式

SetTopicAttributes アクションにリクエストを送信し、AttributeName パラメータを DeliveryPolicy の値に設定する場合、AttributeValue パラメータの値は有効な JSON オブジェクトである必要があります。例えば、次の例では、配信ポリシーを合計 5 回の再試行に設定します。

```

http://sns.us-east-2.amazonaws.com/
?Action=SetTopicAttributes
&TopicArn=arn%3Aaws%3Asns%3Aus-east-2%3A123456789012%3AMy-Topic
&AttributeName=DeliveryPolicy
&AttributeValue={"http":{"defaultHealthyRetryPolicy":{"numRetries":5}}}
...

```

AttributeValue パラメータの値には、次の JSON 形式を使用します。

```

{
  "http" : {
    "defaultHealthyRetryPolicy" : {
      "minDelayTarget": int,
      "maxDelayTarget": int,
      "numRetries": int,

```

```
        "numMaxDelayRetries": int,
        "backoffFunction": "linear|arithmetic|geometric|exponential"
    },
    "disableSubscriptionOverrides" : Boolean,
    "defaultThrottlePolicy" : {
        "maxReceivesPerSecond" : int
    },
    "defaultRequestPolicy" : {
        "headerContentType" : "text/plain | application/json | application/xml"
    }
}
```

SetTopicAttribute アクションの詳細については、「Amazon Simple Notification Service API リファレンス」の[SetTopicAttributes](#)にアクセスしてください。サポートされている HTTP コンテンツタイプヘッダーの詳細については、「[HTTP/S 配信ポリシーの作成](#)」を参照してください。

Amazon SNS イベントを AWS Event Fork Pipelines にファンアウトする

イベントのアーカイブと分析のために、Amazon SNS は Amazon Data Firehose とのネイティブ統合の使用を推奨するようになりました。Firehose 配信ストリームを SNS トピックにサブスクライブできます。これにより、Amazon Simple Storage Service (Amazon S3) バケット、Amazon Redshift テーブル、Amazon OpenSearch Service (OpenSearch Service) などのアーカイブと分析エンドポイントへ通知を送信することができます。Firehose 配信ストリームで Amazon SNS を使用するのには、フルマネージド型のコードレスソリューションであり、AWS Lambda 関数を使用する必要はありません。詳細については、「[Firehose 配信ストリームへのファンアウト](#)」を参照してください。

Amazon SNS で構築したイベント駆動型アプリケーションで受信者サービスを使用し、発行者サービスでトリガーされたイベントに応答して自動的に作業を実行できます。このアーキテクチャパターンにより、サービスの再利用性、相互運用性、およびスケーラビリティを高めることができます。ただし、イベントのストレージ、バックアップ、検索、分析、再生などの一般的なイベント処理要件に対応するパイプラインにイベント処理を分岐させることは多大な労力を要する場合があります。

イベント駆動型アプリケーションの開発を加速するには、Event Fork Pipelines を搭載した AWS イベント処理パイプラインを Amazon SNS トピックにサブスクライブできます。AWS Event Fork

Pipelines は、[AWS Serverless Application Model](#) (AWS SAM) に基づくオープンソースの[ネストされたアプリケーションのスイート](#)です。[AWS Event Fork Pipelines スイート](#) (カスタム IAM ロールまたはリソースポリシーを作成する Show アプリケーションを選択) から AWS アカウントに直接デプロイできます。

AWS Event Fork Pipelines のユースケースについては、「」を参照してください[Amazon SNS Event Fork Pipelines サンプルアプリケーションをデプロイしてテストする](#)。

トピック

- [AWS Event Fork Pipelines の仕組み](#)
- [Event AWS Fork Pipelines のデプロイ](#)
- [Amazon SNS Event Fork Pipelines サンプルアプリケーションをデプロイしてテストする](#)
- [AWS Event Fork Pipelines を Amazon SNS トピックにサブスクライブする](#)

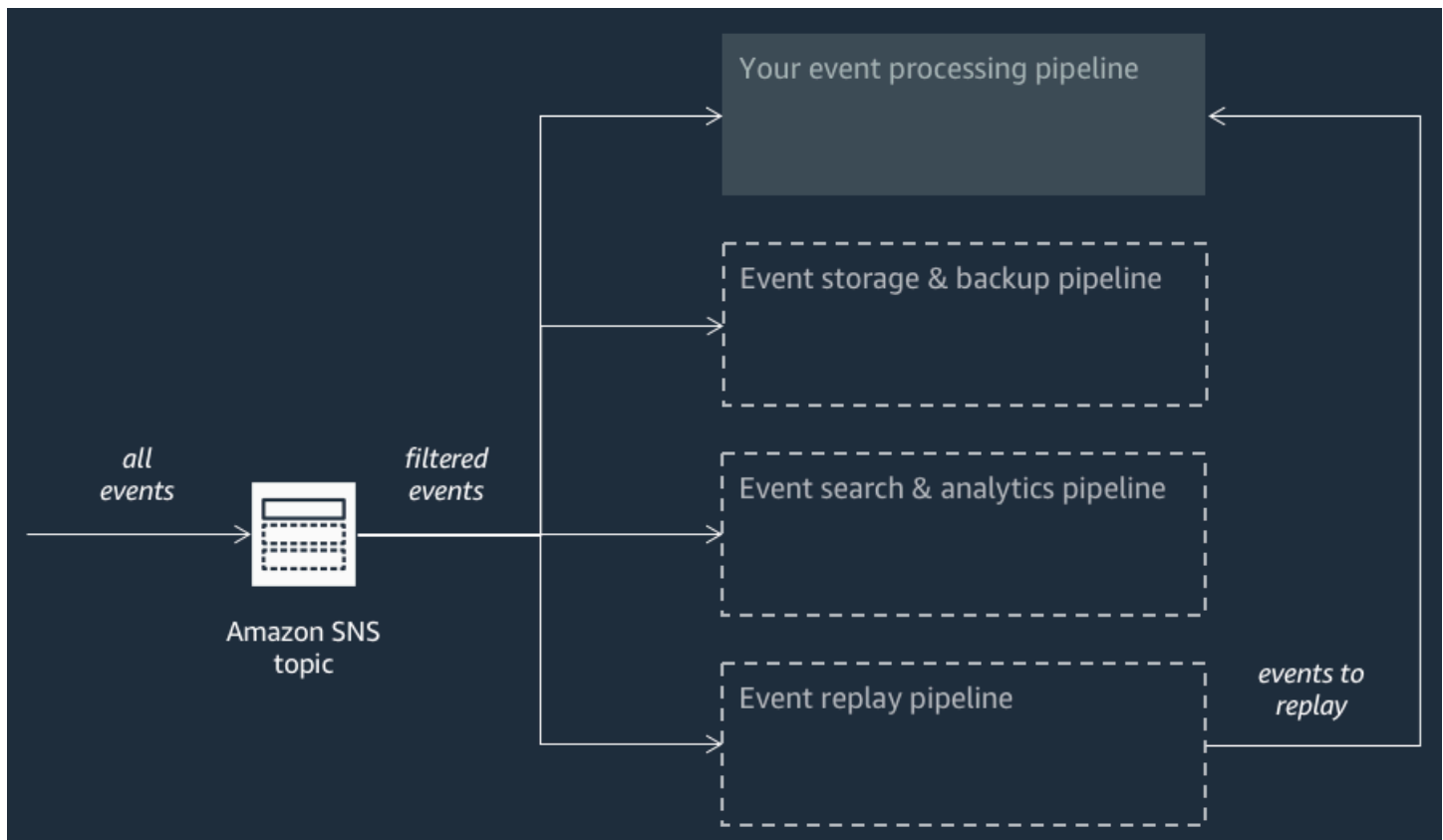
AWS Event Fork Pipelines の仕組み

AWS Event Fork Pipelines はサーバーレス設計パターンです。ただし、AWS SAM に基づくネストされたサーバーレスアプリケーションのスイートでもあります (イベント駆動型プラットフォームを強化 AWS アカウント するために (AWS SAR) から AWS Serverless Application Repository に直接デプロイできます)。アーキテクチャの必要に応じて、これらのネストされたアプリケーションを個別にデプロイできます。

トピック

- [イベントのストレージおよびバックアップパイプライン](#)
- [イベントの検索および分析パイプライン](#)
- [イベントの再生パイプライン](#)

次の図は、3 つのネストされたアプリケーションで補完された AWS Event Fork Pipelines アプリケーションを示しています。アーキテクチャの必要に応じて、AWS Event Fork Pipelines スイートから任意のパイプラインを AWS SAR に個別にデプロイできます。



各パイプラインは、同じ Amazon SNS トピックにサブスクライブされるため、このトピックに発行された複数のイベントを並列処理できます。各パイプラインは独立しており、独自の[サブスクリプションフィルターポリシー](#)を設定できます。これにより、パイプラインでは、トピックに発行されたすべてのイベントではなく、関心のあるイベントのサブセットに絞って処理できます。

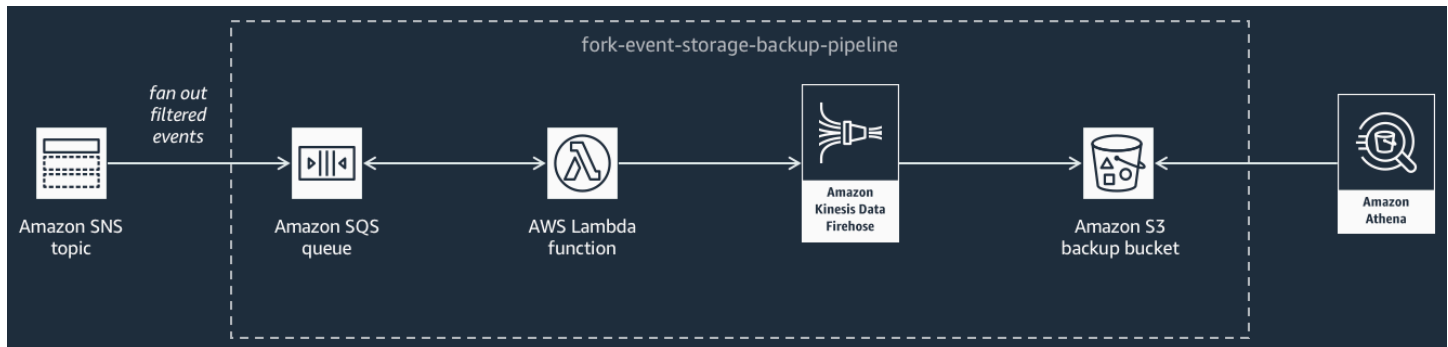
Note

通常のイベント処理パイプライン (Amazon SNS トピックに既にサブスクライブされている可能性がある) と一緒に 3 つの AWS Event Fork Pipelines を配置するため、既存のワークロードで AWS Event Fork Pipelines を利用するために、現在のメッセージパブリッシャーの一部を変更する必要はありません。

イベントのストレージおよびバックアップパイプライン

次の図は、[イベントのストレージおよびバックアップパイプライン](#)を示しています。このパイプラインを Amazon SNS トピックにサブスクライブして、システムを通過するイベントを自動的にバックアップできます。

このパイプラインは、Amazon SNS トピックによって配信されるイベントをバッファする Amazon SQS Amazon SNS キュー、キュー内のこれらのイベントを自動的にポーリングしてストリームにプッシュする AWS Lambda 関数、およびストリームによってロードされたイベントを永続的にバックアップする Amazon S3 バケットで構成されます。

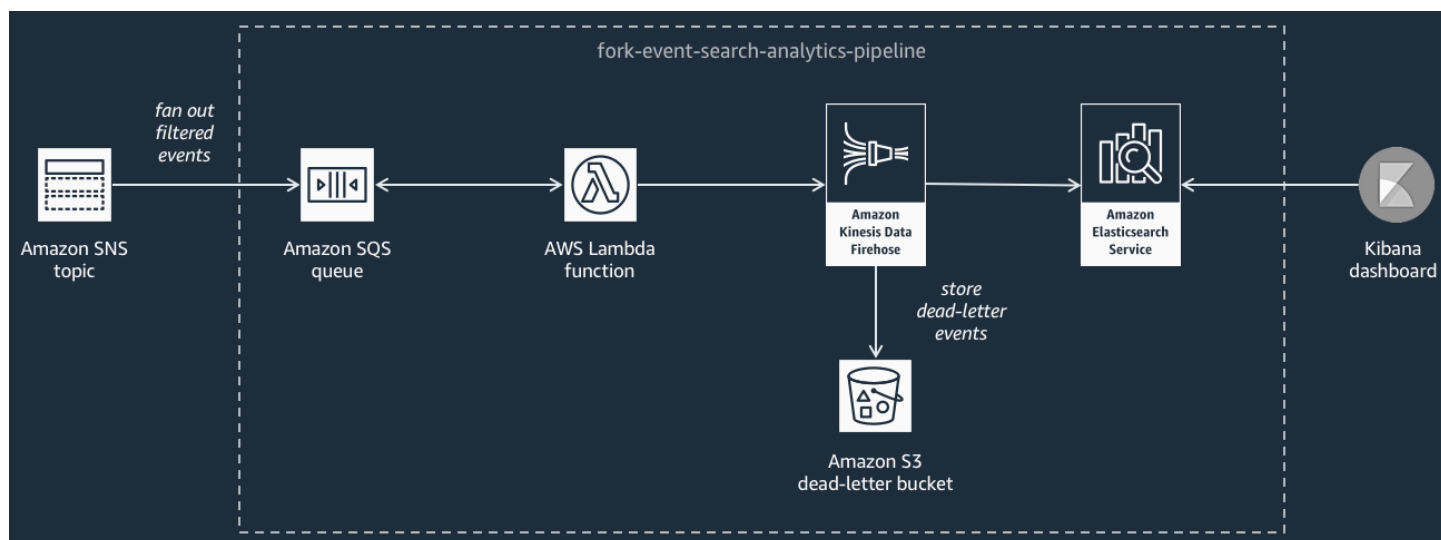


Firehose ストリームの動作を微調整するには、イベントをバケット内にロードする前に、イベントをバッファ処理、変換、および圧縮するようにストリームを設定できます。イベントがロードされたら、Amazon Athena で標準の SQL クエリを使用し、バケットに対してクエリを実行できます。既存の Amazon S3 バケットを再利用したり、新しいバケットを作成したりするようにパイプラインを設定することもできます。

イベントの検索および分析パイプライン

次の図は、[イベントの検索および分析パイプライン](#)を示しています。このパイプラインを Amazon SNS トピックにサブスクライブして、システムを通過するイベントを検索ドメインでインデックス付けし、これらのイベントに対して分析を実行できます。

このパイプラインは、Amazon SNS トピックによって配信されるイベントをバッファする Amazon SQS Amazon SNS キュー、キューからイベントをポーリングしてストリームにプッシュする AWS Lambda 関数、Firehose ストリームによってロードされたイベントをインデックスする Amazon OpenSearch Service ドメイン、および検索ドメインでインデックス化できないデッドレターイベントを保存する Amazon S3 バケットで構成されます。



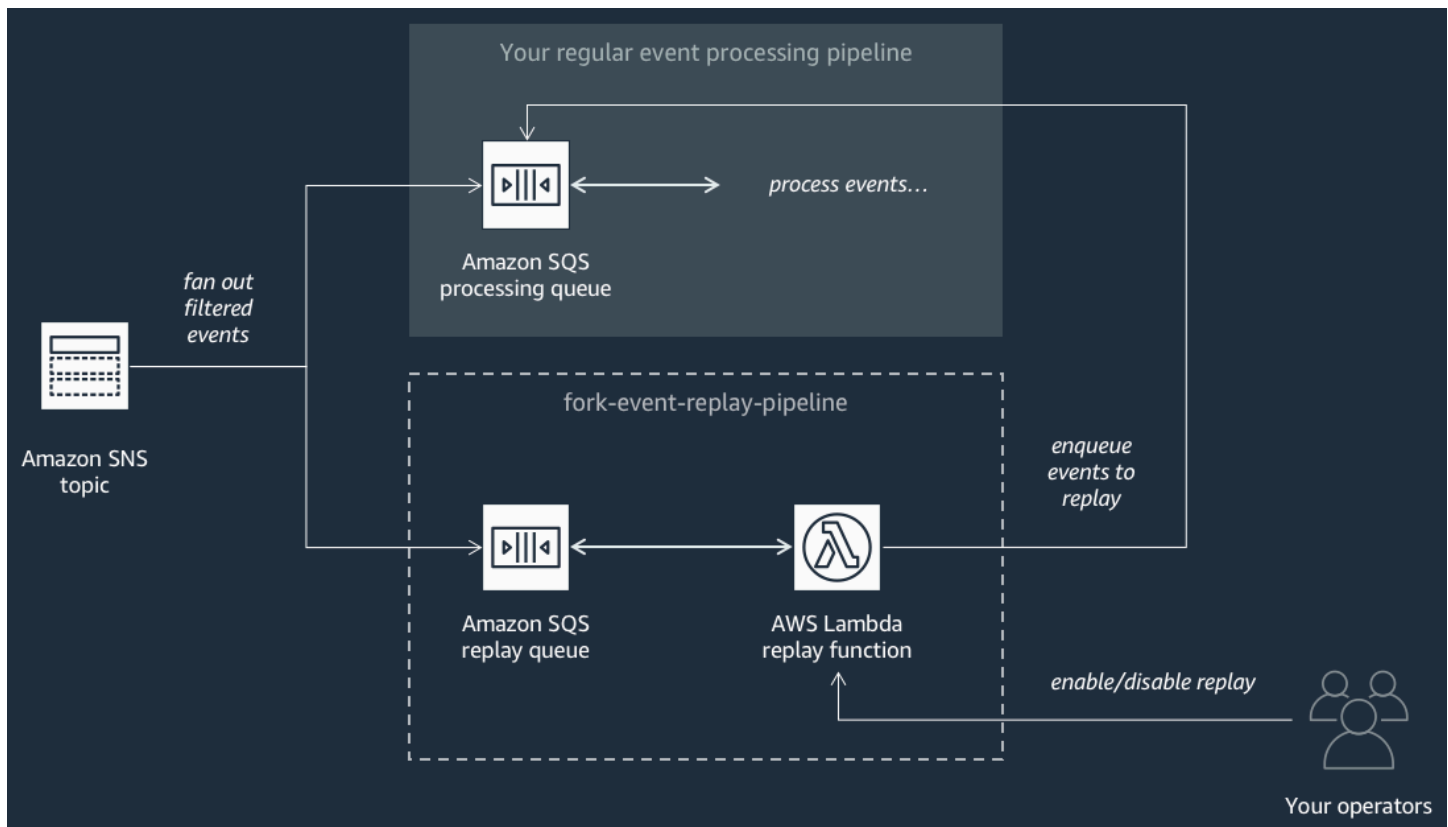
Firehose ストリームについてイベントのバッファ処理、変換、および圧縮を微調整するには、このパイプラインを設定できます。

パイプラインが 内の既存の OpenSearch ドメインを再利用するか、新しいドメイン AWS アカウント を作成するかを設定することもできます。イベントは検索ドメインでインデックス付けされるため、Kibana を使用してイベントに対して分析を実行し、リアルタイムでビジュアルダッシュボードを更新できます。

イベントの再生パイプライン

次の図は、[イベントの再生パイプライン](#)を示しています。過去 14 日間にシステムで処理されたイベントを記録するには (プラットフォームを障害から復旧する必要がある場合など)、このパイプラインを Amazon SNS トピックにサブスクライブしてイベントを再処理できます。

このパイプラインは、Amazon SQS トピックによって配信されるイベントを Amazon SNS キューと、キューからイベントをポーリングして通常のイベント処理パイプラインに再処理する AWS Lambda 関数で構成されます。この関数は、トピックにもサブスクライブされています。



Note

デフォルトでは、再生関数は無効になっており、イベントを再ルーティングしません。イベントを再処理する必要がある場合は、Amazon SQS 再生関数のイベントソースとして AWS Lambda 再生キューを有効にする必要があります。

Event AWS Fork Pipelines のデプロイ

[AWS Event Fork Pipelines スイート](#) (カスタム IAM ロールまたはリソースポリシーを作成する Show アプリを選択) は、でパブリックアプリケーションのグループとして使用できます。ここから AWS Serverless Application Repository、[AWS Lambda コンソール](#)を使用して手動でデプロイしてテストできます。AWS Lambda コンソールを使用してパイプラインをデプロイする方法については、「」を参照してください[AWS Event Fork Pipelines を Amazon SNS トピックにサブスクライブする](#)。

本稼働シナリオでは、アプリケーション全体の SAM テンプレートに AWS Event Fork Pipelines AWS を埋め込むことをお勧めします。ネストされたアプリケーション機能を使用すると、リソースを AWS SAM テンプレートに追加[AWS::Serverless::Application](#)し、ネストされたアプリケーションの AWS SAR ApplicationIdと SemanticVersion を参照できます。

たとえば、次の YAML スニペットを AWS SAM テンプレートの Resources セクションに追加することで、イベントストレージとバックアップパイプラインをネストされたアプリケーションとして使用できます。

```
Backup:
  Type: AWS::Serverless::Application
  Properties:
    Location:
      ApplicationId: arn:aws:serverlessrepo:us-east-2:123456789012:applications/fork-
event-storage-backup-pipeline
      SemanticVersion: 1.0.0
    Parameters:
      #The ARN of the Amazon SNS topic whose messages should be backed up to the Amazon
S3 bucket.
      TopicArn: !Ref MySNSTopic
```

パラメータ値を指定すると、AWS CloudFormation 組み込み関数を使用してテンプレート内の他のリソースを参照できます。たとえば、上記の YAML スニペットでは、TopicArn パラメータは AWS SAM テンプレートの他の場所で MySNSTopic 定義されている [AWS::SNS::Topic](#) リソースを参照します。詳細については、『AWS CloudFormation ユーザーガイド』の「[組み込み関数リファレンス](#)」を参照してください。

Note

AWS SAR アプリケーションの AWS Lambda コンソールページには、SAM リソースとしてコピーボタンが含まれており、SAR アプリケーションをクリップボードにネストするために必要な YAML AWS をコピーします。

Amazon SNS Event Fork Pipelines サンプルアプリケーションをデプロイしてテストする

イベント駆動型アプリケーションの開発を加速するには、Event Fork Pipelines を搭載した AWS イベント処理パイプラインを Amazon SNS トピックにサブスクライブできます。AWS Event Fork Pipelines は、[AWS Serverless Application Model](#) (AWS SAM) に基づくオープンソースの [ネストされたアプリケーションの](#)スイートです。[AWS Event Fork Pipelines スイート](#) (カスタム IAM ロールまたはリソースポリシーを作成する Show アプリケーションを選択) から AWS アカウントに直接デプロイできます。詳細については、「[AWS Event Fork Pipelines の仕組み](#)」を参照してください。

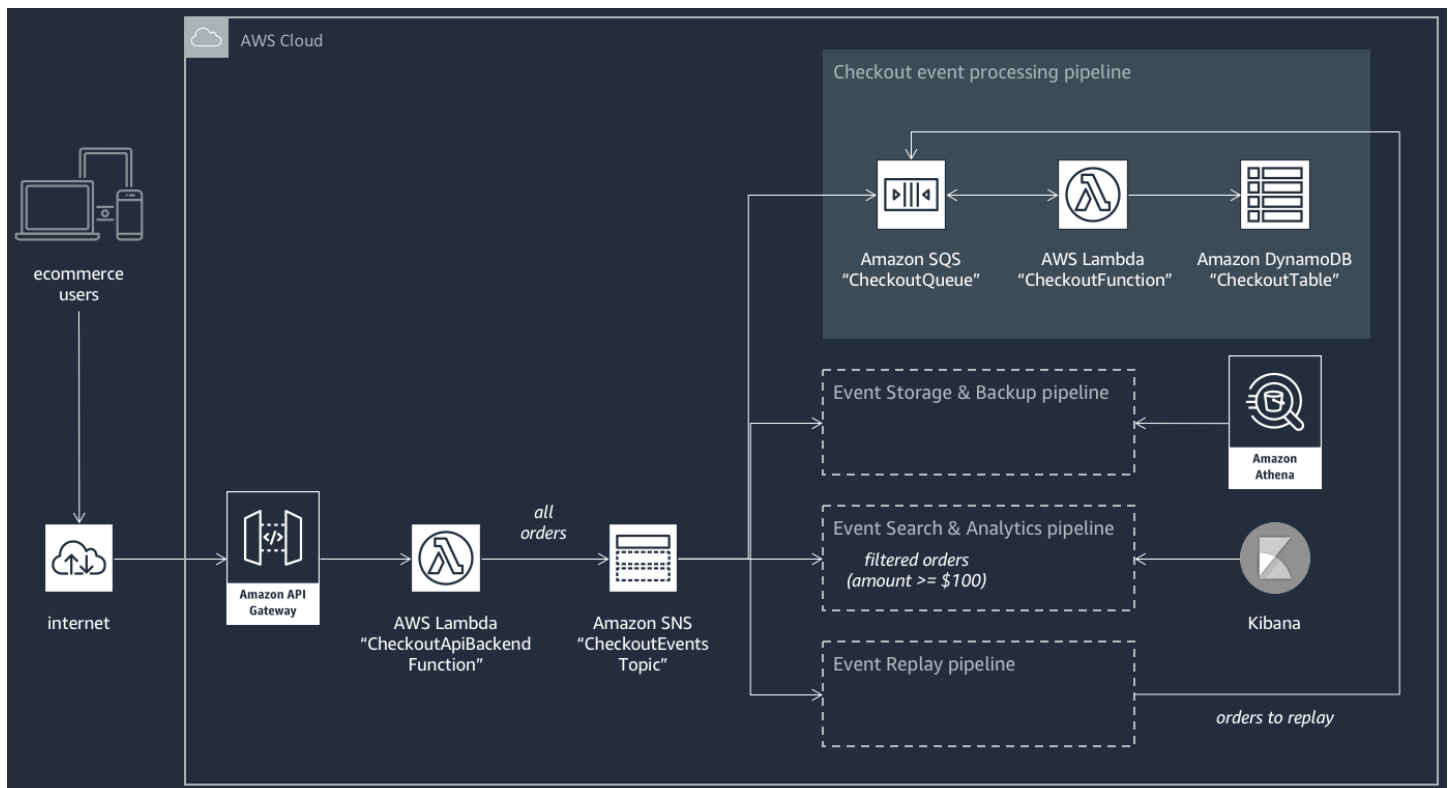
このページでは、AWS マネジメントコンソール を使用して AWS Event Fork Pipelines サンプルアプリケーションをデプロイおよびテストする方法を示します。

⚠ Important

AWS Event Fork Pipelines サンプルアプリケーションのデプロイが完了した後に不要なコストが発生しないようにするには、その CloudFormation スタックを削除します。詳細については、「AWS CloudFormation ユーザーガイド」の「[CloudFormation コンソールのスタックを削除](#)」を参照してください。

AWS Event Fork Pipelines ユースケースの例

次のシナリオでは、Event Fork Pipelines を使用する AWS イベント駆動型のサーバーレス e コマースアプリケーションについて説明します。この[例の e コマースアプリケーション](#)を使用し、AWS Serverless Application Repository AWS Lambda コンソール AWS アカウント を使用してにデプロイできます。ここでテストしてGitHub でソースコードを調べることができます。



この e コマースアプリケーションは、API Gateway によってホストされ、AWS Lambda 関数によってバックアップされた RESTful API を通じて購入者からの注文を受け取ります CheckoutApiBackendFunction。この関数は、すべての受注を CheckoutEventsTopic という

う名前の Amazon SNS トピックに発行します。このトピックでは、これらの受注を 4 つの異なるパイプラインにファンアウトします。

最初のパイプラインは、e コマースアプリケーションの所有者によって設計および実装された通常のチェックアウト処理パイプラインです。このパイプラインには、受信したすべての注文をバッファ CheckoutQueue する Amazon SQS キュー、これらの注文を処理するためにキューをポーリング CheckoutFunction するという名前の AWS Lambda 関数、およびすべての注文を安全に保存する DynamoDB テーブル CheckoutTable があります。

AWS Event Fork Pipelines の適用

e コマースアプリケーションのコンポーネントは、主要ビジネスロジックを処理します。ただし、e コマースアプリケーションの所有者は以下にも対処する必要があります。

- コンプライアンス - 安全な、圧縮されたバックアップの保管時の暗号化と機密情報のサニタイズ
- レジリエンス - フルフィルメントプロセスの中断が発生した場合の最新の注文の再生
- 検索可能性 - 受注に対する分析の実行とメトリクスの生成

このイベント処理ロジックを実装する代わりに、アプリケーション所有者は AWS Event Fork Pipelines を CheckoutEventsTopic Amazon SNS トピックにサブスクライブできます。

- [イベントのストレージおよびバックアップパイプライン](#) は、データを変換して、クレジットカードの詳細の削除、データの 60 秒間のバッファ処理、GZIP を使用した圧縮、Amazon S3 のデフォルトのカスタマーマネージドキーを使用した暗号化を行うように設定されています。このキーはによって管理 AWS され、AWS Key Management Service () を利用します AWS KMS。

詳細については、「Amazon Data Firehose デベロッパーガイド」の「[送信先の Amazon S3 の選択](#)」、「[Amazon Data Firehose データ送信](#)」、および「[設定](#)」を参照してください。

- [イベントの検索および分析パイプライン](#) には、30 秒間のインデックス再試行期間、検索ドメインでインデックスの作成に失敗した注文を保存するためのバケット、およびインデックスを作成した注文のセットを制限するためのフィルターポリシーが設定されています。

詳細については、「Amazon Data Firehose デベロッパーガイド」の「[宛先の OpenSearch Service の選択](#)」を参照してください。

- [イベントの再生パイプライン](#) には、e コマースアプリケーションの所有者によって設計および実装された通常の注文処理パイプラインの Amazon SQS キュー部分が設定されています。

詳細については、『Amazon Simple Queue Service デベロッパーガイド』の「[キーの名前と URL](#)」を参照してください。

次の JSON フィルターポリシーは、イベントの検索および分析パイプラインの設定で設定されます。これは、合計金額が 100 USD 以上の受注とのみ一致します。詳細については、「[Amazon SNS メッセージフィルター処理](#)」を参照してください。

```
{
  "amount": [{ "numeric": [ ">=", 100 ] }]
}
```

AWS Event Fork Pipelines パターンを使用すると、e コマースアプリケーションの所有者は、イベント処理の差別化ロジックのコーディングに従うことが多い開発オーバーヘッドを回避できます。代わりに、AWS Event Fork Pipelines を から直接 AWS Serverless Application Repository にデプロイできます AWS アカウント。

ステップ 1: サンプル Amazon SNS アプリケーションをデプロイする

1. [AWS Lambda コンソール](#) にサインインします。
2. ナビゲーションパネルで [関数] を選択し、[関数の作成] を選択します。
3. [関数の作成] ページで、次の操作を実行します。
 - a. [サーバーレスアプリケーションリポジトリの参照]、[パブリックアプリケーション]、[カスタム IAM ロールまたはリソースポリシーを作成するアプリケーションの表示] を選択します。
 - b. `fork-example-ecommerce-checkout-api` を検索し、このアプリケーションを選択します。
4. [fork-example-ecommerce-checkout-api] ページで、以下の操作を行います。
 - a. [アプリケーション設定] セクションで、[アプリケーション名] に名前 (`fork-example-ecommerce-my-app` など) を入力します。

Note

- リソースを後で簡単に見つけられるように、プレフィックス `fork-example-ecommerce` を保持します。

- 名前は、デプロイごとに一意にする必要があります。アプリケーション名を再利用すると、デプロイは (新しいスタックを作成するのではなく) 以前にデプロイされた CloudFormation スタックのみを更新します。

b. (オプション) アプリケーションの Lambda 関数の実行のためにも以下の LogLevel 設定のいずれかを入力します。

- DEBUG
- ERROR
- INFO (デフォルト)
- WARNING

5. [このアプリケーションがカスタム IAM ロール、リソースポリシーを作成し、ネストされたアプリケーションをデプロイすることを承認します] を選択して、ページの下部にある [デプロイ] を選択します。

[fork-example-ecommerce-**my-app** のデプロイステータス] ページで、Lambda が [アプリケーションをデプロイ中] ステータスを表示します。

リソースセクションで、スタックの作成 CloudFormation を開始し、各リソースの CREATE_IN_PROGRESS ステータスを表示します。プロセスが完了すると、は CREATE_COMPLETE ステータス CloudFormation を表示します。

 Note

すべてのリソースがデプロイされるまで 20 ~ 30 分かかる場合があります。

デプロイが完了すると、Lambda によって [アプリケーションはデプロイ済みです] ステータスが表示されます。

ステップ 2: SNS にリンクされたサンプルアプリケーションを実行する

1. AWS Lambda コンソールのナビゲーションパネルで、アプリケーションを選択します。
2. [アプリケーション] ページの検索フィールドで、serverlessrepo-fork-example-ecommerce-**my-app** を検索し、そのアプリケーションを選択します。
3. [リソース] セクションで、以下の操作を行います。

- a. タイプが `ApiGateway RestApi` であるリソースを見つけるには、すべてのリソースを [タイプ] (`ServerlessRestApi` など) でソートし、リソースを展開します。
- b. 2 つのネストされたリソース (`ApiGateway デプロイタイプ` と `ApiGateway ステージタイプ`) が表示されます。
- c. リンク [`Prod API エンドポイント`] をコピーし、これに `/checkout` を付加します。次に例を示します。

```
https://abcdefghij.execute-api.us-east-2.amazonaws.com/Prod/checkout
```

4. 次の JSON を `test_event.json` という名前のファイルにコピーします。

```
{
  "id": 15311,
  "date": "2019-03-25T23:41:11-08:00",
  "status": "confirmed",
  "customer": {
    "id": 65144,
  },
  "quantity": 2,
  "price": 25.00,
  "subtotal": 50.00
}]
}
```

5. API エンドポイントに HTTPS リクエストを送信するには、`curl` コマンドを実行してサンプル イベントペイロードを入力として渡します。次に例を示します。

```
curl -d "$(cat test_event.json)" https://abcdefghij.execute-api.us-east-2.amazonaws.com/Prod/checkout
```

API は、次の空のレスポンスを返し、実行が成功したことを示します。

```
{ }
```

ステップ 3: Amazon SNS アプリケーションとパイプラインのパフォーマンスを検証する

ステップ 1: サンプルのチェックアウトパイプラインの実行を検証する

1. [Amazon DynamoDB コンソール](#)にサインインします。
2. ナビゲーションパネルで、[テーブル] を選択します。
3. `serverlessrepo-fork-example` を検索して `CheckoutTable` を選択します。
4. テーブルの詳細ページで [項目] を選択し、作成済みの項目を選択します。

保存済みの属性が表示されます。

ステップ 2: イベントのストレージおよびバックアップパイプラインの実行を検証する

1. [\[Amazon S3 コンソール\]](#) にサインインします。
2. ナビゲーションパネルで [バケット] を選択します。
3. `serverlessrepo-fork-example` を検索して、`CheckoutBucket` を選択します。
4. `.gz` を拡張子とするファイルが見つかるまで、ディレクトリ階層を移動します。
5. ファイルをダウンロードするには、[アクション]、[開く] の順に選択します。
6. パイプラインには、コンプライアンスの理由でクレジットカード情報をサニタイズする Lambda 関数が設定されています。

保存済みの JSON ペイロードにクレジットカード情報が含まれていないことを確認するために、ファイルを解凍します。

ステップ 3: イベントの検索および分析パイプラインの実行を検証する

1. [OpenSearch Service コンソール](#)にサインインします。
2. ナビゲーションパネルの [My domains] で、`server1-analyt` というプレフィックスが付いたドメインを選択します。
3. パイプラインには、数値一致条件を設定する Amazon SNS サブスクリプションフィルターポリシーが設定されています。

金額が 100 USD を超える注文を参照するためにイベントのインデックスが作成されていることを確認するには、`[server1-analyt-abcdefgh1ijk]` ページで、[インデックス]、`[checkout_events]` の順に選択します。

ステップ 4: イベントの再生パイプラインの実行を検証する

1. [Amazon SQS コンソール](#)にサインインします。
2. キューのリストで、serverlessrepo-fork-example を検索して ReplayQueue を選択します。
3. [メッセージの送信と受信] を選択します。
4. [Send and receive messages in fork-example-ecommerce-**my-app**...ReplayP-ReplayQueue-**123ABCD4E5F6**] (fork-example-ecommerce-my-app...ReplayP-ReplayQueue-123ABCD4E5F6 のメッセージの表示/削除) ダイアログボックスで、[Poll for messages] (メッセージのポーリング) を選択します。
5. イベントがキューに入っていることを確認するには、キューに表示されているメッセージの横にある [詳細] を選択します。

ステップ 4: 復旧のために問題をシミュレートしてイベントを再生する

ステップ 1: シミュレートした問題を有効にして別の API リクエストを送信する

1. [AWS Lambda コンソール](#) にサインインします。
2. ナビゲーションパネルで [関数] を選択します。
3. serverlessrepo-fork-example を検索して CheckoutFunction を選択します。
4. [fork-example-ecommerce-**my-app**-CheckoutFunction-**ABCDEF**...] ページの [環境変数] セクションで、BUG_ENABLED 変数を true に設定して [保存] を選択します。
5. 次の JSON を test_event_2.json という名前のファイルにコピーします。

```
{
  "id": 9917,
  "date": "2019-03-26T21:11:10-08:00",
  "status": "confirmed",
  "customer": {
    "id": 56999,
    "quantity": 1,
    "price": 75.00,
    "subtotal": 75.00
  }
}
```

6. API エンドポイントに HTTPS リクエストを送信するには、curl コマンドを実行してサンプルイベントペイロードを入力として渡します。次に例を示します。

```
curl -d "$(cat test_event_2.json)" https://abcdefghij.execute-api.us-east-2.amazonaws.com/Prod/checkout
```

API は、次の空のレスポンスを返し、実行が成功したことを示します。

```
{ }
```

ステップ 2: シミュレートしたデータの破損を検証する

1. [Amazon DynamoDB コンソール](#) にサインインします。
2. ナビゲーションパネルで、[テーブル] を選択します。
3. `serverlessrepo-fork-example` を検索して `CheckoutTable` を選択します。
4. テーブルの詳細ページで [項目] を選択し、作成済みの項目を選択します。

保存済みの属性が表示されます。一部の属性は [CORRUPTED!] とマークされています。

ステップ 3: シミュレートした問題を無効にする

1. [AWS Lambda コンソール](#) にサインインします。
2. ナビゲーションパネルで [関数] を選択します。
3. `serverlessrepo-fork-example` を検索して `CheckoutFunction` を選択します。
4. `[fork-example-ecommerce-my-app-CheckoutFunction-ABCDEF...]` ページの [環境変数] セクションで、`BUG_ENABLED` 変数を `false` に設定して [保存] を選択します。

ステップ 4: 再生を有効にして問題から復旧する

1. AWS Lambda コンソールのナビゲーションパネルで、関数を選択します。
2. `serverlessrepo-fork-example` を検索して `ReplayFunction` を選択します。
3. [Designer] セクションを展開して [SQS] タイルを選択し、次に [SQS] セクションで [有効] を選択します。

Note

Amazon SQS のイベントソーストリガーが有効になるまで約 1 分かかります。

4. [保存] を選択します。
5. 復旧された属性を表示するには、Amazon DynamoDB コンソールに戻ります。
6. 再生を無効にするには、AWS Lambda コンソールに戻り、 の Amazon SQS イベントソーストリガーを無効にしますReplayFunction。

AWS Event Fork Pipelines を Amazon SNS トピックにサブスクライブする

イベント駆動型アプリケーションの開発を加速するには、Event Fork Pipelines を搭載した AWS イベント処理パイプラインを Amazon SNS トピックにサブスクライブできます。AWS Event Fork Pipelines は、[AWS Serverless Application Model](#) (AWS SAM) に基づくオープンソースの[ネストされたアプリケーションの](#)スイートです。[AWS Event Fork Pipelines スイート](#) (カスタム IAM ロールまたはリソースポリシーを作成する Show アプリケーションを選択) から AWS アカウントに直接デプロイできます。詳細については、「[AWS Event Fork Pipelines の仕組み](#)」を参照してください。

このセクションでは、を使用してパイプライン AWS マネジメントコンソール をデプロイし、AWS Event Fork Pipelines を Amazon SNS トピックにサブスクライブする方法について説明します。開始する前に、[Amazon SNS トピック](#)を作成します。

パイプラインを構成するリソースを削除するには、AWS Lambda コンソールの のアプリケーションページでパイプラインを見つけ、SAM テンプレートセクションを展開し、CloudFormation スタックを選択し、その他のアクション、スタックの削除を選択します。

イベントのストレージとバックアップパイプラインを Amazon SNS にデプロイしてサブスクライブする

イベントのアーカイブと分析のために、Amazon SNS は Amazon Data Firehose とのネイティブ統合の使用を推奨するようになりました。Firehose 配信ストリームを SNS トピックにサブスクライブできます。これにより、Amazon Simple Storage Service (Amazon S3) バケット、Amazon Redshift テーブル、Amazon OpenSearch Service (OpenSearch Service) などのアーカイブと分析エンドポイントへ通知を送信することができます。Firehose 配信ストリームで Amazon SNS を使用するのには、フルマネージド型のコードレスソリューションであり、AWS Lambda 関数を使用する必要はありません。詳細については、「[Firehose 配信ストリームへのファンアウト](#)」を参照してください。

このページでは、[イベントのストレージおよびバックアップパイプライン](#)をデプロイして Amazon SNS トピックにサブスクライブする方法を説明します。このプロセスは、パイプラインに関連付け

られた AWS SAM テンプレートを自動的に CloudFormation スタックに変換し、スタックを にデプロイします AWS アカウント。また、このプロセスでは、イベントのストレージおよびバックアップパイプラインを構成する、以下のようなリソースのセットを作成して設定します。

- Amazon SQS キュー
- Lambda function
- Firehose 配信ストリーム
- Amazon S3 バックアップバケット

Amazon S3 バケットを送信先としてストリームを設定する方法の詳細については、「Amazon Data Firehose API リファレンス」の「[S3DestinationConfiguration](#)」を参照してください。

イベントの変換、イベントバッファ処理の設定、イベント圧縮の設定、およびイベント暗号化の設定の詳細については、「Amazon Data Firehose デベロッパーガイド」の「[配信ストリームの作成](#)」を参照してください。

イベントのフィルター処理の詳細については、このガイドの「[Amazon SNS サブスクリプション フィルターポリシー](#)」を参照してください。

1. [AWS Lambda コンソール](#) にサインインします。
2. ナビゲーションパネルで [関数] を選択し、[関数の作成] を選択します。
3. [関数の作成] ページで、次の操作を実行します。
 - a. [サーバーレスアプリケーションリポジトリの参照]、[パブリックアプリケーション]、[カスタム IAM ロールまたはリソースポリシーを作成するアプリケーションの表示] を選択します。
 - b. fork-event-storage-backup-pipeline を検索し、このアプリケーションを選択します。
4. [fork-event-storage-backup-pipeline] ページで、以下の操作を行います。
 - a. [アプリケーション設定] セクションで、[アプリケーション名] に名前 (my-app-backup など) を入力します。

 Note

- 名前は、デプロイごとに一意にする必要があります。アプリケーション名を再利用すると、デプロイは (新しいスタックを作成するのではなく) 以前にデプロイされた CloudFormation スタックのみを更新します。

- b. (オプション) [BucketArn] に、着信イベントのロード先となる Amazon S3 バケットの ARN を入力します。値を入力しない場合、AWS アカウントに新しい Amazon S3 バケットが作成されます。
- c. (オプション) [DataTransformationFunctionArn] に、着信イベントを変換する Lambda 関数の ARN を入力します。値を入力しないと、データ変換は無効になります。
- d. (オプション) アプリケーションの Lambda 関数の実行のためにも以下の LogLevel 設定のいずれかを入力します。
 - DEBUG
 - ERROR
 - INFO (デフォルト)
 - WARNING
- e. [TopicArn] に、このフォークパイプラインのインスタンスをサブスクライブする Amazon SNS トピックの ARN を入力します。
- f. (オプション) [StreamBufferingIntervalInSeconds] および [StreamBufferingSizeInMBs] に、着信イベントのバッファ処理を設定するための値を入力します。値を入力しないと、300 秒と 5 MB が使用されます。
- g. (オプション) 着信イベントを圧縮する形式として次の [StreamCompressionFormat] 設定のいずれかを入力します。
 - GZIP
 - SNAPPY
 - UNCOMPRESSED (デフォルト)
 - ZIP
- h. (オプション) [StreamPrefix] に、Amazon S3 バックアップバケットに保存されているファイルを指定するための文字列プレフィックスを入力します。値を入力しないと、プレフィックスは使用されません。

- i. (オプション) [SubscriptionFilterPolicy] に、着信イベントをフィルター処理するために使用する Amazon SNS サブスクリプションフィルターポリシーを JSON 形式で入力します。フィルターポリシーは、OpenSearch Service インデックスでインデックスを作成するイベントを決定します。値を入力しないと、フィルター処理は使用されません (すべてのイベントにインデックスが作成されます)。
- j. (オプション) SubscriptionFilterPolicyScope には、文字列 MessageBody または MessageAttributes を入力して、ペイロードベースまたは属性ベースのメッセージフィルタリングを有効にします。
- k. [I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.] を選択して、[デプロイ] を選択します。

[**my-app** のデプロイステータス] ページに、Lambda によって [Your application is being deployed] ステータスが表示されます。

リソースセクションで、スタックの作成 CloudFormation を開始し、各リソースの CREATE_IN_PROGRESS ステータスを表示します。プロセスが完了すると、は CREATE_COMPLETE ステータス CloudFormation を表示します。

デプロイが完了すると、Lambda によって [アプリケーションはデプロイ済みです] ステータスが表示されます。

Amazon SNS トピックに発行されたメッセージは、イベントのストレージおよびバックアップパイプラインによってプロビジョニングされた Amazon S3 バックアップバケットに自動的に保存されます。

イベントの検索および分析パイプラインを Amazon SNS にデプロイしてサブスクライブする

イベントのアーカイブと分析のために、Amazon SNS は Amazon Data Firehose とのネイティブ統合の使用を推奨するようになりました。Firehose 配信ストリームを SNS トピックにサブスクライブできます。これにより、Amazon Simple Storage Service (Amazon S3) バケット、Amazon Redshift テーブル、Amazon OpenSearch Service (OpenSearch Service) などのアーカイブと分析エンドポイントへ通知を送信することができます。Firehose 配信ストリームで Amazon SNS を使用するのには、フルマネージド型のコードレスソリューションであり、AWS Lambda 関数を使用する必要はありません。詳細については、「[Firehose 配信ストリームへのファンアウト](#)」を参照してください。

このページでは、[イベントの検索および分析パイプライン](#)をデプロイして Amazon SNS トピックにサブスクライブする方法を示します。このプロセスは、パイプラインに関連付けられた AWS SAM テンプレートを自動的に CloudFormation スタックに変換し、スタックを にデプロイします AWS アカウント。また、このプロセスでは、イベントの検索および分析パイプラインを構成する、以下のようナリソースのセットを作成して設定します。

- Amazon SQS キュー
- Lambda function
- Firehose 配信ストリーム
- Amazon OpenSearch Service ドメイン
- Amazon S3 配信不能バケット

インデックスを送信先としてストリームを設定する方法の詳細については、「Amazon Data Firehose API リファレンス」の「[ElasticsearchDestinationConfiguration](#)」を参照してください。

イベントの変換、イベントバッファ処理の設定、イベント圧縮の設定、およびイベント暗号化の設定の詳細については、「Amazon Data Firehose デベロッパーガイド」の「[配信ストリームの作成](#)」を参照してください。

イベントのフィルター処理の詳細については、このガイドの「[Amazon SNS サブスクリプションフィルターポリシー](#)」を参照してください。

1. [AWS Lambda コンソール](#) にサインインします。
2. ナビゲーションパネルで [関数] を選択し、[関数の作成] を選択します。
3. [関数の作成] ページで、次の操作を実行します。
 - a. [サーバーレスアプリケーションリポジトリの参照]、[パブリックアプリケーション]、[カスタム IAM ロールまたはリソースポリシーを作成するアプリケーションの表示] を選択します。
 - b. fork-event-search-analytics-pipeline を検索し、このアプリケーションを選択します。
4. [fork-event-search-analytics-pipeline] ページで、以下の操作を行います。
 - a. [アプリケーション設定] セクションで、[アプリケーション名] に名前 (my-app-search など) を入力します。

Note

名前は、デプロイごとに一意にする必要があります。アプリケーション名を再利用すると、デプロイは (新しいスタックを作成するのではなく) 以前にデプロイされた CloudFormation スタックのみを更新します。

- b. (オプション) [DataTransformationFunctionArn] に、着信イベントの変換に使用する Lambda 関数の ARN を入力します。値を入力しないと、データ変換は無効になります。
- c. (オプション) アプリケーションの Lambda 関数の実行のため以下の LogLevel 設定のいずれかを入力します。
 - DEBUG
 - ERROR
 - INFO (デフォルト)
 - WARNING
- d. (オプション) [SearchDomainArn] に、OpenSearch Service ドメインの ARN を入力します。このドメインは、必要なコンピューティングおよびストレージ機能を設定するクラスターです。値を入力しないと、新しいドメインがデフォルト設定で作成されます。
- e. [TopicArn] に、このフォークパイプラインのインスタンスをサブスクライブする Amazon SNS トピックの ARN を入力します。
- f. [SearchIndexName] に、イベントの検索と分析用の OpenSearch Service インデックスの名前を入力します。

Note

インデックス名には、次の制限が適用されます。

- 大文字を含めることはできません
- 次の文字を含めることはできません: \ / * ? " < > | ` , #
- 次の文字で始めることはできません: - + _
- 次の形式にすることはできません: . . .
- 80 文字より長くすることはできません
- 255 バイトより長くすることはできません
- コロンを含めることはできません (OpenSearch Service 7.0 から)

- g. (オプション) OpenSearch Service インデックスのローテーション期間に、以下の [SearchIndexRotationPeriod] 設定のいずれかを入力します。

- NoRotation (デフォルト)
- OneDay
- OneHour
- OneMonth
- OneWeek

インデックスのローテーションでは、インデックス名にタイムスタンプを付加し、古いデータの有効期限切れをわかりやすくします。

- h. [SearchTypeName] に、インデックスでイベントを整理するための OpenSearch Service タイプの名前を入力します。

 Note

- OpenSearch Service タイプ名には任意の文字 (null バイトを除く) を含めることができますが、先頭に `_` を使用することはできません。
- OpenSearch Service 6.x では、インデックスあたり 1 つのタイプのみが存在できます。別のタイプが既にある既存のインデックスに新しいタイプを指定すると、Firehose はランタイムエラーを返します。

- i. (オプション) [StreamBufferingIntervalInSeconds] および [StreamBufferingSizeInMBs] に、着信イベントのバッファ処理を設定するための値を入力します。値を入力しないと、300 秒と 5 MB が使用されます。
- j. (オプション) 着信イベントを圧縮する形式として次の [StreamCompressionFormat] 設定のいずれかを入力します。
- GZIP
 - SNAPPY
 - UNCOMPRESSED (デフォルト)
 - ZIP

- k. (オプション) [StreamPrefix] に、Amazon S3 配信不能バケットに保存されているファイルを指定するための文字列プレフィックスを入力します。値を入力しないと、プレフィックスは使用されません。
- l. (オプション) [StreamRetryDurationInSecons] に、Firehose が OpenSearch Service インデックスでイベントのインデックスを作成できない場合の再試行期間を入力します。値を入力しないと、300 秒が使用されます。
- m. (オプション) [SubscriptionFilterPolicy] に、着信イベントをフィルター処理するために使用する Amazon SNS サブスクリプションフィルターポリシーを JSON 形式で入力します。フィルターポリシーは、OpenSearch Service インデックスでインデックスを作成するイベントを決定します。値を入力しないと、フィルター処理は使用されません (すべてのイベントにインデックスが作成されます)。
- n. [I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.] を選択して、[デプロイ] を選択します。

[**my-app-search** のデプロイステータス] ページに、Lambda によって [Your application is being deployed] ステータスが表示されます。

リソースセクションで、スタックの作成 CloudFormation を開始し、各リソースの CREATE_IN_PROGRESS ステータスを表示します。プロセスが完了すると、は CREATE_COMPLETE ステータス CloudFormation を表示します。

デプロイが完了すると、Lambda によって [アプリケーションはデプロイ済みです] ステータスが表示されます。

Amazon SNS トピックに発行されたメッセージは、イベントの検索および分析パイプラインによってプロビジョニングされた OpenSearch Service インデックスで自動的にインデックスが作成されます。パイプラインでイベントのインデックスを作成できない場合、イベントは Amazon S3 配信不能バケットに保存されます。

Amazon SNS 統合によるイベントの再生パイプラインのデプロイ

このページでは、[\[イベントの再生パイプライン\]](#) をデプロイして Amazon SNS トピックにサブスクライブする方法を示します。このプロセスは、パイプラインに関連付けられた AWS SAM テンプレートを自動的に CloudFormation スタックに変換し、スタックを にデプロイします AWS アカウント。また、このプロセスでは、イベントの再生パイプラインを構成する Amazon SQS キューや Lambda 関数などのリソースのセットを作成して設定します。

イベントのフィルター処理の詳細については、このガイドの「[Amazon SNS サブスクリプション フィルターポリシー](#)」を参照してください。

1. [AWS Lambda コンソール](#) にサインインします。
2. ナビゲーションパネルで [関数] を選択し、[関数の作成] を選択します。
3. [関数の作成] ページで、次の操作を実行します。
 - a. [サーバーレスアプリケーションリポジトリの参照]、[パブリックアプリケーション]、[カスタム IAM ロールまたはリソースポリシーを作成するアプリケーションの表示] を選択します。
 - b. `fork-event-replay-pipeline` を検索し、このアプリケーションを選択します。
4. [fork-event-replay-pipeline] ページで、以下の操作を行います。
 - a. [アプリケーション設定] セクションで、[アプリケーション名] に名前 (`my-app-replay` など) を入力します。

 Note

名前は、デプロイごとに一意にする必要があります。アプリケーション名を再利用すると、デプロイは (新しいスタックを作成するのではなく) 以前にデプロイされた CloudFormation スタックのみを更新します。

- b. (オプション) アプリケーションの Lambda 関数の実行のためにも以下の LogLevel 設定のいずれかを入力します。
 - DEBUG
 - ERROR
 - INFO (デフォルト)
 - WARNING
- c. (オプション) [ReplayQueueRetentionPeriodInSeconds] に、Amazon SQS 再生キューでメッセージを保持する時間を秒単位で入力します。値を入力しないと、1,209,600 秒 (14 日間) が使用されます。
- d. [TopicArn] に、このフォークパイプラインのインスタンスをサブスクライブする Amazon SNS トピックの ARN を入力します。
- e. [DestinationQueueName] に、Lambda 再生関数がメッセージを転送する先の Amazon SQS キューの名前を入力します。

- f. (オプション) [SubscriptionFilterPolicy] に、着信イベントをフィルター処理するために使用する Amazon SNS サブスクリプションフィルターポリシーを JSON 形式で入力します。フィルターポリシーは、再生用にバッファ処理するイベントを決定します。値を入力しないと、フィルター処理は使用されません (すべてのイベントが再生用にバッファ処理されます)。
- g. [I acknowledge that this app creates custom IAM roles, resource policies and deploys nested applications.] を選択して、[デプロイ] を選択します。

[**my-app-replay** のデプロイステータス] ページで、Lambda によって [Your application is being deployed] ステータスが表示されます。

リソースセクションで、スタックの作成 CloudFormation を開始し、各リソースの CREATE_IN_PROGRESS ステータスを表示します。プロセスが完了すると、は CREATE_COMPLETE ステータス CloudFormation を表示します。

デプロイが完了すると、Lambda によって [アプリケーションはデプロイ済みです] ステータスが表示されます。

Amazon SNS トピックに発行されたメッセージは、イベントの再生パイプラインによってプロビジョニングされた Amazon SQS キューで自動的に再生用にバッファ処理されます。

Note

再生は、デフォルトでは無効になります。再生を有効にするには、Lambda コンソールで関数のページに移動し、[Designer] セクションを展開して [SQS] タイルを選択します。次に [SQS] セクションで [有効] を選択します。

Amazon SNS での Amazon EventBridge スケジューラの使用

[Amazon EventBridge スケジューラ](#) はサーバーレススケジューラで、一元化されたマネージドサービスからタスクを作成、実行、管理できます。EventBridge スケジューラでは、繰り返しのパターンに Cron やレート式を使ってスケジュールを作成したり、1回限りの呼び出しを設定したりできます。配信時間枠の柔軟な設定、再試行制限の定義、失敗した API 呼び出しの最大保持時間の設定を行うことができます。

このページでは、EventBridge スケジューラを使用してスケジュールに基づき Amazon SNS トピックからメッセージを発行する方法について説明します。

実行ロールの設定

新しいスケジュールを作成する場合、EventBridge スケジューラにはユーザーに代わってターゲット API オペレーションを呼び出すアクセス許可が必要です。実行ロールを使用して、これらのアクセス許可を EventBridge スケジューラに付与します。スケジュールの実行ロールにアタッチするアクセス許可ポリシーによって、必要なアクセス許可が定義されます。これらのアクセス許可は、EventBridge スケジューラが呼び出すターゲット API によって異なります。

次の手順のように EventBridge スケジューラコンソールを使用してスケジュールを作成すると、EventBridge スケジューラは選択したターゲットに基づき実行ロールを自動的に設定します。EventBridge スケジューラ SDKs、AWS CLIまたはのいずれかを使用してスケジュールを作成する場合は CloudFormation、EventBridge スケジューラがターゲットを呼び出すために必要なアクセス許可を付与する既存の実行ロールが必要です。スケジュールに合わせて実行ロールを手動で設定する方法についての詳細は、「EventBridge スケジューラユーザーガイド」の「[実行ロールを設定する](#)」を参照してください。

新しいスケジュールを作成する

コンソールを使用してスケジュールを作成するには

1. <https://console.aws.amazon.com/scheduler/home> で Amazon EventBridge スケジューラコンソールを開きます。
2. [スケジュール] ページで、[スケジュールを作成] を選択します。
3. [スケジュールの詳細を指定] ページの [スケジュールの名前と説明] セクションで、次を実行します。
 - a. [スケジュール名] で、スケジュールの名前を入力します。例えば、**MyTestSchedule**。
 - b. (オプション) [説明] で、スケジュールの説明を入力します。例えば、**My first schedule**。
 - c. [スケジュールグループ] で、ドロップダウンリストからスケジュールグループを選択します。グループがない場合は、[デフォルト] を選択します。スケジュールグループを作成するには、[独自のスケジュールを作成] を選択します。

スケジュールグループを使用して、スケジュールのグループにタグを追加します。

4. • スケジュールオプションを選択します。

頻度	手順	
[1 回限りのスケジュール] 1 回限りのスケジュールは、指定した日時に 1 回だけターゲットを呼び出します。	[日付と時刻] で、次を実行します。 - 有効な日付を YYYY/MM/DD 形式で入力します。 - タイムスタンプを 24 時間 (hh:mm) 形式で入力します。 [タイムゾーン] で、タイムゾーンを選択します。	

頻度	手順	
[繰り返しのスケジュール] 繰り返しのスケジュールは、cron 式またはレート式を使用して指定したレートでターゲットを呼び出します。	a. [スケジュールの種類] では、次のいずれかを実行します。 • Cron 式を使用してスケジュールを定義するには、[cron ベースのスケジュール] を選択して Cron 式を入力します。 • Rate 式を使用してスケジュールを定義するには、[rate ベースのスケジュール] を選択して Rate 式を入力します。 cron および rate 式の詳細については、「Amazon EventBridge スケジューラ ユーザーガイド」の「EventBridge スケジューラのスケジュールタイプ」を参照してください。 b. [フレックスタイムウィンドウ] で、[オフ] を選択してオプションをオフにするか、事前定義された時間枠のいずれかを選択します。例えば、[15 分] を選択し、1 時間に 1 回ターゲットを呼び出す繰り返しのスケジュー	

頻度	手順	
	ルを設定した場合、スケジュールは毎時の開始後 15 分以内に実行されます。	

5. (オプション) 前のステップで [定期的なスケジュール] を選択した場合は、[時間枠] セクションで次を実行します。
 - a. [タイムゾーン] で、タイムゾーンを選択します。
 - b. [開始日時] で、有効な日付を YYYY/MM/DD 形式で入力してから、タイムスタンプを 24 時間 (hh:mm) 形式で指定します。
 - c. [終了日時] で、有効な日付を YYYY/MM/DD 形式で入力してから、タイムスタンプを 24 時間 (hh:mm) 形式で指定します。
6. [次へ] を選択します。
7. ターゲットの選択ページで、EventBridge スケジューラが呼び出す AWS API オペレーションを選択します。
 - a. [Amazon SNS 発行] を選択します。
 - b. [発行] セクションで、SNS トピックを選択するか、[新しい SNS トピックを作成] を選択します。
 - c. (オプション) JSON ペイロードを入力します。ペイロードを入力しない場合、EventBridge スケジューラは空のイベントを使用して関数を呼び出します。
8. [次へ] を選択します。
9. [Settings] (設定) ページで、以下の操作を行います。
 - a. スケジュールをオンにするには、[スケジュールの状態] で [スケジュールを有効にする] をオンに切り替えます。
 - b. スケジュールの再試行ポリシーを設定するには、[再試行ポリシーとデッドレターキュー (DLQ)] で次を実行します。
 - [再試行] を切り替えてオンにします。
 - [イベントの最大有効期間] で、EventBridge スケジューラが未処理のイベントを保持しなければならない最大の [時間] と [分] を入力します。
 - 最大 24 時間です。

- [最大再試行回数] で、ターゲットがエラーを返した場合に EventBridge スケジューラがスケジュールを再試行する最大回数を入力します。

再試行の最大値は 185 です。

再試行ポリシーを使用すると、スケジュールがそのターゲットの呼び出しに失敗した場合、EventBridge スケジューラはスケジュールを再実行します。設定されている場合は、スケジュールの最大保持時間と再試行を設定する必要があります。

- c. EventBridge スケジューラが未配信のイベントを保存する場所を選択します。

[デッドレターキュー (DLQ)] オプション	手順	
保存しない	[None] を選択します。	
スケジュールを作成するのと同じ AWS アカウント 場所にイベントを保存する	a. DLQ AWS アカウントとしてで Amazon SQS キューを選択するを選択します。 b. Amazon SQS キューの Amazon リソースネーム (ARN) を選択します。	
スケジュールを作成する場所 AWS アカウント とは異なる にイベントを保存する	a. DLQ AWS アカウントとして他ので Amazon SQS キューを指定するを選択します。 b. Amazon SQS キューの Amazon リソースネーム (ARN) を入力します。	

- d. カスタマーマネージドキーを使用してターゲットの入力を暗号化するには、[暗号化] で [暗号化設定をカスタマイズする (高度)] を選択します。

このオプションを選択した場合は、既存の KMS キー ARN を入力するか、[AWS KMS key を作成] を選択して AWS KMS コンソールに移動します。EventBridge スケジューラが

保管中のデータを暗号化する方法の詳細については、「Amazon EventBridge スケジューラ ユーザーガイド」の「[保管中の暗号化](#)」を参照してください。

- e. EventBridge スケジューラに新しい実行ロールを作成させるには、[このスケジュールの新しいロールを作成] を選択します。その後、[ロール名] で名前を入力します。このオプションを選択すると、EventBridge スケジューラは、テンプレート化されたターゲットに必要な許可をロールにアタッチします。

10. [次へ] を選択します。

11. [スケジュールの確認と作成] ページで、スケジュールの詳細を確認します。各セクションで、そのステップに戻って詳細を編集するには、[編集] を選択します。

12. [スケジュールを作成] を選択します。

[スケジュール] ページで、新規および既存のスケジュールのリストを表示できます。[ステータス] 列で、新しいスケジュールが [有効] になっていることを確認します。

関連リソース

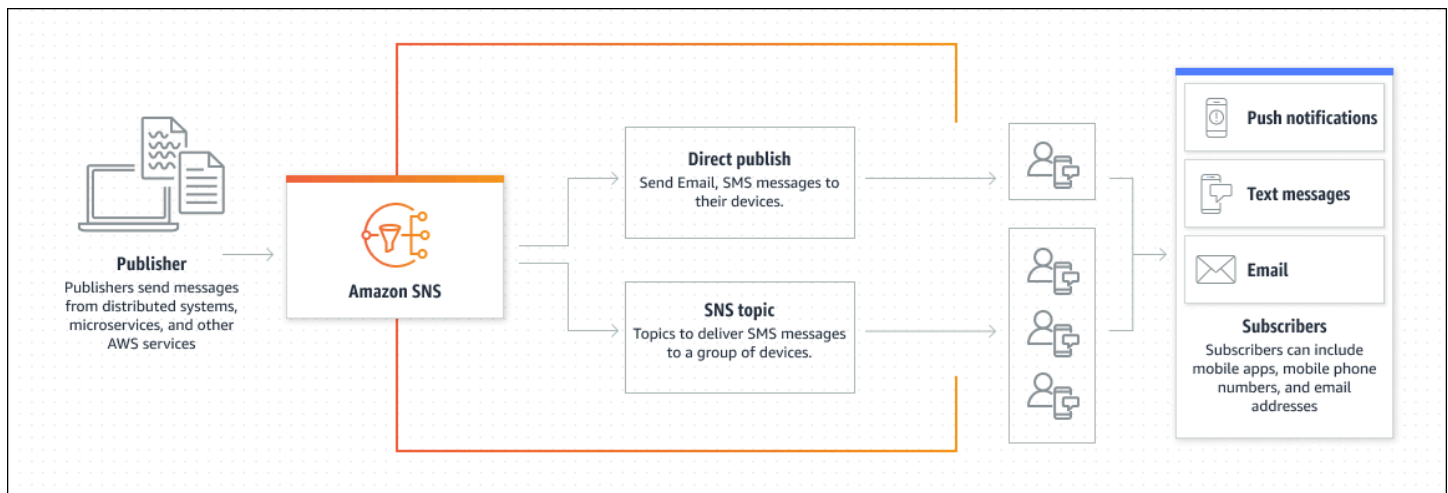
EventBridge スケジューラに関する詳細については、次を参照してください。

- [EventBridge スケジューラユーザーガイド](#)
- [EventBridge スキーマ API リファレンス](#)
- [EventBridge Scheduler Pricing](#)

Amazon SNS を使用した Application-to-Person メッセージング

Amazon SNS Application-to-Person (A2P) メッセージングを使用すると、SMS (ショートメッセージサービス) を介してユーザーのモバイルデバイスに通知とアラートを直接配信できます。この機能を使用すると、モバイルアプリにプッシュ通知を送信したり、携帯電話番号にテキストメッセージを送信したり、E メールアドレスにプレーンテキスト E メールを送信したりできます。さらに、トピックを使用して複数の受信者にメッセージを配信したり、個々のモバイルエンドポイントにメッセージを直接発行してパーソナライズされたコミュニケーションを実現したりできる柔軟性も備えています。

このトピックでは、モバイルアプリケーション、携帯電話番号、E メールアドレスなどのサブスクライバーへのユーザー通知に Amazon SNS を使用する方法について説明します。



Amazon SNS を使用したモバイルテキストメッセージ

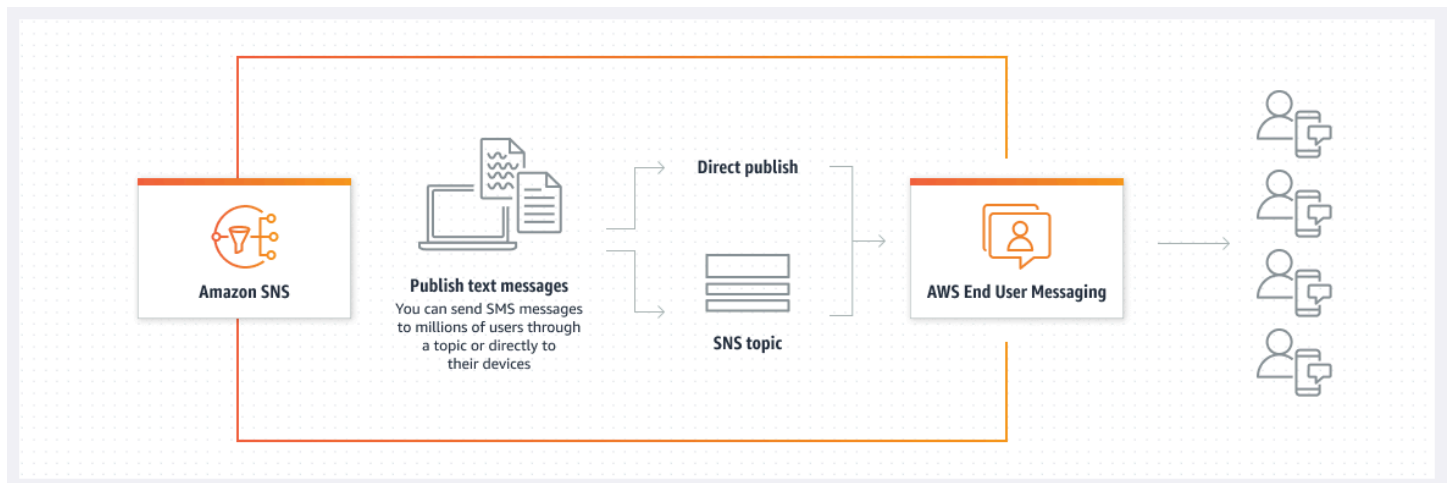
⚠ Important

Amazon SNS SMS デベロッパーガイドが更新されました。Amazon SNS は、SMS メッセージの配信のために [AWS End User Messaging SMS](#) と統合されています。このガイドには、Amazon SNS SMS メッセージを作成、設定、管理する方法に関する最新情報が含まれています。

Amazon SNS モバイルテキストメッセージング (SMS) は、SMS をサポートするウェブ、モバイル、ビジネスアプリケーションなど、さまざまなプラットフォームへのメッセージ配信をサポートするように設計されています。ユーザーは、トピックにサブスクライブすると、1 つまたは複数の電話番号にメッセージを送信でき、配信プロセスを簡素化できます。

Amazon SNS メッセージは によって配信されるため AWS End User Messaging SMS、信頼性の高いメッセージ送信が保証されます。Amazon SNS API では、メッセージタイプ (プロモーションまたはトランザクション)、[毎月の支出制限](#)、[オプトアウトリスト](#)、[メッセージ配信の最適化](#)など、さまざまなプロパティを設定できます。

AWS End User Messaging SMS は、グローバル SMS 供給ネットワークを介して送信先電話番号へのメッセージの送信を処理します。ルーティング、配信ステータス、および地域の規制に必要なコンプライアンスを管理します。詳細なアクセス許可、電話プール、設定セット、SMS シミュレーター、国ルールなど、追加の SMS 機能にアクセスするには、「[AWS End User Messaging SMS ユーザーガイド](#)」を参照してください。



次の主要機能を利用すると、スケラブルで拡張が容易な Amazon SNS SMS メッセージを送信できます。

[メッセージ設定のカスタマイズ](#)

予算とユースケースに基づいて SMS 設定を設定 AWS アカウントして、の SMS 配信をカスタマイズします。例えば、メッセージがコスト効率を優先するか、確実な配信を優先するかを選択できます。

[支出クォータを設定する](#)

SMS 配信をカスタマイズするには、個別のメッセージ配信の支出クォータを設定するか、AWS アカウントの毎月の支出クォータを設定します。現地の法律や規制 (米国やカナダなど) で義務

付けられている場合、SMS 受信者は[オプトアウト](#)できます。つまり、SMS メッセージの受信を停止することを選択できます AWS アカウント。受信者はメッセージの受信をオプトアウトした後、再度電話番号をオプトインして、メッセージの送信を再開できます (ただし制限があります)。

[SMS メッセージをグローバルに送信する](#)

Amazon SNS は、複数のリージョンで SMS メッセージングをサポートしているので、ユーザーは、240 以上の国とリージョンにメッセージを送信できます。

Amazon SNS が SMS メッセージを配信する方法

ユーザーに代わって SMS を送信するように Amazon SNS にリクエストすると、メッセージは AWS End User Messaging SMSを使用してディスパッチされます。Amazon SNS との統合 AWS End User Messaging SMS には、次の利点があります。

[IAM ポリシー](#)

IAM ポリシーとリソースポリシーを活用して、他の AWS サービスやリージョン間で SMS リソースへのアクセスを制御および分散できます。

[AWS End User Messaging SMS 設定](#)

すべての発信元 ID 関連の設定 (作成、設定の更新、新しい発信元 IDs プロビジョニング、登録テンプレートの変更) が、を使用します AWS End User Messaging SMS。

[AWS End User Messaging SMS 請求](#)

ただし、すべての SMS 請求は行われます AWS End User Messaging SMS。SMS リソースを一元的に調達および管理しながら、SMS ワークロードの AWS 支出を統合することができます。

Amazon SNS SMS の開始方法

Important

Amazon SNS SMS デベロッパーガイドが更新されました。Amazon SNS は、SMS メッセージの配信のために [AWS End User Messaging SMS](#) と統合されています。このガイドには、Amazon SNS SMS メッセージを作成、設定、管理する方法に関する最新情報が含まれています。

このトピックでは、SMS サンドボックスを管理し、AWS End User Messaging SMS APIs にアクセスして使用するために必要なアクセス許可を Amazon SNS に付与するように IAM およびリソースベースのポリシーを設定する方法について説明します。

前提条件

Amazon SNS は、IAM ポリシーを更新し、以下のアクションを含めて、Amazon SNS リソースへの包括的な制御と可視性を確保することをお勧めします。

- [AmazonSNSFullAccess](#)
- [AmazonSNSReadOnly](#)

Amazon SNS SMS サンドボックスの使用

新しく作成された Amazon SNS SMS アカウントは、SMS サンドボックスに自動的に配置されます。これにより、不正行為や悪用のリスクを軽減し、AWS ユーザーと受信者の両方のセキュリティを確保します。この環境は、テストおよび開発用の安全なスペースとして機能します。SMS サンドボックスで動作している間は、すべての Amazon SNS 機能にアクセスできますが、特定の制限が適用されます。

- SMS メッセージは、検証済みの送信先電話番号にのみ送信できます。
- 最大 10 個の認証済みの送信先電話番号を持つことができます。
- 送信先電話番号は、検証または最後の検証の試行から 24 時間以上経過した後に削除できるようになります。

アカウントがサンドボックスから移動されると、これらの制限は解除され、SMS メッセージを任意の受信者に送信できます。

最初のステップ

新しい Amazon SNS SMS アカウントは SMS サンドボックスに配置されます。次の手順を使用して、サンドボックスでの電話番号の作成と管理、発信元 ID と送信者 ID の作成、会社の登録を行います。

1. 送信先の電話番号を SMS サンドボックスに追加します。Amazon SNS SMS サンドボックスでの電話番号の追加、管理、移行の詳細については、「[Amazon SNS SMS サンドボックスでの電話番号の追加と確認](#)」を参照してください。

2. SMS メッセージを送信したときに受信者のデバイスに表示される発信元 ID を作成します。発信元 ID に関して、使用できるさまざまなタイプの詳細などについては、[Amazon SNS SMS メッセージの送信元アイデンティティ](#) ドキュメントを参照してください。
3. 会社を登録します。一部の国では、会社の ID を登録して、電話番号や送信者 ID を購入したり、その国の受信者に送信されたメッセージを確認したりできるようにする必要があります。登録が必要な国については、「AWS End User Messaging SMS ユーザーガイド」の「[AWS End User Messaging SMS での SMS メッセージングでサポートされている国と地域](#)」を参照してください。
4. トピックまたは携帯電話にメッセージを送信します。詳細については、「[Amazon SNS を使用して SMS メッセージを送信する](#)」を参照してください。

Amazon SNS SMS サンドボックスでの電話番号の追加と確認

SMS サンドボックス AWS アカウント から SMS メッセージを送信する前に、以下のセットアップ手順を完了する必要があります。[???](#)これにより、アカウントの SMS メッセージングの準備ができ、送信先の電話番号が適切に検証されます。

1. [発信元 ID](#) を作成します。SMS サンドボックス内にないアカウントと同様に、一部の国や地域の受信者に SMS メッセージを送信するには、事前に発信元 ID が必要になります。
2. メッセージを送信する送信先の電話番号を SMS サンドボックス内に追加します。
3. 電話番号を検証して、送信先の電話番号が SMS メッセージで使えることを確認します。

送信先の電話番号を追加および確認する

1. [Amazon SNS コンソール](#) にサインインします。
2. コンソールメニューで、[\[SMS メッセージングをサポートしているリージョン\]](#) を選択します。
3. ナビゲーションペインで、[テキストメッセージング (SMS)] を選択します。
4. [サンドボックスの送信先電話番号] セクションで、[電話番号の追加] を選択します。
5. [送信先の詳細] で、次の情報を入力し、[電話番号の追加] を選択します。
 - 送信先の [国コード] と [電話番号]。
 - 検証メッセージを送信する [言語]。
6. 電話番号を追加すると、Amazon SNS は指定された送信先の電話番号に OTP を送信します。この OTP は検証に必要です。
7. OTP は、指定した送信先の電話番号に標準の SMS メッセージとして届きます。

- 15 分以内に OTP を受信しない場合は、Amazon SNS コンソールで [検証コードを再送信] を選択します。
 - OTP は 24 時間で最大 5 回再送信できます。
8. OTP を受け取ったら、[検証コード] ボックスに入力し、[電話番号を検証] を選択します。
 9. 検証ステータスを確認します。
 - 電話番号が正常に検証されると、電話番号とその検証ステータスが [サンドボックスの送信先電話番号] セクションに表示されます。
 - ステータスが [保留中] の場合、検証は失敗しています。これは、例えば、国コードを正しく入力しなかった場合に発生する可能性があります。
 - 保留中や検証済みの電話番号を削除できるのは、前回の検証の試行から 24 時間以上経過後に限られます。
 10. 他の地域で同じ送信先の電話番号を使用する場合は、使用する地域ごとに上記の手順を繰り返します。

OTP テキストの非受信のトラブルシューティング

電話番号での OTP テキストの受信を妨げる可能性がある一般的な問題をトラブルシューティングします。

- Amazon SNS SMS 支出制限: AWS アカウント で SMS メッセージ送信の支出制限を超えた場合、メッセージ (OTP テキストを含む) は、制限が引き上げられるか、請求の問題が解決されるまで配信されない可能性があります。
- 電話番号が SMS 通知にオプトインされていない: 一部の国や地域では、受信者はオプトインを行って、ショートコード (OTP テキストに通常使用されます) から SMS メッセージを受信できるようにする必要があります。受信者の電話番号がオプトインされていない場合、OTP テキストは受信されません。
- 通信事業者の制限またはフィルタリング: 一部のモバイル通信事業者には、特定のタイプの SMS メッセージ (OTP テキストを含む) の配信を妨げる制限またはフィルタリングメカニズムがあります。これは、通信事業者によって実装されたセキュリティポリシーまたはスパム対策が原因である可能性があります。
- 無効または不正な電話番号: 受信者が提供した電話番号が不正確または無効な場合、OTP テキストは配信されません。
- ネットワークの問題: 一時的なネットワークの問題や障害により、SMS メッセージ (OTP テキストを含む) が受信者の電話に配信されない可能性があります。

- 配信の遅延: 場合によっては、ネットワークの輻輳やその他の要因により、SMS メッセージの配信が遅延することがあります。OTP テキストは最終的には配信される可能性がありますが、想定以上に遅延する場合があります。
- アカウントの停止または終了: 不払いや AWS 利用規約違反など AWS アカウント、に問題がある場合、OTP テキストを含む Amazon SNS メッセージング機能は停止または終了される可能性があります。

Amazon SNS SMS サンドボックスで電話番号を削除する

[SMS サンドボックス](#)で、検証待ちまたは検証済みの送信先電話番号を削除できます。

Important

[電話番号の検証](#)を行ってから 24 時間後、または最後の検証の試行の 24 時間後に、電話番号を削除できます。

SMS サンドボックスから送信先の電話番号を削除するには

1. [Amazon SNS コンソール](#)にサインインします。
2. コンソールメニューで、送信先電話番号を追加した、[\[SMS メッセージングをサポートしているリージョン\]](#)を選択します。
3. ナビゲーションペインで、[テキストメッセージング (SMS)] を選択します。
4. モバイルテキストメッセージ (SMS) ページで、サンドボックスの送信先電話番号セクションに移動します。
5. 削除する特定の電話番号を選択し、[電話番号の削除] を選択します。
6. 送信先電話番号の削除を確定するには、「**delete me**」と入力し、[削除] を選択します。

送信先電話番号の検証または検証の試行から 24 時間以上経過していることを確認してから、削除を進めます。

7. 追加したけれどももう使わない送信先電話番号を追加したリージョンごとに、上記のステップを繰り返します。

Amazon SNS SMS サンドボックス外への移動

を [SMS サンドボックス](#) AWS アカウント の外に移動するには、まず送信先の電話番号を追加、検証、テストする必要があります。これを行ったら、 を使用してケースを作成します AWS サポート。

AWS アカウントを SMS サンドボックス外に移動するようにリクエストするには

1. 電話番号を検証する

- a. AWS アカウント が SMS サンドボックスにある間に、[Amazon SNS コンソール](#)を開きます。
- b. ナビゲーションペインの [モバイル] の下で、[テキストメッセージング (SMS)] を選択します。
- c. サンドボックスの送信先電話番号セクションで、1 つまたは複数の送信先電話番号を[追加して検証](#)します。この検証により、メッセージを正常に送受信できるようになります。

2. SMS 発行のテスト

- 少なくとも 1 つの検証済みの電話番号で、メッセージを送受信できることを確認します。SMS メッセージの発行方法の詳細については、「[Amazon SNS を使用して SMS メッセージを携帯電話に発行する](#)」を参照してください。

3. サンドボックス編集を開始する

- 「Amazon SNS コンソールのモバイルテキストメッセージング (SMS)」のページの [アカウント情報] で、[SMS サンドボックスを終了する] を選択します。。このアクションにより、[Amazon サポートセンター](#)にリダイレクトされ、Service Quotas の引き上げオプションが選択されたサポートケースが自動的に作成されます。

4. フォームへの入力

- 「Service Quotas の引き上げ」のサポートフォームで、以下を実行します。
 - i. サービスとして [SNS テキストメッセージング] を選択します。
 - ii. SMS メッセージを送信するウェブサイト URL またはアプリ名を指定します。
 - iii. 送信するメッセージのタイプとして、ワンタイムパスワード、プロモーション、トランザクションのいずれかを選択します。
 - iv. SMS メッセージの送信元の AWS リージョン を選択します。
 - v. SMS メッセージの送信先の国または地域を指定します。

- vi. お客様がメッセージの受信をオプトインする方法の説明を入力します。
- vii. 使用するメッセージテンプレートを含めます。

5. クォータとリージョンを指定する

- [リクエスト]で、以下の操作を行います。
 - i. を移動する AWS リージョン を選択します AWS アカウント。
 - ii. [リソースタイプ] で、[一般的な制限] を選択します。
 - iii. [クォータ] で、[SMS サンドボックスを終了] を選択します。
 - iv. (オプション) 追加の引き上げやその他の調整をリクエストするには、[別のリクエストを追加] を選択して、必要な詳細を指定します。
 - v. [申請する上限数] に、リクエストするクォータを米ドルで入力します。

6. その他の詳細

- a. [ケースの説明] に、リクエストに関連する追加の詳細を入力します。
- b. [お問い合わせオプション] で、[連絡の際に希望する言語] を選択します。

7. リクエストを送信する

- [送信] を選択して、リクエストを サポート に送信します。

サポート チームは 24 時間以内にリクエストに最初の応答を提供します。

迷惑なコンテンツや悪意のあるコンテンツを送信するためにシステムが悪用されないように、私たちは各リクエストを慎重に検討しています。可能であれば、24 時間以内にリクエストを承認します。ただし、追加情報が必要な場合は、お客様のリクエストの解決に時間がかかる場合があります。

お客様のユースケースが私たちのポリシーに一致しない場合は、リクエストを承認できない場合があります。

Amazon SNS SMS メッセージの送信元アイデンティティ

Important

Amazon SNS SMS デベロッパーガイドが更新されました。Amazon SNS は、SMS メッセージの配信のために [AWS End User Messaging SMS](#) と統合されています。このガイドに

は、Amazon SNS SMS メッセージを作成、設定、管理する方法に関する最新情報が含まれています。

SMS メッセージの送信元アイデンティティは、SMS メッセージの送信者を表すために使用される識別子です。次のタイプの送信元アイデンティティを使用して受信者に対しお客様の身元証明ができます。

送信元番号

SMS メッセージの送信者の電話番号を識別する数値文字列です。送信元番号には、ロングコード (通常 10 桁以上の標準電話番号)、10 桁のロングコード (10DLC)、通話料無料番号 (TFN)、ショートコード (4～7 桁の電話番号) など、いくつかのタイプがあります。

送信元番号のサポートは、送信元番号ではなく送信者 ID の使用が現地法により義務付けられている国では利用できません。送信元番号を使用して SMS メッセージを送信すると、受信者のデバイスには送信元番号が送信者の電話番号として表示されます。ユースケースごとに異なる送信元番号を指定できます。

詳細については、「AWS End User Messaging SMS ユーザーガイド」の「[電話番号](#)」を参照してください。

Tip

AWS アカウント内のすべての既存の発信元番号のリストを表示するには、[Amazon SNS コンソール](#)のナビゲーションペインで、発信元番号を選択します。

送信者 IDs

SMS メッセージの送信者を識別する英字名です。送信者 ID を使用して SMS メッセージを送信し、送信者 ID 認証がサポートされているエリアに受信者がいる場合は、電話番号ではなく、送信者 ID が受信者のデバイスに表示されます。送信者 ID は、電話番号、ロングコード、ショートコードよりも、送信者に関する詳細な情報を SMS の受信者に示すことができます。

送信者 ID は、世界の一部の国や地域でサポートされています。一部の地域では、企業として個別のお客様に SMS メッセージを送信する場合は、事前に規制当局または業界団体に登録されている送信者 ID を使用する必要があります。送信者 ID をサポートまたは必要とする国と地域の完全なリストについては、「AWS End User Messaging SMS ユーザーガイド」の「[AWS End](#)

[User Messaging SMSの SMS メッセージングがサポートされている国と地域](#)」を参照してください。

送信者 ID の使用には追加料金はかかりません。ただし、送信者 ID 認証のサポートや要件は国によって異なります。主要な市場 (例: カナダ、中国、米国) の中には、送信者 ID の使用がサポートされていないものがあります。一部の地域では、企業として個人のお客様に SMS メッセージを送信する場合は、規制当局または業界団体に事前登録されている送信者 ID を使用する必要があります。

詳細については、「AWS End User Messaging SMS ユーザーガイド」の「[送信者 ID](#)」を参照してください。

Amazon SNS での SMS メッセージングの設定

Important

Amazon SNS SMS デベロッパーガイドが更新されました。Amazon SNS は、SMS メッセージの配信のために [AWS End User Messaging SMS](#) と統合されています。このガイドには、Amazon SNS SMS メッセージを作成、設定、管理する方法に関する最新情報が含まれています。

Amazon SNS SMS の設定を使用して、使用クォータの調整や配信ステータスのログ記録の設定など、要件に合わせて SMS プリファレンスの設定を行うことができます。このトピックでは、Amazon SNS コンソールと AWS SDK を使用してトピックに SMS メッセージを発行する方法、クォータを効率的に処理する方法、SMS アクティビティに関する詳細な統計を取得する方法についても詳しく説明します。

Amazon SNS を使用して SMS メッセージを送信する

このセクションでは、Amazon SNS を使用して SMS メッセージを送信する方法について説明します。ここでは、トピックへの発行、トピックへの電話番号のサブスクライブ、メッセージの属性の設定、携帯電話への直接発行などについて説明します。

SMS メッセージを Amazon SNS トピックに発行する

Amazon SNS トピックにこれらの電話番号を登録することで、1 通の SMS メッセージを一度に複数の電話番号に送信できます。SNS トピックは通信チャンネルで、受信者を追加し、すべての受信者にメッセージを発行できます。サブスクライバーは、サブスクリプションをキャンセルするか、サブ

スクライバーが AWS アカウントからの SMS メッセージの受信をオプトアウトするまで、トピックに発行されたすべてのメッセージを受信します。

AWS コンソールを使用してトピックにメッセージを送信する

トピックを作成する

SMS メッセージを送信するトピックがまだない場合は、次の手順を実行します。

1. [Amazon SNS コンソール](#)にサインインします。
2. コンソールメニューで、[\[SMS メッセージングをサポートしているリージョン\]](#) を選択します。
3. ナビゲーションペインで、[トピック] を選択してください。
4. [トピック] ページで、[トピックの作成] を選択します。
5. [トピックの作成] ページの [詳細] で、次の操作を行います。
 - a. [Type] (タイプ) で、[Standard] (標準) を選択します。
 - b. [名前] にトピック名を入力します。
 - c. (オプション) [表示名] に SMS メッセージのカスタムプレフィックスを入力します。トピックにメッセージを送信する場合、Amazon SNS によって右アングルブラケット (>) とスペースに続いて表示名が付加されます。表示名では大文字と小文字が区別されず、Amazon SNS は表示名を大文字に変換します。例えば、トピックの表示名が MyTopic で、メッセージが Hello World! である場合、メッセージは次のように表示されます。

```
MYTOPIC> Hello World!
```

6. [トピックを作成] を選択します。トピックの名前と Amazon リソースネーム (ARN) が、[トピック] ページに表示されます。

SMS サブスクリプションを作成するには

トピックに 1 回だけメッセージを発行することによって、サブスクリプションを使用して SMS メッセージを複数の受信者に送信できます。

Note

Amazon SNS を使用して SMS メッセージを送信し始めると、AWS アカウントは SMS サンドボックスにあります。SMS サンドボックスは、SMS 送信者としての評判を損なうことなく、Amazon SNS 機能を試すための安全な環境を提供します。アカウントが SMS サンド

ボックスにあり、Amazon SNS のすべての機能を使用できますが、SMS メッセージを送信できるのは、確認済みの送信先の電話番号だけです。詳細については、「[Amazon SNS SMS サンドボックスの使用](#)」を参照してください。

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションペインで [サブスクリプション] を選択します。
3. [サブスクリプション] ページで [サブスクリプションの作成] を選択します。
4. [サブスクリプションの作成] ページで [詳細] セクションで、以下を実行します。
 - a. トピック ARNで、SMS メッセージを送信するトピックの Amazon リソースネーム (ARN) を入力または選択します。
 - b. [プロトコル] で SMS を選択します。
 - c. [エンドポイント] に、トピックをサブスクライブする電話番号を入力します。
5. [サブスクリプションの作成] を選択します。サブスクリプション情報は [サブスクリプション] ページに表示されます。

さらに電話番号を追加するには、このステップを繰り返します。電子メールなど、他の種類のサブスクリプションを追加することもできます。

単一のメッセージを送信するには

トピックにメッセージを発行すると、Amazon SNS はトピックにサブスクライブされているすべての電話番号にメッセージの配信を試みます。

1. [Amazon SNS コンソール](#) の [トピック] ページで、SMS メッセージを送信するトピックの名前を選択します。
2. トピックの詳細ページで、[トピックへの発行] を選択します。
3. [トピックへのメッセージの発行] ページの [メッセージの詳細] で、次の操作を行います。
 - a. トピックに E メールサブスクリプションが含まれていない場合、および E メールと SMS のサブスクリプションの両方を発行する場合は、[件名] のフィールドを空のままにします。Amazon SNS の [件名] は、Eメールの件名の欄とにあたります。
 - b. (オプション) [有効期限 (TTL)] に、Amazon SNS がモバイルアプリケーションエンドポイント受信者に SMS メッセージを送信するのに必要な秒数を入力します。
4. [メッセージ本文] で、次の作業を行います。

- a. [メッセージ構造] で、[すべての配信プロトコルに同一のペイロード] をクリックして、トピックに登録されているすべてのプロトコルタイプに同じメッセージを送信します。または、[配信プロトコルごとにカスタムペイロード] を選択して、異なるプロトコルタイプの受信者用のメッセージをカスタマイズします。例えば、電話番号受信者にはデフォルトメッセージを入力し、E メール受信者にはカスタムメッセージを入力できます。
- b. [エンドポイントに送信するメッセージ本文] を選択し、メッセージを入力するか、配信プロトコルごとにカスタムメッセージを入力します。

トピックに表示名がある場合、Amazon SNS はそれをメッセージの長さを増やすメッセージに追加します。表示名の長さは、名前の文字数に Amazon SNS が追加する右アングルブラケット (>) とスペースの 2 文字をプラスしたものです。

SMS メッセージのサイズ限度の詳細については、「[Amazon SNS を使用して SMS メッセージを携帯電話に発行する](#)」を参照してください。

5. (オプション) [メッセージ属性] で、タイムスタンプ、署名、ID などのメッセージメタデータを追加します。
6. [メッセージの発行] を選択します。Amazon SNS から SMS メッセージを送信し、成功のメッセージを表示します。

AWS SDKs

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、『AWS SDK とツールのリファレンスガイド』の「[共有設定ファイルおよび認証情報ファイル](#)」を参照してください。

次のコードの例は以下の方法を示しています。

- Amazon SNS トピックを作成します。
- 携帯電話番号をトピックにサブスクライブする。
- SMS メッセージをトピックに発行して、登録されているすべての電話番号がメッセージを一度に受信できるようにします。

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。

トピックを作成し、その ARN を返します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class CreateTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicName>

            Where:
                topicName - The name of the topic to create (for example,
mytopic).

            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
    }
}
```

```
String topicName = args[0];
System.out.println("Creating a topic with name: " + topicName);
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();

String arnVal = createSNSTopic(snsClient, topicName);
System.out.println("The topic ARN is" + arnVal);
snsClient.close();
}

public static String createSNSTopic(SnsClient snsClient, String topicName) {
    CreateTopicResponse result;
    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
            .name(topicName)
            .build();

        result = snsClient.createTopic(request);
        return result.topicArn();

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
}
```

トピックへのエンドポイントのサブスクライブ。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * 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
*/
public class SubscribeTextSMS {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicArn> <phoneNumber>

            Where:
                topicArn - The ARN of the topic to subscribe.
                phoneNumber - A mobile phone number that receives
notifications (for example, +1XXX5550100).
            "";

        if (args.length < 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        subTextSNS(snsClient, topicArn, phoneNumber);
        snsClient.close();
    }

    public static void subTextSNS(SnsClient snsClient, String topicArn, String
phoneNumber) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                .protocol("sms")
                .endpoint(phoneNumber)
                .returnSubscriptionArn(true)
                .topicArn(topicArn)
                .build();

            SubscribeResponse result = snsClient.subscribe(request);
        }
    }
}
```

```

        System.out.println("Subscription ARN: " + result.subscriptionArn() +
"\n\n Status is "
        + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

送信者の ID、上限価格、タイプなど、メッセージ上の属性を設定します。メッセージ属性はオプションです。

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.HashMap;

/**
 * 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
 */
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }
}

```

```

    public static void setSNSAttributes(SnsClient snsClient, HashMap<String,
String> attributes) {
        try {
            SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
                .attributes(attributes)
                .build();

            SetSmsAttributesResponse result =
snsClient.setSMSAttributes(request);
            System.out.println("Set default Attributes to " + attributes + ".
Status was "
                + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}

```

トピックへのメッセージの発行 メッセージは、すべての受信者に送信されます。

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class PublishTextSMS {
    public static void main(String[] args) {
        final String usage = ""

```



```
Usage:    <message> <phoneNumber>

Where:
    message - The message text to send.
    phoneNumber - The mobile phone number to which a message is
sent (for example, +1XXX5550100).\s
    """;

if (args.length != 2) {
    System.out.println(usage);
    System.exit(1);
}

String message = args[0];
String phoneNumber = args[1];
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();
pubTextSMS(snsClient, message, phoneNumber);
snsClient.close();
}

public static void pubTextSMS(SnsClient snsClient, String message, String
phoneNumber) {
    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .phoneNumber(phoneNumber)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out
            .println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

Amazon SNS を使用して SMS メッセージを携帯電話に発行する

Amazon SNS を使用して、Amazon SNS トピックに電話番号を登録せずに、SMS メッセージを携帯電話に直接送信できます。

Note

1 つのメッセージを同時に複数の電話番号に送信する場合、電話番号をトピックにサブスクライブすると便利です。SMS メッセージをトピックに発行する方法については、「[SMS メッセージを Amazon SNS トピックに発行する](#)」を参照してください。

メッセージを送信するとき、メッセージがコストまたは信頼性に対して最適化されているかどうかを制御できます。[送信者 ID または発信番号](#)を指定することもできます。Amazon SNS API または AWS SDKs を使用してプログラムでメッセージを送信する場合は、メッセージ配信の上限価格を指定できます。

各 SMS メッセージは最大 140 バイトまで含めることができ、文字限度はエンコーディングスキームによって異なります。例えば、SMS メッセージには以下を含めることができます。

- 160 GSM 文字
- 140 ASCII 文字
- 70 UCS-2 文字

サイズ限度を超えてメッセージを発行する場合は、Amazon SNS により、複数のメッセージとして送信され、それぞれが文字数の限度以内に収められます。メッセージは単語の途中ではなく、単語の境目で切り離されます。1 回の SMS 発行アクションの合計サイズ限度は、1,600 バイトです。

SMS メッセージを送信するときは、E.164 形式を使用して電話番号を指定します。E.164 形式は国際電気通信に使用される標準電話番号付与構造です。この形式に従う電話番号は最大 15 桁を設定でき、プラス記号 (+) および国コードのプレフィックスが付いています。例えば、E.164 形式の米国の電話番号は +1XXX5550100 と表示されます。

メッセージの送信 (コンソール)

1. [Amazon SNS コンソール](#)にサインインします。
2. コンソールメニューで、[\[SMS メッセージングをサポートしているリージョン\]](#) を選択します。
3. ナビゲーションペインで、[テキストメッセージング (SMS)] を選択します。

4. リポジトリの [モバイルテキストメッセージング (SMS)] ページで、[テキストメッセージの発行]。
5. リポジトリの [SMS メッセージを発行する] ページ、[メッセージの種類] で、次のいずれかを選択します。
 - プロモーション - マーケティングメッセージなどの非クリティカルなメッセージ。
 - [トランザクション] - 多要素認証のワンタイムパスコードなど、顧客のトランザクションをサポートするクリティカルメッセージ。

 Note

このメッセージレベルの設定をすると、アカウントレベルのデフォルトメッセージタイプは上書きされます。アカウントレベルのデフォルトのメッセージタイプは、[モバイルテキストメッセージング (SMS)] ページの [テキストメッセージングの優先設定] のセクションで設定できます。

プロモーションおよびトランザクションメッセージの料金表の詳細については、「[世界各地の SMS 料金表](#)」を参照してください。

6. [送信先電話番号] に、メッセージを送信する電話番号を入力します。
7. [メッセージ] に、送信するメッセージを入力します。
8. (オプション) [送信元アイデンティティ] で、受信者を識別する方法を指定します。
 - (オプション) [送信者 ID] に、少なくとも 1 つの文字を含み、スペースは含まない、3~11 文字の英数字のカスタム ID を入力します。送信者 ID は、受信側デバイスにメッセージ送信者として表示されます。例えば、メッセージソースを識別しやすいように、ビジネスブランドを使用できます。

送信者 ID のサポートについては、国、リージョン、またはその両方によって異なります。例えば、米国の電話番号へ配信されるメッセージは、送信者 ID を表示しません。送信者 ID をサポートしている国と地域については、「AWS End User Messaging SMS ユーザーガイド」の「[AWS End User Messaging SMS の SMS メッセージングがサポートされている国と地域](#)」を参照してください。

送信者 ID を指定しない場合、次のいずれか 1 つが送信元 ID として表示されます。

- ロングコードをサポートしている国では、ロングコードが表示されます。

- 送信者 ID のみがサポートされている国では、「NOTICE」が表示されます。

このメッセージレベルの送信者 ID は、[テキストメッセージプリファレンス] ページで設定するデフォルトの送信者 ID を上書きします。

- [送信元番号] を指定するには、受信者のデバイス上で送信者の電話番号として表示する 5～14 の数字の文字列を入力します。この文字列は、AWS アカウント 送信先の国の で設定されている送信元番号と一致する必要があります。発信番号には、10DLC 番号、通話料無料の番号、個人間のロングコード、またはショートコードを指定できます。詳細については、「[Amazon SNS SMS メッセージの送信元アイデンティティ](#)」を参照してください。

発信番号を指定しない場合、AWS アカウント 設定に基づき、Amazon SNS は SMS テキストメッセージに使用する送信元番号を選択します。

9. インドの受取人に SMS メッセージを送信する場合は、[国固有の属性] を展開し、次の属性を指定します。

- エンティティ ID - インドの受信者に SMS メッセージを送信するためのエンティティ ID または プリンシパルエンティティ (PE) ID。この ID は、1 ～ 50 文字の一意の文字列で、Telecom Regulatory Authority of India (TRAI) が提供する、TRAI に登録したエンティティを識別するものです。
- テンプレート ID - インドの受信者に SMS メッセージを送信するためのテンプレート ID。この ID は、TRAI に登録したテンプレートを識別する 1 ～ 50 文字の一意の文字列です。テンプレート ID は、メッセージに対して指定した送信者 ID に関連付ける必要があります。

インドの受信者に SMS メッセージを送信する方法の詳細については、「AWS End User Messaging SMS ユーザーガイド」の「[インドの送信者 ID 登録プロセス](#)」を参照してください。

10. [メッセージの発行] を選択します。

Tip

送信元番号から SMS メッセージを送信するには、Amazon SNS コンソールのナビゲーションパネルで送信元番号を送信します。[機能] 列で [SMS] 含む送信元番号を選択し、次に [テキストメッセージの発行] を選択します。

メッセージの送信 (AWS SDKs)

AWS SDKs のいずれかを使用して SMS メッセージを送信するには、Amazon SNS API の Publish リクエストに対応する SDK の API オペレーションを使用します。このリクエストでは、電話番号に SMS メッセージを直接送信できます。次の属性名の値を設定する場合、MessageAttributes パラメータも使用できます。

AWS.SNS.SMS.SenderID

少なくとも 1 つの文字を含み、スペースは含まない、3 ~ 11 文字の英数字のカスタム ID。送信者 ID は、受信デバイスにメッセージ送信者として表示されます。例えば、メッセージ ソースを識別しやすいように、ビジネスブランドを使用できます。

送信者 ID のサポートについては、国またはリージョンによって異なります。例えば、米国の電話番号へ配信されるメッセージは、送信者 ID を表示しません。送信者 ID をサポートする国または地域のリストについては、「AWS End User Messaging SMS ユーザーガイド」での「[AWS End User Messaging SMS の SMS メッセージングがサポートされている国と地域](#)」を参照してください。

送信者 ID を指定しない場合、サポートされている国またはリージョンでは、メッセージは送信者 ID として [ロングコード](#) を表示します。アルファベットの送信者 ID を必要とする国またはリージョンでは、送信者 ID は「NOTICE」と表示されます。

このメッセージレベルの属性は、SetSMSAttributes リクエストを使用して設定する、アカウントレベルの属性 DefaultSenderId を上書きします。

AWS.MM.SMS.OriginationNumber

5 ~ 14 個の数値のカスタム文字列で、オプションの先頭にプラス記号 (+) を付けられます。この番号文字列は、受信側のデバイスの送信者の電話番号として表示されます。文字列は、送信先の国の AWS アカウントで設定された送信元番号と一致する必要があります。発信番号には、10DLC 番号、通話料無料の番号、Person-to-Person (P2P) のロングコード、またはショートコードを指定できます。詳細については、「AWS End User Messaging SMS ユーザーガイド」の「[電話番号](#)」を参照してください。

発信元番号を指定しない場合、Amazon SNS は AWS アカウント設定に基づいて発信元番号を選択します。

AWS.SNS.SMS.MaxPrice

SMS メッセージの送信に使用できる上限価格 (USD)。Amazon SNS がメッセージの送信により上限価格を超えるコストが発生すると判断した場合、Amazon SNS はメッセージを送信しません。

過去 1 か月の SMS コストが既に `MonthlySpendLimit` 属性の限度を超えた場合、この属性は何も実行しません。`SetSMSAttributes` リクエストを使用して `MonthlySpendLimit` 属性を設定できます。

Amazon SNS トピックにメッセージを送信する場合、トピックにサブスクライブされている各電話番号への各メッセージの配信に上限価格が適用されます。

AWS.SNS.SMS.SMSType

送信するメッセージのタイプ。

- **Promotional** (デフォルト) - マーケティングメッセージなどの非クリティカルなメッセージ。
- **Transactional** - 多要素認証のワンタイムパスコードなど、顧客のトランザクションをサポートするクリティカルメッセージ。

このメッセージレベルの属性は、`SetSMSAttributes` リクエストを使用して設定する、アカウントレベルの属性 `DefaultSMSType` を上書きします。

AWS.MM.SMS.EntityId

この属性は、インドの受信者に SMS メッセージを送信するときのみ必要です。

これは、インドの受信者に SMS メッセージを送信するためのエンティティ ID またはプリンシパルエンティティ (PE) ID です。この ID は、1 ~ 50 文字の一意の文字列で、Telecom Regulatory Authority of India (TRAI) が提供する、TRAI に登録したエンティティを識別するものです。

AWS.MM.SMS.TemplateId

この属性は、インドの受信者に SMS メッセージを送信するときのみ必要です。

これは、インドの受信者に SMS メッセージを送信するためのテンプレートです。この ID は、TRAI に登録したテンプレートを識別する 1 ~ 50 文字の一意の文字列です。テンプレート ID は、メッセージに対して指定した送信者 ID に関連付ける必要があります。

メッセージを送信する

次のコード例は、Amazon SNS を使用して SMS メッセージを発行する方法を示しています。

.NET

SDK for .NET

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。

```
namespace SNSMessageExample
{
    using System;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    public class SNSMessage
    {
        private AmazonSimpleNotificationServiceClient snsClient;

        /// <summary>
        /// Initializes a new instance of the <see cref="SNSMessage"/> class.
        /// Constructs a new SNSMessage object initializing the Amazon Simple
        /// Notification Service (Amazon SNS) client using the supplied
        /// Region endpoint.
        /// </summary>
        /// <param name="regionEndpoint">The Amazon Region endpoint to use in
        /// sending test messages with this object.</param>
        public SNSMessage(RegionEndpoint regionEndpoint)
        {
            snsClient = new
AmazonSimpleNotificationServiceClient(regionEndpoint);
        }

        /// <summary>
        /// Sends the SMS message passed in the text parameter to the phone
        number
        /// in phoneNum.
        /// </summary>
        /// <param name="phoneNum">The ten-digit phone number to which the text
```

```
/// message will be sent.</param>
/// <param name="text">The text of the message to send.</param>
/// <returns>Async task.</returns>
public async Task SendTextMessageAsync(string phoneNum, string text)
{
    if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
    {
        return;
    }

    // Now actually send the message.
    var request = new PublishRequest
    {
        Message = text,
        PhoneNumber = phoneNum,
    };

    try
    {
        var response = await snsClient.PublishAsync(request);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error sending message: {ex}");
    }
}
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[Publish](#)」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。

```
/**
 * Publish SMS: use Amazon Simple Notification Service (Amazon SNS) to send an
 * SMS text message to a phone number.
 * Note: This requires additional AWS configuration prior to running example.
 *
 * NOTE: When you start using Amazon SNS to send SMS messages, your AWS account
 * is in the SMS sandbox and you can only
 * use verified destination phone numbers. See https://docs.aws.amazon.com/sns/
 * latest/dg/sns-sms-sandbox.html.
 * NOTE: If destination is in the US, you also have an additional restriction
 * that you have use a dedicated
 * origination ID (phone number). You can request an origination number using
 * Amazon Pinpoint for a fee.
 * See https://aws.amazon.com/blogs/compute/provisioning-and-using-10dlc-
 * origination-numbers-with-amazon-sns/
 * for more information.
 *
 * <phone_number_value> input parameter uses E.164 format.
 * For example, in United States, this input value should be of the form:
 * +12223334444
 */

//! Send an SMS text message to a phone number.
/*!
 \param message: The message to publish.
 \param phoneNumber: The phone number of the recipient in E.164 format.
 \param clientConfiguration: AWS client configuration.
 \return bool: Function succeeded.
 */
bool AwsDoc::SNS::publishSms(const Aws::String &message,
                             const Aws::String &phoneNumber,
```

```
const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::PublishRequest request;
    request.SetMessage(message);
    request.SetPhoneNumber(phoneNumber);

    const Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Message published successfully with message id, '"
                    << outcome.GetResult().GetMessageId() << "'."
                    << std::endl;
    }
    else {
        std::cerr << "Error while publishing message "
                    << 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「[Publish](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
```

```
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class PublishTextSMS {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <message> <phoneNumber>

            Where:
                message - The message text to send.
                phoneNumber - The mobile phone number to which a message is
sent (for example, +1XXX5550100).\s
            """;

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String message = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();
        pubTextSMS(snsClient, message, phoneNumber);
        snsClient.close();
    }

    public static void pubTextSMS(SnsClient snsClient, String message, String
phoneNumber) {
        try {
            PublishRequest request = PublishRequest.builder()
                .message(message)
                .phoneNumber(phoneNumber)
```

```
        .build();

        PublishResponse result = snsClient.publish(request);
        System.out
            .println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[Publish](#)」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 pubTextSMS(
    messageVal: String?,
    phoneNumberVal: String?,
) {
    val request =
        PublishRequest {
            message = messageVal
            phoneNumber = phoneNumberVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

```
}  
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[Publish](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';  
  
use Aws\Exception\AwsException;  
use Aws\Sns\SnsClient;  
  
/**  
 * Sends a text message (SMS message) directly to a phone number using Amazon  
 * SNS.  
 *  
 * This code expects that you have AWS credentials set up per:  
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/  
 * guide_credentials.html  
 */  
  
$snsClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);  
  
$message = 'This message is sent from a Amazon SNS code sample.';  
$phone = '+1XXX5550100';
```

```
try {
    $result = $SnSClient->publish([
        'Message' => $message,
        'PhoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[発行](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def publish_text_message(self, phone_number, message):
        """
        Publishes a text message directly to a phone number without need for a
        subscription.
        """
```

```

        :param phone_number: The phone number that receives the message. This
        must be
                                in E.164 format. For example, a United States phone
                                number might be +12065550101.
        :param message: The message to send.
        :return: The ID of the message.
        """
        try:
            response = self.sns_resource.meta.client.publish(
                PhoneNumber=phone_number, Message=message
            )
            message_id = response["MessageId"]
            logger.info("Published message to %s.", phone_number)
        except ClientError:
            logger.exception("Couldn't publish message to %s.", phone_number)
            raise
        else:
            return message_id

```

- APIの詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[Publish](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```

" iv_phone_number = '+12065550101' - Phone number in E.164 format
TRY.
    oo_result = lo_sns->publish(
        " oo_result is returned for
testing purposes. "
        iv_phonenumber = iv_phone_number
        iv_message = iv_message ).

```

```
MESSAGE 'Message published to phone number.' TYPE 'I'.  
CATCH /aws1/cx_snsnotfoundexception.  
MESSAGE 'Phone number does not exist.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[Publish](#)」(発行) を参照してください。

Amazon SNS の SMS メッセージプリファレンスを設定する

Amazon SNS を使用して、SMS メッセージングのプリファレンスを指定します。例えば、配信をコストまたは信頼性に対して最適化するかや、毎月の使用量の上限、配信がログに記録される方法、SMS の毎日の使用状況レポートをサブスクライブするかどうか、などを指定できます。

これらの設定は、アカウントから送信するすべての SMS メッセージに対して有効になりますが、個々のメッセージの送信時に上書きすることができます。詳細については、「[Amazon SNS を使用して SMS メッセージを携帯電話に発行する](#)」を参照してください。

を使用した SMS メッセージングの設定 AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. [SMS メッセージングをサポートしているリージョン](#)を選択します。
3. ナビゲーションパネルで、[モバイル]、[テキストメッセージング (SMS)] の順に選択します。
4. [モバイルテキストメッセージング (SMS)] ページの [テキストメッセージプリファレンス] セクションで、[編集] を選択します。
5. [テキストメッセージングの優先設定の編集] ページの [詳細] セクションで、以下の操作を実行します。
 - a. [デフォルトメッセージタイプ] で、次のいずれかを選択します。
 - [プロモーション] — 重要度が低いメッセージ (広告など)。Amazon SNS は、最も低いコストが発生するようにメッセージ配信を最適化します。
 - [トランザクション] - 多要素認証のワンタイムパスコードなど、お客様のトランザクションをサポートするクリティカルメッセージ。Amazon SNS は、メッセージ配信を最も高い信頼性の実現のために最適化します。

プロモーションおよびトランザクションメッセージの料金表の詳細については、「[グローバル SMS 料金表](#)」を参照してください。

- b. [アカウントの使用料制限] に、各暦月で SMS メッセージに使用する額 (USD) を入力します。

 Important

- デフォルトでは、使用限度は 1.00 USD に設定されます。サービス限度を引き上げるには、[リクエストを送信](#)します。
- コンソールに設定した量がサービス限度を超える場合、Amazon SNS は SMS メッセージの発行を停止します。
- Amazon SNS は分散システムであるため、使用クォータを超えると数分以内に SMS メッセージの送信を停止します。その間に SMS メッセージを送信し続けると、クォータを超えるコストが発生する可能性があります。

6. (オプション) [デフォルトの送信者 ID] にカスタム ID (ビジネスブランドなど) を入力します。これは、受信デバイスに送信者として表示されます。

 Note

送信者 ID のサポートについては、国によって異なります。

7. (オプション) 使用状況レポート用の Amazon S3 バケット名の名前を入力します。

 Note

Amazon S3 バケットポリシーによって Amazon SNS への書き込みアクセスが付与される必要があります。

8. [Save changes] (変更の保存) をクリックします。

設定 (AWS SDKs の設定)

AWS SDKs のいずれかを使用して SMS 設定を設定するには、Amazon SNS API の `SetSMSAttributes` リクエストに対応する SDK のアクションを使用します。Amazon SNS このリクエストによって、月ごとの使用限度およびデフォルトの SMS タイプ (プロモーションまたは

トランザクション)などの、異なる SMS の属性に値を割り当てます。利用可能な SMS 属性の詳細については、『Amazon Simple Notification Service API リファレンス』の「[SetSMSAttributes](#)」を参照してください。

次のサンプルコードは、SetSMSAttributes 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。

Amazon SNS を使用して DefaultSMSType 属性を設定する方法。

```
//! Set the default settings for sending SMS messages.
/*!
    \param smsType: The type of SMS message that you will send by default.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::setSMSType(const Aws::String &smsType,
                             const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SetSMSAttributesRequest request;
    request.AddAttributes("DefaultSMSType", smsType);

    const Aws::SNS::Model::SetSMSAttributesOutcome outcome =
snsClient.SetSMSAttributes(
    request);

    if (outcome.IsSuccess()) {
        std::cout << "SMS Type set successfully " << std::endl;
    }
    else {
        std::cerr << "Error while setting SMS Type: '"
        << 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「[SetSMSAttributes](#)」を参照してください。

CLI

AWS CLI

SMS メッセージ属性を設定するには

次の `set-sms-attributes` の例では、SMS メッセージのデフォルトの送信者 ID を `MyName` に設定します。

```
aws sns set-sms-attributes \
    --attributes DefaultSenderId=MyName
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[SetSMSAttributes](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesResponse;
```

```
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.HashMap;

/**
 * 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
 */
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();
        setSMSAttributes(snsClient, attributes);
        snsClient.close();
    }

    public static void setSMSAttributes(SnsClient snsClient, HashMap<String,
String> attributes) {
        try {
            SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
                .attributes(attributes)
                .build();

            SetSmsAttributesResponse result =
snsClient.setSMSAttributes(request);
            System.out.println("Set default Attributes to " + attributes + ".
Status was "
                + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[SetSMSAttributes](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional - (Default) Noncritical messages, such as marketing
        // messages.
        // Transactional - Critical messages that support customer transactions,
```

```
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
    },
  )),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   }
// }
return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[SetSMSAttributes](#)」を参照してください。

PHP

SDK for PHP

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。

```
$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
```

```
$result = $SnSClient->SetSMSAttributes([
    'attributes' => [
        'DefaultSMSType' => 'Transactional',
    ],
]);
var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[SetSMSAttributes](#)」を参照してください。

国に固有の配信のための SMS メッセージプリファレンスの設定

特定の国の送信先にのみメッセージを送信して、SMS トラフィックを管理および制御できます。これにより、メッセージは承認された国にのみ送信され、不要な SMS 料金を回避できます。次の手順により、Amazon Pinpoint の Protect 設定を使用して、許可またはブロックする国を指定します。

1. <https://console.aws.amazon.com/sms-voice/> で AWS SMS コンソールを開きます。
2. ナビゲーションペインの [概要] の [クイックスタート] セクションで、[保護設定の作成] を選択します。
3. [保護設定の詳細] で、保護設定のわかりやすい名前を入力します (例えば、「オーストラリアのみを許可」など)。
4. [SMS に関する国ルール] で、[地域/国] のチェックボックスをオンにして、サポートされているすべての国へのメッセージの送信をブロックします。
5. メッセージを送信する国のチェックボックスをオフにします。例えば、オーストラリアへのメッセージのみを許可するには、[オーストラリア] のチェックボックスをオフにします。
6. [保護設定の関連付け] セクションの [関連付けタイプ] で、[アカウントのデフォルト] を選択します。これにより、Protect AWS End User Messaging SMS 設定が Amazon SNS、[Amazon Cognito](#)、Amazon Pinpoint [SendMessage](#) API コールを介して送信されるすべてのメッセージに影響するようになります。
7. [保護設定を作成] を選択して、設定を保存します。

次の確認メッセージが表示されます。

```
Success Protect configuration protect-abc0123456789 has been created.
```

8. [Amazon SNS コンソール](#)にサインインします。
9. [メッセージを発行](#)して、例えばインドなど、ブロックされている国のいずれかに送信します。

メッセージが配信されないことを確認します。これは、[CloudWatch](#) を使用して、配信失敗ログで確認できます。ロググループ `sns/region/AccountID/DirectPublishToPhoneNumber/Failure` で、次の例のようなレスポンスを検索します。

```
{
  "notification": {
    "messageId": "bd59a509-XXXX-XXXX-82f8-fbdb8cb68217",
    "timestamp": "YYYY-MM-DD XX:XX:XX.XXXX"
  },
  "delivery": {
    "destination": "+91XXXXXXXXXX",
    "smsType": "Transactional",
    "providerResponse": "Cannot deliver message to the specified destination country",
    "dwellTimeMs": 85
  },
  "status": "FAILURE"
}
```

Amazon SNS 電話番号およびサブスクリプションを管理する

Amazon SNS には、アカウントから SMS メッセージを受信するユーザーを管理するためのオプションが複数用意されています。制限された頻度で、アカウントからの SMS メッセージの受信をオプトアウトした電話番号をオプトインできます。SMS サブスクリプションへのメッセージの送信を停止するために、それらに発行されるサブスクリプションやトピックを削除できます。

SMS メッセージの受信をオプトアウトする

現地の法律および規制により義務付けられている場合 (米国およびカナダなど)、SMS の受信者は、デバイスを利用して以下のいずれかのメッセージに返信することによって、オプトアウトできます。

- ARRET (フランス語)
- CANCEL

- END
- OPT-OUT
- OPTOUT
- QUIT
- REMOVE
- STOP
- TD
- UNSUBSCRIBE

オプトアウトするには、受信者は Amazon SNS がメッセージの配信に使用したのと同じ[送信元番号](#)に返信する必要があります。オプトアウトすると、電話番号をオプトイン AWS アカウント しない限り、受信者は から配信された SMS メッセージを受信しなくなります。

電話番号が Amazon SNS トピックにサブスクライブされると、オプトアウトはサブスクリプションを削除しませんが、電話番号をオプトインしない限り、SMS メッセージは、サブスクリプションへ送信できません。

Amazon SNS コンソールを使用した電話番号とサブスクリプションの管理

Amazon SNS コンソールを使用して、アカウントで SMS メッセージを受信する電話番号を管理できます。

Amazon SNS コンソールをオプトアウトした電話番号をオプトインする

ユーザーのアカウントから SMS メッセージの受信をオプトアウトした電話番号を表示できます。これらの電話番号をオプトインして、メッセージの送信を再開することができます。

30 日に 1 回のみ、電話番号をオプトインできます。

1. [Amazon SNS コンソール](#)にサインインします。
2. コンソールメニューで、[SMS メッセージングをサポートしているリージョン](#)にリージョンセレクトを設定します。
3. ナビゲーションパネルで、[テキストメッセージング (SMS)] を選択します。
4. モバイルテキストメッセージ (SMS) ページのオプトアウト電話番号セクションに、オプトアウトされた電話番号が表示されます。

5. オプトインする電話番号のチェックボックスをオンにして、[オプトイン] を選択します。電話番号はオプトアウトではなくなり、送信した SMS メッセージを受信します。

Amazon SNS コンソール SMS サブスクリプション削除

トピックにパブリッシュするときに、その電話番号への SMS メッセージの送信を停止するように SMS サブスクリプションを削除します。

1. ナビゲーションパネルで、[サブスクリプション] を選択します。
2. 削除するサブスクリプションのチェックボックスをオンにします。それから、[アクション] を選択して、[アプリケーションの削除] を選択します。
3. [削除] ウィンドウで、[削除] を選択します。Amazon SNS はサブスクリプションを削除し、成功のメッセージを表示します。

トピック削除 Amazon SNS コンソール

サブスクライブしたエンドポイントにメッセージを発行しない場合トピックを削除します。

1. ナビゲーションパネルで、[トピック] を選択します。
2. 削除するトピックのチェックボックスをオンにします。[アクション] を選択してから、[トピックの削除] を選択します。
3. [削除] ウィンドウで、[削除] を選択します。Amazon SNS はトピックを削除し、成功のメッセージを表示します。

AWS SDK を使用した電話番号とサブスクリプションの管理

AWS SDKs を使用して Amazon SNS にプログラムでリクエストを行い、アカウントから SMS メッセージを受信できる電話番号を管理できます。

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、「AWS SDK とツールのリファレンスガイド」の「[共有ファイル config と credentials](#)」を参照してください。

AWS SDK を使用してオプトアウトしたすべての電話番号を表示する

すべてのオプトアウトした電話番号を表示するには、Amazon SNS API を使用して、ListPhoneNumbersOptedOut リクエストを送信します。

次のサンプルコードは、`ListPhoneNumbersOptedOut` を使用する方法を説明しています。

CLI

AWS CLI

SMS メッセージのオプトアウトを一覧表示するには

次の `list-phone-numbers-opted-out` の例では、SMS メッセージの受信をオプトアウトした電話番号を一覧表示しています。

```
aws sns list-phone-numbers-opted-out
```

出力:

```
{
  "phoneNumbers": [
    "+15555550100"
  ]
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ListPhoneNumbersOptedOut](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListPhoneNumbersOptedOutRequest;
import
  software.amazon.awssdk.services.sns.model.ListPhoneNumbersOptedOutResponse;
```

```
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class ListOptOut {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listOpts(snsClient);
        snsClient.close();
    }

    public static void listOpts(SnsClient snsClient) {
        try {
            ListPhoneNumbersOptedOutRequest request =
                ListPhoneNumbersOptedOutRequest.builder().build();
            ListPhoneNumbersOptedOutResponse result =
                snsClient.listPhoneNumbersOptedOut(request);
            System.out.println("Status is " +
                result.sdkHttpResponse().statusCode() + "\n\nPhone Numbers: \n\n"
                + result.phoneNumbers());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[ListPhoneNumbersOptedOut](#)」を参照してください。

PHP

SDK for PHP

 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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Returns a list of phone numbers that are opted out of receiving SMS messages
 * from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSClient->listPhoneNumbersOptedOut();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。

- API の詳細については、AWS SDK for PHP API リファレンスの「[ListPhoneNumbersOptedOut](#)」を参照してください。

AWS SDK を使用して電話番号がオプトアウトされているかどうかを確認する

電話番号がオプトアウトしているかどうかを確認するには、Amazon SNS API を使用して、`CheckIfPhoneNumberIsOptedOut` リクエストを送信します。

次のサンプルコードは、`CheckIfPhoneNumberIsOptedOut` を使用する方法を説明しています。

.NET

SDK for .NET

 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。

```
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use the Amazon Simple Notification Service
/// (Amazon SNS) to check whether a phone number has been opted out.
/// </summary>
public class IsPhoneNumOptedOut
{
    public static async Task Main()
    {
        string phoneNumber = "+15551112222";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await CheckIfOptedOutAsync(client, phoneNumber);
    }
}
```

```
/// <summary>
/// Checks to see if the supplied phone number has been opted out.
/// </summary>
/// <param name="client">The initialized Amazon SNS Client object used
/// to check if the phone number has been opted out.</param>
/// <param name="phoneNumber">A string representing the phone number
/// to check.</param>
public static async Task
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string
phoneNumber)
{
    var request = new CheckIfPhoneNumberIsOptedOutRequest
    {
        PhoneNumber = phoneNumber,
    };

    try
    {
        var response = await
client.CheckIfPhoneNumberIsOptedOutAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            string optOutStatus = response.IsOptedOut ? "opted out" :
"not opted out.";
            Console.WriteLine($"The phone number: {phoneNumber} is
{optOutStatus}");
        }
    }
    catch (AuthorizationException ex)
    {
        Console.WriteLine($"{ex.Message}");
    }
}
```

- APIの詳細については、AWS SDK for .NET API リファレンスの「[CheckIfPhoneNumberIsOptedOut](#)」を参照してください。

CLI

AWS CLI

電話番号での SMS メッセージのオプトアウトを確認するには

次の `check-if-phone-number-is-opted-out` 例では、指定された電話番号が現在の AWS アカウントからの SMS メッセージの受信をオプトアウトされているかどうかを確認します。

```
aws sns check-if-phone-number-is-opted-out \  
--phone-number +1555550100
```

出力:

```
{  
  "isOptedOut": false  
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[CheckIfPhoneNumberIsOptedOut](#)」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。

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sns.SnsClient;  
import  
    software.amazon.awssdk.services.sns.model.CheckIfPhoneNumberIsOptedOutRequest;  
import  
    software.amazon.awssdk.services.sns.model.CheckIfPhoneNumberIsOptedOutResponse;  
import software.amazon.awssdk.services.sns.model.SnsException;  
  
/**
```



```
* 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
*/
public class CheckOptOut {
    public static void main(String[] args) {

        final String usage = ""

            Usage:    <phoneNumber>

            Where:
                phoneNumber - The mobile phone number to look up (for example,
+1XXX5550100).

            "";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String phoneNumber = args[0];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        checkPhone(snsClient, phoneNumber);
        snsClient.close();
    }

    public static void checkPhone(SnsClient snsClient, String phoneNumber) {
        try {
            CheckIfPhoneNumberIsOptedOutRequest request =
                CheckIfPhoneNumberIsOptedOutRequest.builder()
                    .phoneNumber(phoneNumber)
                    .build();

            CheckIfPhoneNumberIsOptedOutResponse result =
                snsClient.checkIfPhoneNumberIsOptedOut(request);
        }
    }
}
```

```
        System.out.println(
            result.isOptedOut() + "Phone Number " + phoneNumber + " has
Opted Out of receiving sns messages." +
            "\n\nStatus was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[CheckIfPhoneNumberIsOptedOut](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの [CheckIfPhoneNumberIsOptedOut](#) を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';
```

```
use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Indicates whether the phone number owner has opted out of receiving SMS
 * messages from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$phone = '+1XXX5550100';

try {
    $result = $SnSClient->checkIfPhoneNumberIsOptedOut([
        'phoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの [CheckIfPhoneNumberIsOptedOut](#) を参照してください。

Amazon SNS API を使用してオプトアウトされた電話番号をオプトインする

電話番号をオプトインするには、Amazon SNS API を使用して、OptInPhoneNumber リクエストを送信します。

30 日に 1 回のみ、電話番号をオプトインできます。

AWS SDK を使用した SMS サブスクリプションの削除

Amazon SNS トピックからの SMS サブスクリプションを削除するには、Amazon SNS API を使用して ListSubscriptions リクエストを送信することにより、サブスクリプション ARN を取得し、次に、その ARN を Unsubscribe リクエストに渡します。

次のサンプルコードは、Unsubscribe を使用する方法を説明しています。

.NET

SDK for .NET

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。

サブスクリプション ARN でトピックからサブスクライブを解除します。

```
/// <summary>
/// Unsubscribe from a topic by a subscription ARN.
/// </summary>
/// <param name="subscriptionArn">The ARN of the subscription.</param>
/// <returns>True if successful.</returns>
public async Task<bool> UnsubscribeByArn(string subscriptionArn)
{
    var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
        new UnsubscribeRequest()
        {
            SubscriptionArn = subscriptionArn
        });
    return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[Unsubscribe](#)」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 subscription to an Amazon Simple Notification Service (Amazon SNS)
topic.
/*!
  \param subscriptionARN: The Amazon Resource Name (ARN) for an Amazon SNS topic
subscription.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::SNS::unsubscribe(const Aws::String &subscriptionARN,
                              const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::UnsubscribeRequest request;
    request.SetSubscriptionArn(subscriptionARN);

    const Aws::SNS::Model::UnsubscribeOutcome outcome =
snsClient.Unsubscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Unsubscribed successfully " << std::endl;
    }
    else {
        std::cerr << "Error while unsubscribing " <<
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「[Unsubscribe](#)」を参照してください。

CLI

AWS CLI

トピックからサブスクライブを解除するには

次の unsubscribe の例では、指定したサブスクリプションをトピックから削除します。

```
aws sns unsubscribe \  
    --subscription-arn arn:aws:sns:us-west-2:0123456789012:my-  
topic:8a21d249-4329-4871-acc6-7be709c6ea7f
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[Unsubscribe](#)」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。

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sns.SnsClient;  
import software.amazon.awssdk.services.sns.model.SnsException;  
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;  
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;  
  
/**  
 * 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
*/
public class Unsubscribe {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <subscriptionArn>

            Where:
                subscriptionArn - The ARN of the subscription to delete.
            "";

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String subscriptionArn = args[0];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        unSub(snsClient, subscriptionArn);
        snsClient.close();
    }

    public static void unSub(SnsClient snsClient, String subscriptionArn) {
        try {
            UnsubscribeRequest request = UnsubscribeRequest.builder()
                .subscriptionArn(subscriptionArn)
                .build();

            UnsubscribeResponse result = snsClient.unsubscribe(request);
            System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode()
                + "\n\nSubscription was removed for " +
request.subscriptionArn());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```



```
}  
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[Unsubscribe](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave  
// it blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";  
import { snsClient } from "../libs/snsClient.js";  
  
/**  
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.  
 */  
const unsubscribe = async (  
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-  
xxxx-xxxx-xxxxxxxxxxxxxx",  
) => {  
  const response = await snsClient.send(  
    new UnsubscribeCommand({
```

```
        SubscriptionArn: subscriptionArn,
    })),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   }
// }
return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[Unsubscribe](#)」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 unSub(subscriptionArnVal: String) {
    val request =
        UnsubscribeRequest {
            subscriptionArn = subscriptionArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.unsubscribe(request)
        println("Subscription was removed for ${request.subscriptionArn}")
    }
}
```

```
}  
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[Unsubscribe](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';  
  
use Aws\Exception\AwsException;  
use Aws\Sns\SnsClient;  
  
/**  
 * Deletes a subscription to an Amazon SNS topic.  
 *  
 * This code expects that you have AWS credentials set up per:  
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/  
guide_credentials.html  
 */  
  
$SnSClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);  
  
$subscription = 'arn:aws:sns:us-east-1:111122223333:MySubscription';  
  
try {  
    $result = $SnSClient->unsubscribe([
```

```

        'SubscriptionArn' => $subscription,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}

```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[Unsubscribe](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def delete_subscription(subscription):
        """
        Unsubscribes and deletes a subscription.
        """
        try:
            subscription.delete()

```

```
        logger.info("Deleted subscription %s.", subscription.arn)
    except ClientError:
        logger.exception("Couldn't delete subscription %s.",
subscription.arn)
        raise
```

```
class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def unsubscribe(self, subscription_arn: str) -> bool:
        """
        Unsubscribe from an SNS topic.

        :param subscription_arn: The ARN of the subscription to remove.
        :return: True if successful.
        :raises ClientError: If the unsubscribe operation fails.
        """
        try:
            self.sns_client.unsubscribe(SubscriptionArn=subscription_arn)

            logger.info(f"Unsubscribed: {subscription_arn}")
            return True
```

```
except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')

    if error_code == 'NotFound':
        logger.warning(f"Subscription not found: {subscription_arn}")
        return True # Already unsubscribed
    else:
        logger.error(f"Error unsubscribing: {error_code} - {e}")
        raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[Unsubscribe](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.
    lo_sns->unsubscribe( iv_subscriptionarn = iv_subscription_arn ).
    MESSAGE 'Subscription deleted.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Subscription does not exist.' TYPE 'E'.
CATCH /aws1/cx_snsinvalidparameterex.
    MESSAGE 'Subscription with "PendingConfirmation" status cannot be
    deleted/unsubscribed. Confirm subscription before performing unsubscribe
    operation.' TYPE 'E'.
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「[Unsubscribe](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

_ = try await snsClient.unsubscribe(
    input: UnsubscribeInput(
        subscriptionArn: arn
    )
)

print("Unsubscribed.")
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[Unsubscribe](#)」を参照してください。

AWS SDK を使用してトピックを削除する

トピックとサブスクリプションすべてを削除するには、Amazon SNS API を使用して ListTopics リクエストを送信することでトピック ARN を取得し、次に、その ARN を DeleteTopic リクエストに渡します。

次のサンプルコードは、DeleteTopic を使用する方法を説明しています。

.NET

SDK for .NET

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。

トピック ARN でトピックを削除します。

```
/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「[DeleteTopic](#)」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 an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::deleteTopic(const Aws::String &topicARN,
                              const Aws::Client::ClientConfiguration
                              &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::DeleteTopicRequest request;
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::DeleteTopicOutcome outcome =
        snsClient.DeleteTopic(request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted the Amazon SNS topic " << topicARN <<
            std::endl;
    }
    else {
        std::cerr << "Error deleting topic " << topicARN << ":" <<
            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「[DeleteTopic](#)」を参照してください。

CLI

AWS CLI

SNS トピックを削除するには

次の delete-topic の例では、指定した SNS トピックを削除します。

```
aws sns delete-topic \
```

```
--topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic"
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[DeleteTopic](#)」を参照してください。

Go

SDK for Go V2

 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 (
    "context"
    "encoding/json"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/sns"
    "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type SnsActions struct {
    SnsClient *sns.Client
}

// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
    _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
        TopicArn: aws.String(topicArn)})
    if err != nil {
```

```
    log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
}
return err
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[DeleteTopic](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class DeleteTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:      <topicArn>
```

```
        Where:
            topicArn - The ARN of the topic to delete.
        """;

    if (args.length != 1) {
        System.out.println(usage);
        System.exit(1);
    }

    String topicArn = args[0];
    SnsClient snsClient = SnsClient.builder()
        .region(Region.US_EAST_1)
        .build();

    System.out.println("Deleting a topic with name: " + topicArn);
    deleteSNSTopic(snsClient, topicArn);
    snsClient.close();
}

public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
    try {
        DeleteTopicRequest request = DeleteTopicRequest.builder()
            .topicArn(topicArn)
            .build();

        DeleteTopicResponse result = snsClient.deleteTopic(request);
        System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[DeleteTopic](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
```

```
// }  
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[DeleteTopic](#)」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 deleteSNSTopic(topicArnVal: String) {  
    val request =  
        DeleteTopicRequest {  
            topicArn = topicArnVal  
        }  
  
    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->  
        snsClient.deleteTopic(request)  
        println("$topicArnVal was successfully deleted.")  
    }  
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[DeleteTopic](#)」を参照してください。

PHP

SDK for PHP

 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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Deletes an SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->deleteTopic([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- API の詳細については、AWS SDK for PHP API リファレンスの「[DeleteTopic](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def delete_topic(topic):
        """
        Deletes a topic. All subscriptions to the topic are also deleted.
        """
        try:
            topic.delete()
            logger.info("Deleted topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't delete topic %s.", topic.arn)
            raise
```

```
class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""
```



```
def __init__(self, sns_client: Any) -> None:
    """
    Initialize the SnsWrapper.

    :param sns_client: A Boto3 Amazon SNS client.
    """
    self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def delete_topic(self, topic_arn: str) -> bool:
        """
        Delete an SNS topic.

        :param topic_arn: The ARN of the topic to delete.
        :return: True if successful.
        :raises ClientError: If the topic deletion fails.
        """
        try:
            self.sns_client.delete_topic(TopicArn=topic_arn)

            logger.info(f"Deleted topic: {topic_arn}")
            return True

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')

            if error_code == 'NotFound':
                logger.warning(f"Topic not found: {topic_arn}")
                return True # Already deleted
            else:
                logger.error(f"Error deleting topic: {error_code} - {e}")
                raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[DeleteTopic](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.  
    lo_sns->deletetopic( iv_topicarn = iv_topic_arn ).  
    MESSAGE 'SNS topic deleted.' TYPE 'I'.  
CATCH /aws1/cx_snsnotfoundexception.  
    MESSAGE 'Topic does not exist.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[DeleteTopic](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS
```

```
let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

_ = try await snsClient.deleteTopic(
    input: DeleteTopicInput(topicArn: arn)
)
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[DeleteTopic](#)」を参照してください。

Amazon SNS SMS アクティビティモニタリング

SMS をモニタリングすることによって、送信先の電話番号、正常または失敗した配信、失敗の原因、コスト、およびその他の情報を追跡できます。Amazon SNS は、コンソールで統計をまとめ、Amazon CloudWatch に情報を送信し、指定した Amazon S3 バケットに SMS の毎日の使用状況レポートを送信します。

Amazon SNS SMS 配信統計を表示する

最新の SMS 配信に関する統計を表示するために Amazon SNS コンソールを使用できます。

1. [Amazon SNS コンソール](#)にサインインします。
2. コンソールメニューで、[SMS メッセージングをサポートしているリージョン](#)にリージョンセレクトを設定します。
3. ナビゲーションパネルで、[テキストメッセージング (SMS)] を選択します。
4. [テキストメッセージ (SMS)] ページの、[Account stats] セクションで、トランザクションおよびプロモーション SMS メッセージ配信のグラフを表示します。各グラフは、前述の 15 日間の次のデータを表示します。
 - 配信成功率 (正常な配信の割合)
 - 送信 (配信試行数)
 - 失敗 (配信の失敗数)

このページで、[使用状況]ボタンを選択して、毎日の使用状況レポートを保存する Amazon S3 バケットに移動することもできます。詳細については、「[Amazon SNS の毎日の SMS 使用状況レポートにサブスクライブする](#)」を参照してください。

Amazon CloudWatch メトリクスとログを使用した Amazon SNS SMS 配信モニタリング

Amazon CloudWatch および Amazon CloudWatch Logs を使用して、SMS のメッセージ配信をモニタリングできます。

Amazon CloudWatch メトリクスを表示する

Amazon SNS は、自動的に SMS メッセージ配信に関するメトリクスを収集し、これらを Amazon CloudWatch にプッシュします。CloudWatch を使用してこれらのメトリクスをモニタリングし、そのメトリクスがしきい値を超えたときに警告するようにアラームを作成できます。例えば、CloudWatch メトリクスをモニタリングして、SMS 配信率と今月の始めから今日までの SMS 料金を知ることができます。

CloudWatch メトリクスのモニタリング、アラームの設定、利用可能なメトリクスの種類の詳細については、「[CloudWatch を使用した Amazon SNS のモニタリング](#)」を参照してください。

CloudWatch Logs を表示する

Amazon SNS による Amazon CloudWatch Logs への書き込みを有効化することにより、正常および失敗した SMS メッセージ配信の情報を収集できます。送信した各 SMS メッセージに対して、Amazon SNS は、メッセージの価格、成功または失敗のステータス、失敗の理由 (メッセージが失敗した場合)、メッセージのドウェル時間、その他の情報を含むログを書き込みます。

SMS メッセージに対して CloudWatch Logs を有効にして表示するには

1. [Amazon SNS コンソール](#)にサインインします。
2. コンソールメニューで、[SMS メッセージングをサポートしているリージョン](#)にリージョンセレクトを設定します。
3. ナビゲーションパネルで、[テキストメッセージング (SMS)] を選択します。
4. [モバイルテキストメッセージング (SMS)] ページの [テキストメッセージプリファレンス] セクションで、[編集] を選択します。
5. 次のページで、[Delivery status logging] セクションに追加します。
6. [成功サンプルの割合] で、Amazon SNS が CloudWatch Logs でログを書き込む正常な SMS 配信の割合を指定します。例:
 - 失敗した配信のみをログを書き込むには、この値を 0 に設定します。
 - 正常な配信の 10% に対してログを書き込むには、10 に設定します。

割合を指定しないなら、Amazon SNS は、すべての正常配信に対してログを書き込みます。

7. 必要なアクセス権限を提供するには、次のいずれかを行います。

- 新しいサービスロールを作成するには、[新しいサービスロールの作成]を選択し、[新しいロールの作成]を選択します。次のページで、[許可]を使用して、Amazon SNS にアカウントのリソースへの書き込みアクセス許可を付与します。
- 既存のサービスロールを使用するには、[既存のサービスロールを使用する]を選択し、ARN 名を [成功した配信および失敗した配信用の IAM ロール] ボックスにペーストします。

指定するサービスロールは、アカウントのリソースへの書き込みアクセスを許可する必要があります。IAM ロールの作成の詳細については、IAM [ユーザーガイドの「AWS サービスのロールの作成」](#)を参照してください。

8. [Save changes] (変更の保存) をクリックします。

9. [モバイルテキストメッセージング (SMS)] ページに戻り、[配信ステータスログ] セクションに移動し、使用可能なログを表示します。

Note

送信先の電話番号のキャリアによっては、Amazon SNS コンソールに配信ログが表示されるまでに最大 72 時間かかる場合があります。

正常な SMS 配信のログの例

正常な SMS 配信の配信ステータスのログは次の例のようになります。

```
{
  "notification": {
    "messageId": "34d9b400-c6dd-5444-820d-fbeb0f1f54cf",
    "timestamp": "2016-06-28 00:40:34.558"
  },
  "delivery": {
    "phoneCarrier": "My Phone Carrier",
    "mnc": 270,
    "numberOfMessageParts": 1,
    "destination": "+1XXX5550100",
    "priceInUSD": 0.00645,
    "smsType": "Transactional",
    "mcc": 310,
    "providerResponse": "Message has been accepted by phone carrier",
    "dwellTimeMs": 599,
```

```
    "dwellTimeMsUntilDeviceAck": 1344
  },
  "status": "SUCCESS"
}
```

失敗した SMS 配信のログの例

失敗した SMS 配信の配信ステータスのログは次の例のようになります。

```
{
  "notification": {
    "messageId": "1077257a-92f3-5ca3-bc97-6a915b310625",
    "timestamp": "2016-06-28 00:40:34.559"
  },
  "delivery": {
    "mnc": 0,
    "numberOfMessageParts": 1,
    "destination": "+1XXX5550100",
    "priceInUSD": 0.00645,
    "smsType": "Transactional",
    "mcc": 0,
    "providerResponse": "Unknown error attempting to reach phone",
    "dwellTimeMs": 1420,
    "dwellTimeMsUntilDeviceAck": 1692
  },
  "status": "FAILURE"
}
```

SMS 配信の失敗の原因

失敗の理由は、`providerResponse` 属性で提供されます。SMS メッセージは以下の理由で送信に失敗することがあります。

- 電話業者がスパムとしてブロック
- 送信先がブロックリストにある
- 無効な電話番号
- メッセージ本文が無効
- 電話業者がこのメッセージをブロック
- 電話業者が現在、到達不可能/使用不可能
- 電話が SMS をブロック

- 電話がブロックリストにある
- 電話が現在、到達不可能/使用不可能
- 電話番号はオプトアウトしている
- この配信が上限価格を超えている
- 電話に達する際の原因不明なエラー

Amazon SNS の毎日の SMS 使用状況レポートにサブスクライブする

Amazon SNS から毎日の使用状況レポートにサブスクライブすることによって、SMS 配信をモニタリングできます。1 つ以上の SMS メッセージを送信する日ごとに、Amazon SNS は、指定した Amazon S3 バケットに CSV ファイルとして使用状況レポートを配信します。SMS 使用状況レポートが Amazon S3 バケットで利用可能になるまでに、24 時間かかります。

毎日の使用状況レポートの情報

使用状況レポートには、アカウントから正常に配信した各 SMS メッセージに関する以下の情報が含まれています。

このレポートには、オプトアウトした受取人に送信されるメッセージは含まれない点に注意してください。

- メッセージの発行時間 (UTC)
- メッセージ ID
- 送信先電話番号
- メッセージタイプ
- 配信ステータス
- メッセージの価格 (USD)
- パート番号 (1 通のメッセージとして長すぎる場合、メッセージは複数のパートに分割されます)
- パートの総数

Note

Amazon SNS がパート番号を受け取らなかった場合は、その値をゼロに設定します。

毎日の使用状況レポートにサブスクライブする

毎日の使用状況レポートにサブスクライブするには、適切なアクセス権限を持つ Amazon S3 バケットを作成する必要があります。

毎日の使用状況レポート用の Amazon S3 バケットを作成するには

1. SMS メッセージを送信 AWS アカウント する から、[Amazon S3 コンソール](#)にサインインします。
2. [バケットの作成] を選択します。
3. [バケット名] には、アカウントと組織で一意的な名前を入力することをお勧めします。例えば、パターン `<my-bucket-prefix>-<account_id>-<org-id>` を使用します。

バケット名に関する規則と制限については、『Amazon Simple Storage Service ユーザーガイド』の「[バケットの命名規則](#)」を参照してください。

4. [作成] を選択します。
5. [すべてのバケット] テーブルでバケットを選択します。
6. [Permissions] セクションで、[バケットポリシー] を選択します。
7. [バケットポリシーエディター] ウィンドウで、Amazon SNS サービスプリンシパルをバケットに書き込むことを許可するポリシーを指定します。例については、[バケットポリシーの例](#)を参照してください。

サンプルポリシーを使用する場合は、*my-s3-bucket* をステップ 3 で選択したバケット名に置き換えてください。

8. [保存] を選択します。

毎日の使用状況レポートにサブスクライブするには

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[テキストメッセージング (SMS)] を選択します。
3. [テキストメッセージング (SMS)] ページの [テキストメッセージングの設定] セクションで、[編集] を選択します。

Text messaging preferences		Edit
Default message type	IAM role for logging delivery status in CloudWatch Logs	
-	-	
Account spend limit	Amazon S3 bucket name for usage reports	

4. [テキストメッセージングの設定の編集] ページの [詳細] セクションで、[使用状況レポートに使用する Amazon S3 バケット名] を指定します。

Amazon S3 bucket name for usage reports - *optional*
The Amazon S3 bucket to receive daily SMS usage reports. The bucket policy must grant write access to Amazon SNS.

Enter the name of the bucket

The name of a bucket must be 3 to 63 characters long, not containing uppercase letters, spaces or underscores ().

5. [Save changes] (変更の保存) をクリックします。

バケットポリシーの例

以下のポリシーは、Amazon SNS サービスプリンシパルに `s3:PutObject`、`s3:GetBucketLocation` および `s3:ListBucket` アクションの実行を許可します。

AWS 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用のツールを提供します。Amazon S3 バケットポリシーステートメントのプリンシパルが [混乱した代理攻撃の問題](#)にある場合。バケットが毎日の使用状況レポートを受信できるリージョンとアカウントを制限するには、以下の例のように `aws:SourceArn` を使用します。これらのレポートを生成できるリージョンを制限したくない場合は、`aws:SourceAccount` を使用して、レポートを生成しているアカウントに基づいて制限します。リソースの ARN が不明の場合は、`aws:SourceAccount` を使用してください。

Amazon S3 バケットを作成して、Amazon SNS から毎日の SMS 使用状況レポートを受信する際に、混乱した代理保護を含む次の例を使用します。

JSON

```
{
  "Version": "2012-10-17",
```

```
"Statement": [  
  {  
    "Sid": "AllowPutObject",  
    "Effect": "Allow",  
    "Principal": {  
      "Service": "sns.amazonaws.com"  
    },  
    "Action": "s3:PutObject",  
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*",  
    "Condition": {  
      "StringEquals": {  
        "aws:SourceAccount": "account_id"  
      },  
      "ArnLike": {  
        "aws:SourceArn": "arn:aws:sns:us-west-1:123456789012:*"  
      }  
    }  
  },  
  {  
    "Sid": "AllowGetBucketLocation",  
    "Effect": "Allow",  
    "Principal": {  
      "Service": "sns.amazonaws.com"  
    },  
    "Action": "s3:GetBucketLocation",  
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",  
    "Condition": {  
      "StringEquals": {  
        "aws:SourceAccount": "account_id"  
      },  
      "ArnLike": {  
        "aws:SourceArn": "arn:aws:sns:us-west-1:123456789012:*"  
      }  
    }  
  },  
  {  
    "Sid": "AllowListBucket",  
    "Effect": "Allow",  
    "Principal": {  
      "Service": "sns.amazonaws.com"  
    },  
    "Action": "s3:ListBucket",  
    "Resource": "arn:aws:s3:::amzn-s3-demo-bucket",  
    "Condition": {
```

```

        "StringEquals": {
            "aws:SourceAccount": "account_id"
        },
        "ArnLike": {
            "aws:SourceArn": "arn:aws:sns:us-west-1:123456789012:*"
        }
    }
}
]
}

```

Note

Amazon S3 ポリシーの Condition エlement で指定される AWS アカウント に所有される Amazon S3 バケットに使用状況レポートを発行できます。別の AWS アカウント が所有する Amazon S3 バケットに使用状況レポートを発行するには、[「別の から Amazon S3 オブジェクトをコピーするにはどうすればよいですか AWS アカウント?」](#)を参照してください。

毎日の使用状況レポートの例

毎日の使用状況レポートにサブスクライブすると、Amazon SNS は毎日、以下の場所に使用状況データを使用して CSV ファイルを配置します。

```
<my-s3-bucket>/SMSUsageReports/<region>/YYYY/MM/DD/00x.csv.gz
```

各ファイルには、最大 50,000 のレコードを含めることができます。1 日のレコードがこの限度を超えると、Amazon SNS は複数のファイルを追加します。レポートの例を以下に示します。

```

PublishTimeUTC,MessageId,DestinationPhoneNumber,MessageType,DeliveryStatus,PriceInUSD,PartNumber
2016-05-10T03:00:29.476Z,96a298ac-1458-4825-
a7eb-7330e0720b72,1XXX5550100,Promotional,Message has been accepted by phone
carrier,0.90084,0,1
2016-05-10T03:00:29.561Z,1e29d394-
d7f4-4dc9-996e-26412032c344,1XXX5550100,Promotional,Message has been accepted by phone
carrier,0.34322,0,1
2016-05-10T03:00:30.769Z,98ba941c-afc7-4c51-
ba2c-56c6570a6c08,1XXX5550100,Transactional,Message has been accepted by phone
carrier,0.27815,0,1

```

Amazon SNS SMS メッセージングのサポートをリクエストする

Important

Amazon SNS SMS デベロッパーガイドが更新されました。Amazon SNS は、SMS メッセージの配信のために [AWS End User Messaging SMS](#) と統合されています。このガイドには、Amazon SNS SMS メッセージを作成、設定、管理する方法に関する最新情報が含まれています。

Amazon SNS での特定の SMS オプションは、サポートに問い合わせるまで AWS アカウントでご利用いただけます。[\[AWS サポート センター\]](#) でケースを開き、以下のいずれかをリクエストします。

- 毎月の SMS 利用額しきい値の増加

デフォルトでは、毎月の利用額しきい値は 1.00 USD です。利用額しきい値により、Amazon SNS で送信できるメッセージのボリュームが決まります。SMS ユースケースで予想される毎月のメッセージ量に合う利用額しきい値をリクエストします。

- [\[SMS サンドボックス\]](#) からの削除すると、制限なく SMS メッセージを送信できます。詳細については、「[Amazon SNS SMS サンドボックス外への移動](#)」を参照してください。
- 専有の [発信元番号](#)
- 専用 [送信者 ID](#)。送信者 ID は、受信者のデバイスで送信者として表示されるカスタム ID です。例えば、メッセージソースを識別しやすいように、ビジネスブランドを使用できます。送信者 ID のサポートについては、国またはリージョンによって異なります。詳細については、「AWS End User Messaging SMS ユーザーガイド」の「[SMS メッセージングがサポートされている国とリージョン](#) [AWS End User Messaging SMS](#)」を参照してください。

Amazon SNS の毎月の SMS 使用クォータの引き上げをリクエストする

Amazon SNS では、アカウントを使用して SMS を送信することで発生する月あたりの最大コストを管理するのに役立つ使用クォータを提供しています。使用クォータは、悪意のある攻撃を受けた場合にリスクを抑え、アップストリームアプリケーションが想定外に多くのメッセージを送信することを防ぎます。Amazon SNS が SMS メッセージの送信により、当月の使用クォータを超えるコストが発生すると判断した場合、SMS メッセージの発行を停止するよう設定できます。

運用に影響が及ばないように、使用クォータが本番用ワークロードをカバーできるよう十分高いクォータをリクエストすることをお勧めします。詳細については、「[ステップ 1: Amazon SNS SMS](#)」

[ケースを開く](#)」を参照してください。クォータを受け取った後、「[ステップ 2: SMS 設定を更新する](#)」の説明に従って、フルクォータまたはより小さい値を適用して、リスクを管理できます。より小さい値を適用することで、毎月のコストを制御できます。そのためには、必要に応じてスケールアップするオプションを使用します。

 Important

Amazon SNS は分散システムであるため、使用クォータを超えると数分以内に SMS メッセージの送信を停止します。この間に SMS メッセージを送信し続けると、クォータを超えるコストが発生する可能性があります。

すべての新しいアカウントに対して 1 か月あたり 1.00 USD で使用クォータを設定します。このクォータは、Amazon SNS のメッセージ送信機能をテストできるようにすることを目的としています。アカウントの SMS 使用クォータの引き上げをリクエストするには、AWS サポートセンターでクォータ引き上げケースを開きます。

トピック

- [ステップ 1: Amazon SNS SMS ケースを開く](#)
- [ステップ 2: Amazon SNS コンソールで SMS 設定を更新する](#)

ステップ 1: Amazon SNS SMS ケースを開く

AWS サポートセンターでクォータ引き上げケースを開くことで、毎月の支出クォータの引き上げをリクエストできます。

 Note

リクエストフォームの一部のフィールドは「オプション」としてマークされています。ただし、サポートでは、リクエストを処理するために次のステップで説明されているすべての情報が必要です。必要な情報をすべて入力しないと、リクエストの処理に遅延が発生する場合があります。

1. <https://console.aws.amazon.com/> AWS マネジメントコンソールでサインインします。
2. [サポート] メニューで [サポートセンター] を選択します。
3. [お客様のサポートケース] ペインで、[ケースを作成] を選択します。

4. [サービス制限の引き上げをご希望ですか?] リンクを選択して、以下を完了します。

- [制限タイプ] で、[SNS テキストメッセージング] を選択します。
- (オプション) [Provide a link to the site or app which will be sending SMS messages (SMS メッセージを送信するサイトまたはアプリケーションへのリンクを指定する)] で、SMS メッセージを送信する Web サイト、アプリケーション、またはサービスに関する情報を入力します。
- (オプション) [What type of messages do you plan to send (送信するメッセージのタイプ)] で、ロングコードを使用して送信する予定のメッセージのタイプを選択します。
- [ワンタイムパスワード] - ウェブサイトまたはアプリケーションを認証するために顧客が使用するパスワードを提供するメッセージ。
- [プロモーション] - 特価販売やお知らせなど、ビジネスやサービスを宣伝する非クリティカルなメッセージ。
- [トランザクション] - 注文確認やアカウントアラートなど、顧客のトランザクションをサポートする重要な情報メッセージ。トランザクションメッセージにプロモーションコンテンツまたはマーケティングコンテンツを含めることはできません。
- (オプション) メッセージを送信する AWS リージョンで、メッセージを送信するリージョンを選択します。
- (オプション) [Which countries do you plan to send messages to] で、ショートコードを購入する国または地域を入力します。
- (オプション) [How do your customers opt to receive messages from you] で、オプトインプロセスの詳細を入力します。
- (オプション) [How do your customers opt to receive messages from you] で、オプトインプロセスの詳細を入力します。[Please provide the message template that you plan to use to send messages to your customers] の項目に使用するテンプレートを入力します。

5. [Requests] で、以下のセクションに入力します。

- [Region] で、メッセージの送信元となるリージョンを選択します。

Note

[Requests] セクションには、リージョンが必要です。また、[Case details] フィールドに入力した場合でも、ここで入力する必要があります。

- [リソースタイプ] で、[一般的な制限] を選択します。

- [制限] で、[Account Spend Threshold Increase] を選択します。
6. [New limit value] に、毎月の SMS メッセージに使用する上限額を USD で入力します。
 7. [Case description] の [Use case description] で、次の詳細を入力します。
 - SMS メッセージを送信する会社またはサービスのウェブサイトやアプリ。
 - ウェブサイトやアプリによって提供されるサービスと、SMS メッセージがそのサービスにどのように貢献するか。
 - ユーザーがウェブサイト、アプリ、他の場所で任意の SMS メッセージ受信にサインアップする方法。

リクエストした使用クォータ ([New quota value] で指定した値) が 10,000 USD を超えた場合、メッセージを送信する国ごとに以下の追加の詳細を指定します。

- 送信者 ID とショートコードのどちらを使用しているか。送信者 ID を使用している場合は、次の情報を指定します。
 - 送信者 ID。
 - 送信者 ID がその国の携帯電話キャリアに登録されているかどうか。
 - メッセージングの予想される 1 秒あたりの最大トランザクション数 (TPS)。
 - 平均メッセージサイズ。
 - 国に送信するメッセージのテンプレート。
 - (オプション) 文字エンコードのニーズ (ある場合)。
- 8. (オプション) さらにリクエストを送信したい場合は、[Add another request] を選択します。複数のリクエストを含める場合は、それぞれ必要な情報を指定します。必要な情報については、「[Amazon SNS SMS メッセージングのサポートをリクエストする](#)」の他のセクションを参照してください。
- 9. [連絡先オプション] の [連絡する際の希望言語] で、このケースに関する連絡を受け取る言語を選択します。
- 10. 完了したら、送信 を選択します。

サポート チームは 24 時間以内にリクエストに最初の応答を提供します。

迷惑なコンテンツや悪意のあるコンテンツを送信するためにシステムが悪用されないように、私たちは各リクエストを慎重に検討しています。可能であれば、24 時間以内にリクエストを承認します。ただし、追加情報が必要な場合は、お客様のリクエストの解決に時間がかかる場合があります。

お客様のユースケースが私たちのポリシーに一致しない場合は、リクエストを承認できない場合があります。

ステップ 2: Amazon SNS コンソールで SMS 設定を更新する

毎月の使用クォータの引き上げが通知された後、Amazon SNS コンソールのアカウントで使用クォータを調整する必要があります。

Important

次のステップを完了する必要があります。完了しない場合、SMS の利用限度額は増えません。

コンソールで使用クォータを調整するには

1. [Amazon SNS コンソール](#)にサインインします。
2. 左側のナビゲーションメニューを開き、[モバイル] を展開し、[テキストメッセージング (SMS)] を選択します。
3. [モバイルテキストメッセージング (SMS)] ページの [テキストメッセージプリファレンス] セクションで、[編集] を選択します。
4. [テキストメッセージングの優先設定の編集] ページの [詳細] で、[アカウントの使用制限] に新しい SMS 使用限度額を入力します。

Note

デフォルトの使用制限より高い値を入力すると、警告が表示される場合があります。このテキストは無視して構いません。

5. [Save changes] (変更の保存) をクリックします。

Note

「Invalid Parameter」というエラーが表示された場合は、AWS サポートの連絡先を確認し、新しい SMS 使用制限が正しく入力されていることを確認します。それでも問題が解決しない場合は、AWS サポートセンターでケースを開きます。

サポート センターでケースを作成するときは、送信するリクエストのタイプに必要なすべての情報を含めてください。含めない場合、続行する前に サポート がお客様に連絡してこの情報を取得します。詳細なケースを送信することにより、ケースが遅れなく処理されます。特定のタイプの SMS リクエストに必要な詳細については、以下のトピックを参照してください。

送信者 ID の詳細については、「AWS End User Messaging SMS ユーザーガイド」の以下のドキュメントを参照してください。

AWS End User Messaging SMS トピック	説明
使用クォータの引き上げのリクエスト	使用クォータにより、毎月 AWS End User Messaging SMS を介した SMS メッセージの送信に使用できる金額が設定されます。
サポートセンターで送信者 ID のケースを開く	送信者 ID が必要な国の受信者にメッセージを送信する予定の場合は、サポート センターで新しいケースを作成して送信者 ID をリクエストできます。

Amazon SNS SMS メッセージングのベストプラクティス

Important

Amazon SNS SMS デベロッパーガイドが更新されました。Amazon SNS は、SMS メッセージの配信のために [AWS End User Messaging SMS](#) と統合されています。このガイドには、Amazon SNS SMS メッセージを作成、設定、管理する方法に関する最新情報が含まれています。

携帯電話ユーザーは、未承諾 SMS メッセージに対する許容度が極めて低い傾向があります。未承諾 SMS キャンペーンの反応率はほぼ常に低くなるため、投資効果は低くなります。

さらに、携帯電話キャリアは大量 SMS の送信者を継続的に監査しています。未承諾メッセージを送信していると判断した番号からのメッセージは、調整されたりブロックされたりします。

未承諾コンテンツの送信は、[AWS の適正利用規約](#)にも違反します。Amazon SNS チームは、SMS キャンペーンを定期的に監査しており、未承諾メッセージを送信していると思われる場合はメッセージの送信能力を調整したりブロックしたりすることがあります。

最後に、多くの国、リージョン、管轄区域では、未承諾 SMS メッセージの送信に多額の罰金を科しています。例えば、米国の Telephone Consumer Protection Act (TCPA) には、消費者が未承諾メッセージを 1 通受け取るたびに 500 ~ 1,500 USD の損害賠償を受け取ることができる (送信者が支払う) と規定されています。

このセクションでは、顧客エンゲージメントを高め、罰金を回避するために役立つ可能性があるいくつかのベストプラクティスについて説明します。ただし、このセクションには法的なアドバイスは含まれていないことに注意してください。法的なアドバイスを受けるには、弁護士に相談してください。

法律、規制、および通信事業者の要件の遵守

あなたの顧客が所在する地域の法および規制に違反すると、高額な罰金と罰が課されます。このためにも、事業を展開する各国や各地域の SMS メッセージングに関する法律を理解することは非常に重要です。

次のリストには、世界中の主要なマーケットにおける SMS コミュニケーションに適用される重要な法律のリンクが含まれています。

- 米国: 1991 年の電話利用者保護法 (TCPA) は、特定の種類の SMS メッセージに適用されます。詳細については、[規則および規制](#)を連邦通信委員会のウェブサイトで参照してください。
- 英国: 2003 年のプライバシーおよび電子通信に関する規制 (EC 指令) (PECR) は、特定の種類の SMS メッセージに適用されます。詳細については、英国情報コミッショナーオフィスのウェブサイトで「[What are PECR?](#)」を参照してください。
- EU: 2002 年のプライバシーおよび電子通信に関する規則指令 (ePrivacy 指令と呼ばれることもある) は、特定の種類の SMS メッセージに適用されます。詳細については、[法律全文](#)を Europa.eu ウェブサイトで参照してください。
- カナダ: インターネットおよび無線スパム対策法 (カナダのアンチスパム法あるいは CASL) は、特定の種類の SMS メッセージに適用されます。詳細については、[法律全文](#)をカナダ議会のウェブサイトで参照してください。
- 日本: 特定電子メールの送信の適正化等に関する法律は、特定の種類の SMS メッセージに適用される場合があります。詳細については、[日本の迷惑メール対策](#)を総務省のウェブサイトで参照してください。

送信者として、会社または組織がこれらの国のいずれかに拠点を置いていない場合でも、これらの法律が適用される場合があります。このリストの一部の法規制は迷惑 E メールあるいは電話を対象に元々制定されたものもありますが、SMS メッセージにも同様に適用するとの見解や拡張解釈が行わ

れています。その他の国々や地域では、SMS メッセージの送信に関する独自の法規制を設けています。顧客が所在する各国や各地域の弁護士に相談して、法的なアドバイスを受けてください。

多くの国では、現地の通信事業者がネットワーク上を流れるトラフィックの種類を決定する最終権限を保有しています。つまり、通信事業者は、現地の法律の最低要件より厳しい制限を SMS コンテンツに課す可能性があります。

許可を取得する

送信する予定のメッセージについて特定のタイプの受信を明示的に要求していない受信者には、決してメッセージを送信しないでください。同じ会社内の組織間であっても、オプトインリストを共有しないでください。

受信者がオンラインフォームを使用してメッセージの受信にサインアップできる場合、本人が気づかずに自動スクリプトがサブスクライブすることを防止するシステムを追加してください。また、1 回のセッションでユーザーが電話番号を送信できる回数を制限する必要があります。

SMS オプトインリクエストを受け取ったら、メッセージの受信を希望することを確認するメッセージを受信者に送信します。受信者がサブスクリプションを確認するまで、追加のメッセージを受信者に送信しないでください。サブスクリプション確認メッセージの例は次のとおりです。

```
Text YES to join ExampleCorp alerts. 2 msgs/month. Msg & data rates may apply. Reply HELP for help, STOP to cancel.
```

日付、時刻、各オプトインリクエストおよび確認のソースが含まれる記録を保持してください。これは、通信事業者または規制機関から依頼された場合に役立つことがあり、顧客リストの定期的な監査の実施にも役立つことがあります。

オプトインワークフロー

場合によっては (米国の通話料無料やショートコード登録など)、モバイル通信事業者からオプトインワークフロー全体のモックアップまたはスクリーンショットの提供が求められます。モックアップまたはスクリーンショットは、受信者が完了するオプトインワークフローによく似ている必要があります。

モックアップまたはスクリーンショットには、最高レベルのコンプライアンスを維持するために必要な以下の開示事項をすべて含める必要があります。

必要な開示

- プログラムを通じて送信するメッセージングユースケースの説明。

- 「メッセージ料金とデータ料金が適用される場合があります」というフレーズ。
- 受信者がメッセージを受け取る頻度の提示。例えば、定期的なメッセージングプログラムでは、「週に 1 通のメッセージ」と表示される場合があります。ワンタイムパスワードまたは多要素認証のユースケースでは、「メッセージの頻度は異なる」または「ログイン試行ごとに 1 通のメッセージ」と表示される場合があります。
- 利用規約とプライバシーポリシー文書へのリンク。

コンプライアンス違反オプトインの一般的な拒否理由

- 提供された会社名がモックアップまたはスクリーンショットに記載されているものと一致しない場合。明確ではない関係については、オプトインワークフローの説明で説明する必要があります。
- メッセージが受信者に送信されるように見えても、送信前に明示的に同意が得られない場合。すべてのメッセージには、明示的な同意が必要です。
- サービスにサインアップするためにテキストメッセージの受信が必要と思われる場合。ワークフローが、電子メールや音声通話などの別の形式でオプトインメッセージを受信する代替手段を提供していない場合、これは準拠しません。
- オプトイン言語がすべて利用規約に記載されている場合。開示内容は、リンク先のポリシー文書の中に入れるのではなく、オプトイン時に必ず受信者に提示する必要があります。
- 顧客があるタイプのメッセージを受信することに同意し、他のタイプのテキストメッセージを送信する場合。例えば、顧客はワンタイムパスワードの受信には同意しましたが、投票メッセージやアンケートメッセージも送信されます。
- 必要な開示 (上記) が受信者に提示されない場合。

次の例は、多要素認証のユースケースに関する携帯端末通信業者の要件に準拠しています。

The first screen shows a form for account creation with fields for First name, Last name, and Email address, each marked with a red asterisk. A 'Next >' button is at the bottom.

The second screen asks if the user wants to enable Multi-Factor Authentication (MFA). It has two radio buttons: 'Enable MFA' (selected) and 'Disable MFA (less secure)'. A 'Next >' button is at the bottom.

The third screen asks how the user wants to receive MFA messages. It has three radio buttons: 'Email' (selected), 'Phone call', and 'Text message'. Below the 'Text message' option, there is a note about message rates and a 'Next >' button. A blue bracket on the right side of the screen indicates that the 'Mobile number' field and the verification instructions only appear when 'Text message' is selected.

1. User provides basic account information.

2. User decides whether to enable MFA.

3. If MFA enabled, user chooses how to receive MFA token.

The fourth screen shows an incoming text message from 67876: 'Your ExampleCorp Multi-factor Authentication code is 918273. Text HELP for more info or STOP to opt out.' A 'Send' button is at the bottom.

The fifth screen shows the user entering the MFA token. It displays the text 'We sent a text message to you at (425) 555-0142. Enter the six digit code in that message to confirm your phone number.' Below this is a 'Resend code' link and a numeric keypad with digits 1-9, 0, and a backspace button.

4. If user chooses to receive MFA token by text, send a token.

5. User enters MFA token to verify phone number.

多要素認証のユースケースのモックアップ

完成したテキストと画像が含まれ、オプトインフロー全体が注釈付きで表示されます。オプトインフローでは、顧客はテキストメッセージの受信に同意するために、明確な意図的行動をとり、必要な開示事項をすべて含める必要があります。

その他のオプトインワークフロータイプ

携帯端末通信業者は、上記の規定に準拠している場合、口頭または書面によるオプトインなど、アプリケーションやウェブサイト以外のオプトインワークフローも受け入れます。コンプライアンスに準拠したオプトインワークフローと、口頭または書面によるスクリプトは、特定のメッセージタイプを受信することについて、受信者から明示的な同意を得ます。例としては、サービスデータベースに記録する前にサポートエージェントが同意を得るために使用する口頭スクリプトや、販促用チラシに記載されている電話番号などがあります。これらのオプトインワークフロータイプのモックアップには、オプトインスクリプト、マーケティング資料、または数字が収集されているデータベースのスクリンショットを提供できます。オプトインが明確ではない場合やユースケースが一定量を超える場合、携帯端末通信業者はこれらのユースケースに関して追加の質問をすることがあります。

古いリストには送信しないでください

人は電話番号を頻繁に変更します。2 年前に連絡先として同意を得た電話番号は、今では他の誰かが使用しているかもしれません。新しいメッセージングプログラムに古い電話番号リストを使用しないでください。使用した場合、電話番号がすでに使用されていないため、メッセージが送信されない場合や、そもそも同意したことを忘れてオプトアウトする人がいる可能性があります。

顧客リストを監査する

定期的な SMS キャンペーンを送信している場合、顧客リストを定期的に監査してください。顧客リストを監査すると、メッセージを受信したい顧客だけがメッセージを確実に受信するようになります。

リストを監査するときは、オプトインしている各顧客に、サブスクライブしていることを再確認するメッセージを送信し、サブスクライブ解除に関する情報を提供します。再確認メッセージの例は次のとおりです。

```
You're subscribed to ExampleCorp alerts. Msg & data rates may apply. Reply  
HELP for help, STOP to unsubscribe.
```

レコードを保持する

SMS メッセージを送信するよう各顧客よりリクエストされた日時と、各顧客に送信したメッセージを示すレコードを保持します。世界の多くの国や地域では、SMS の送信者がこれらのレコードを簡単に取り出せるように維持する必要があります。また、この情報は、携帯端末通信事業者より随時リクエストされる場合があります。提供する情報は、国または地域によって異なります。レコード維持の要件の詳細は、顧客が所在する国または地域ごとの商用 SMS メッセージングの規制を確認します。

場合によっては、キャリアや規制機関より、顧客がメッセージの受け取りを選択した事実の提供を求められることがあります。このような状況では、 から通信事業者または機関が必要とする情報のリストが サポート 届きます。必要な情報を提供しない場合は、以降の SMS メッセージを送信する機能が一時停止される場合があります。

メッセージは明確、正直、簡潔にしてください

SMS はユニークなメディアです。メッセージあたり 160 文字という制限があるため、メッセージは簡潔にする必要があります。電子メールなど、他の通信チャネルで使用する手法が SMS チャネルでは適用できない場合があります、SMS メッセージで使用するすると不正または詐欺的に思われる場合もあります。メッセージの内容がベストプラクティスと一致しない場合、受信者はメッセージを無視する可能性があります。最悪の場合、携帯電話通信業者がメッセージをスパムと見なし、その電話番号からのメッセージを今後ブロックする可能性があります。

このセクションでは、効果的な SMS メッセージ本文を作成するためのヒントやアイデアをいくつか紹介します。

自分が送信者であることを明記してください

受信者は、メッセージがあなたから送信されたものであることをすぐに理解する必要があります。このベストプラクティスに従って、送信者は各メッセージの最初に識別名 (「プログラム名」) を含めます。

この操作はしないでください。

Your account has been accessed from a new device. Reply Y to confirm.
代わりにこれを試してください:

ExampleCorp Financial Alerts: You have logged in to your account from a new device. Reply Y to confirm, or STOP to opt-out.

メッセージを個人から個人へのメッセージのように記載しない

マーケティング担当者の中には、メッセージを個人から送信されたように見せて、SMS メッセージに個人的なタッチを加えたいと思う人もいます。ただし、この手法を使用すると、メッセージがフィッシング詐欺のように受け取られる場合があります。

この操作はしないでください。

Hi, this is Jane. Did you know that you can save up to 50% at Example.com? Click here for more info: <https://www.example.com>.

代わりにこれを試してください:

ExampleCorp Offers: Save 25-50% on sale items at Example.com. Click here to browse the sale: <https://www.example.com>. Text STOP to opt-out.

お金の話には注意する

詐欺師は、お金を貯めたり、受け取りたいという人の欲求につけ込むことがよくあります。「話がうますぎる」と思わせるようなオファーを記載してはいけません。人を欺くためにお金の誘惑を使用してはいけません。通貨記号を使ってお金を示してはいけません。

この操作はしないでください。

Save big \$\$\$ on your next car repair by going to <https://www.example.com>.

代わりにこれを試してください:

ExampleCorp Offers: Your ExampleCorp insurance policy gets you discounts at 2300+ repair shops nationwide. More info at <https://www.example.com>. Text STOP to opt-out.

必要な文字のみを使用する

ブランドは、多くの場合において、メッセージに ™ や ® などの商標記号を含めることで商標を保護する傾向があります。しかし、これらの記号は、160 文字の SMS メッセージに含めることができる (GSM アルファベットと呼ばれる) 標準文字セットには含まれていません。これらの文字のいずれかを含むメッセージを送信すると、メッセージは別の文字エンコーディングシステムを使用して自動送信されます。このシステムでは、メッセージパートごとに 70 文字までしかサポートされません。その結果、メッセージが複数に分割される可能性があります。送信するメッセージパーツごとに料金が請求されるため、メッセージ全体の送信にかかる費用が予想よりも高くなる可能性があります。さらに、受信者は 1 つのメッセージではなく、複数の連続したメッセージを受信する場合があります。SMS の文字エンコーディングの詳細については、「[Amazon SNS の SMS の文字数制限について](#)」を参照してください。

この操作はしないでください。

ExampleCorp Alerts: Save 20% when you buy a new ExampleCorp Widget® at example.com and use the promo code WIDGET.

代わりにこれを試してください:

ExampleCorp Alerts: Save 20% when you buy a new ExampleCorp Widget(R) at example.com and use the promo code WIDGET.

 Note

上記の 2 つの例はほとんど同じですが、最初の例には GSM アルファベットではない登録商標記号 (®) が含まれています。その結果、最初の例は 2 つのメッセージパーツとして送信され、2 番目の例は 1 つのメッセージパーツとして送信されます。

有効で安全なリンクを使用する

メッセージにリンクが含まれている場合は、そのリンクが機能することを再確認してください。企業ネットワーク外のデバイスでリンクをテストして、リンクが正しく解決されることを確認します。SMS メッセージは 160 文字に制限されているため、URL が非常に長いと複数のメッセージに分割される可能性があります。短縮 URL を指定するには、リダイレクトドメインを使用する必要があります。ただし、tinyurl.com や bitly.com などの無料のリンク短縮サービスを使用しないでください。通信事業者は、そのドメイン上のリンクを含むメッセージをフィルタリングする傾向があります。リンクが会社または組織の排他的使用専用のドメインを指している限り、有料のリンク短縮サービスを使用できます。

この操作はしないでください。

Go to <https://tinyurl.com/4585y8mr> today for a special offer!

代わりにこれを試してください:

ExampleCorp Offers: Today only, get an exclusive deal on an ExampleCorp Widget. See <https://a.co/cFKmaRG> for more info. Text STOP to opt-out.

使用する略語数を制限する

SMS チャネルには 160 文字の制限があるため、一部の送信者は、メッセージに略語を多用する必要がありますと考えます。しかし、略語を使いすぎると、多くの読者は企業からのものではないと受け止められ、一部のユーザーがメッセージをスパムとして報告する可能性があります。略語を多用せずに、首尾一貫したメッセージを書くことは至って可能です。

この操作はしないでください。

Get a gr8 deal on ExampleCorp widgets when u buy a 4-pack 2day.

代わりにこれを試してください:

ExampleCorp Alerts: Today only—an exclusive deal on ExampleCorp Widgets at example.com. Text STOP to opt-out.

適切に応答する

受信者がメッセージに返信したときは、有用な情報で応答するようにしてください。例えば、顧客がメッセージの1つにキーワード「HELP」で反応した場合、サブスクライブしているプログラム、毎月送信されるメッセージの数、詳細情報が必要な場合の連絡方法に関する情報を送信します。HELP レスポンスメッセージの例は次のとおりです。

HELP: ExampleCorp alerts: email help@example.com or call 425-555-0199. 2 msgs/month. Msg & data rates may apply. Reply STOP to cancel.

顧客がキーワード「STOP」で返信した場合、今後はメッセージが送信されないことを知らせます。STOP レスポンスメッセージの例は次のとおりです。

You're unsubscribed from ExampleCorp alerts. No more messages will be sent. Reply HELP, email help@example.com, or call 425-555-0199 for more info.

エンゲージメントに基づく送信を調整する

顧客の優先順位は時間の経過とともに変わる可能性があります。顧客がメッセージを必要としなくなった場合、メッセージを完全にオプトアウトしたり、メッセージを未承諾として報告したり可能性があります。このため、顧客とのエンゲージメントに基づいて送信手続きを調整することは重要です。

メッセージにめったに反応しない顧客の場合、メッセージの頻度を調整する必要があります。例えば、反応の多い顧客にメッセージを毎週送信している場合、反応の少ない顧客用に別の毎月のダイジェストを作成できます。

最後に、まったく反応のない顧客を顧客リストから削除します。このステップにより、顧客がメッセージを不満に感じることはなくなります。また、コストを削減し、送信者としての評判を維持することもできます。

適切な時間に送信する

通常の営業時間中にのみメッセージを送信します。メッセージを夕食や深夜の時間帯に送信している場合、顧客が邪魔をされないようにリストからサブスクリプション解除する可能性が高くなります。さらに、顧客がすぐに反応できないときに SMS メッセージを送信しても意味がありません。

キャンペーンやジャーニーを多くの受信者に送信する場合は、発信元番号のスループットレートを再確認してください。受信者数をスループットレートで割ると、すべての受信者にメッセージを送信するのにかかる時間が分かります。

複数チャンネルでの重複を回避する

キャンペーンで、複数の通信チャンネル (E メール、SMS、プッシュメッセージ) を使用している場合、各チャンネルで同じメッセージを送信しないでください。同じメッセージを複数のチャンネルで同時に送信すると、顧客は送信動作を有用ではなく迷惑だと受け取る可能性が高くなります。

専用ショートコードを使用する

ショートコードを使用している場合、ブランドおよびメッセージタイプごとに別個のショートコードを維持します。例えば、会社に 2 つのブランドがある場合、それぞれ別のショートコードを使用します。同様に、トランザクションメッセージとプロモーションメッセージの両方を送信している場合、メッセージタイプごとに別のショートコードを使用します。ショートコードのリクエストの詳細については、「AWS End User Messaging SMS ユーザーガイド」の「[AWS End User Messaging SMSでの SMS メッセージングのショートコードのリクエスト](#)」を参照してください。

送信先の電話番号を確認する

Amazon Pinpoint を通じて SMS メッセージを送信すると、送信したメッセージパートごとに課金されます。メッセージパーツごとに支払う料金は、受信者の国や地域によって異なります。SMS の料金の詳細については、「[AWS 世界各国の SMS 料金表](#)」を参照してください。

Amazon SMS が SMS メッセージを送信するリクエストを受け入れた場合 ([SendMessage](#) API、またはキャンペーンまたはジャーニーが開始された結果)、そのメッセージの送信に対して課金されます。このステートメントは、たとえ意図した受信者が実際にメッセージを受け取らなかったとしても有効です。例えば、相手の電話番号が使われていなかったり、送信先の電話番号が有効な携帯電話番号でなかったとしても、メッセージの送信は課金対象となります。

Amazon SNS は、SMS メッセージの有効な送信リクエストを受け付け、配信を試みます。このため、メッセージを送信する電話番号が有効な携帯電話番号であることを検証する必要があります。AWS End User Messaging SMS を使用してテストメッセージを送信し、電話番号が有効かどうか、

電話番号のタイプ (携帯電話、固定電話、VoIP など) を判断できます。詳細については、「AWS End User Messaging SMS ユーザーガイド」の「[SMS シミュレータを使用してテストメッセージを送信する](#)」を参照してください。

冗長性を考慮した設計

ミッションクリティカルなメッセージングプログラムでは、Amazon SNS を複数の AWS リージョンで設定することをお勧めします。Amazon SNS にはいくつかの [AWS リージョン](#) があります。Amazon SNS を使用できるリージョンの一覧については、「[AWS 全般のリファレンス](#)」を参照してください。

ショートコード、ロングコード、通話料無料、10DLC 番号など、SMS メッセージに使用する電話番号は AWS リージョンで複製することはできません。そのため、Amazon SNS を複数のリージョンで使用するには、Amazon SNS を使用するリージョンごとに個別の電話番号をリクエストする必要があります。たとえば、ショートコードを使用して米国の受信者にテキストメッセージを送信する場合は、AWS リージョン 使用する各 で個別のショートコードをリクエストする必要があります。

一部の国では、冗長性を高めるために複数のタイプの電話番号を使用することもできます。例えば、米国では、ショートコード、10DLC 番号、通話料無料の番号をリクエストすることができます。これらの電話番号タイプはそれぞれ、受信者へのルートが異なります。同じまたは複数の [AWS リージョン](#) またがる複数の電話番号タイプを使用できる AWS リージョンと、冗長性がさらに強化され、耐障害性の向上に役立ちます。

SMS の制限と規制

SMS の制限と規制については、「AWS End User Messaging SMS ユーザーガイド」の「[SMS および MMS の制限と規制](#)」を参照してください。

オプトアウトキーワードの管理

SMS 受信者は、各自のデバイスを使用して、キーワードで返信することでメッセージをオプトアウトできます。詳細については、「[SMS メッセージの受信をオプトアウトする](#)」を参照してください。

CreatePool

CreatePool API アクションを使用して新しいプールを作成し、そのプールと指定した送信元アイデンティティとを関連付けます。詳細については、「AWS End User Messaging SMS API リファレンス」の「[CreatePool](#)」を参照してください。

PutKeyword

PutKeyword API アクションを使用して、送信元の電話番号またはプールのキーワード設定を作成または更新します。詳細については、「AWS End User Messaging SMS API リファレンス」の「[PutKeyword](#)」を参照してください。

番号設定の管理

AWS サポートにリクエストし、アカウントに割り当てた専用のショートコードとロングコードの設定を管理するには、「」の「[で電話番号の機能を変更する AWS CLI](#)」を参照してくださいAWS End User Messaging SMS。

Amazon SNS の SMS の文字数制限について

1 回の SMS メッセージには、最大 140 バイトの情報を含めることができます。1 回の SMS メッセージに使用できる文字数は、メッセージに含む文字の種類によって異なります。

GSM 7-bit アルファベットとしても知られる「[GSM 03.38 文字セット](#)」のみメッセージに使用されている場合は、最大 160 文字含めることができます。メッセージに GSM 03.38 文字セット以外の文字を使用する場合は、最大 70 文字を含めることができます。SMS メッセージを送信する場合は、Amazon SNS によって最も効率的なエンコードが自動的に判断されます。

メッセージに最大文字数を超える文字を含めると、メッセージは複数のパートに分割されます。メッセージが複数のパートに分割されると、各パートには、その前のメッセージパートに関する追加情報が含まれます。受取人のデバイスは、このように分割されたメッセージパートを受信すると、この追加情報を使用して、すべてのメッセージパートを正しい順で表示します。受取人の携帯通信事業者やデバイスによっては、複数のメッセージが、1 つのメッセージとして表示されたり、個別のメッセージのシーケンスとして表示されたりする場合があります。その結果として、各メッセージパートの文字数は 153 (GSM 03.38 文字のみを含むメッセージの場合) または 67 (他の文字を含むメッセージの場合) に減少します。SMS の長さ計算ツールを使用すると、メッセージを送信する前にメッセージに含まれるメッセージパートの数を推定できます。これらのツールのいくつかはオンラインで入手できます。メッセージの最大サイズは、GSM 1600 文字または非 GSM 630 文字です。スループットとメッセージサイズの詳細については、Amazon Pinpoint ユーザーガイドの「[SMS character limits in Amazon Pinpoint](#)」(Amazon Pinpoint の SMS の文字数制限について) を参照してください。

送信する各メッセージのメッセージパート数を確認するには、まず[イベントストリーム設定](#)を有効にする必要があります。有効にすると、メッセージが受取人の携帯電話会社に配信されたときに、Amazon SNS によって _SMS.SUCCESS イベントが生成されます。_SMS.SUCCESS イベントレコードには、attributes.number_of_message_parts という属性が含まれています。この属性は、メッセージ内のメッセージパート数を指定します。

⚠ Important

複数のメッセージパートを含むメッセージを送信すると、メッセージ内のメッセージパート数に応じて課金されます。

GSM 03.38 文字セット

GSM 03.38 文字セットに含まれるすべての文字を次のテーブルに示します。次のテーブルに示す文字のみを含むメッセージを送信する場合、そのメッセージには最大 160 文字までを含めることができます。

GSM 03.38 標準文字												
A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ある	b	c	d	e	f	g	h	i	j	k	l	m
n	o	p	q	r	s	t	u	v	w	x	y	z
à	Å	å	Ä	ä	Ç	É	é	è	ì	Ñ	ñ	ò
Ø	ø	Ö	ö	ù	Ü	ü	Æ	æ	ß	0	1	2
3	4	5	6	7	8	9	&	*	@	:	,	¤
\$	=	!	>	#	-	ı	¿	(	<	%	.	+
£	?	"	)	§	;	'	/	_	¥	Δ	Φ	Γ
Λ	Ω	Π	Ψ	Σ	Θ	Ξ						

GSM 03.38 文字セットには、前述のテーブルに示す記号に加えて、複数の記号が含まれています。ただし、これらの文字には、表示されないエスケープ文字も含まれているため、それぞれ 2 文字としてカウントされます。

• ^

- {
- }
- \
- [
-]
- ~
- |
- €

最後に、GSM 03.38 文字セットには、次の非表示の文字も含まれています。

- 空白文字。
- 改行制御。テキストのある行の行末であるとともに、次の行の先頭であることを示します。
- キャリッジリターン制御。テキストの行の先頭に移動します (通常は次の改行文字)。
- エスケープ制御。前述のリストの文字に自動的に追加されます。

メッセージの例

このセクションでは、SMS メッセージの例をいくつか示します。例ごとに、メッセージのメッセージパート数と合計文字数を示します。

例 1: GSM 03.38 アルファベットの文字のみを含む長いメッセージ

次のメッセージには、GSM 03.38 アルファベットの文字のみが含まれています。

```
Hello Carlos. Your Example Corp. bill of $100 is now available. Autopay is scheduled for next Thursday, April 9. To view the details of your bill, go to https://example.com/bill11.
```

上のメッセージには 180 文字が含まれているため、複数のメッセージパートに分割する必要があります。メッセージを複数のメッセージパートに分割した場合、各パートには 153 文字の GSM 03.38 を含めることができます。その結果、このメッセージは 2 つのメッセージパートとして送信されます。

例 2: マルチバイト文字を含むメッセージ

次のメッセージには、GSM 03.38 アルファベットではない、複数の中国語の文字が含まれています。

```
#####.####1994#7#####
```

上のメッセージには 71 文字が含まれています。ただし、メッセージ内のほとんどすべての文字は GSM 03.38 アルファベットではないため、2 つのメッセージパートとして送信されます。メッセージパートごとに最大 67 文字を含めることができます。

例 3: GSM 以外の文字を 1 つ含むメッセージ

次のメッセージには、GSM 03.38 アルファベットに属さない文字が 1 つ含まれています。この例で、その文字は閉じる単一引用符 (') であり、通常のアポストロフィ (') とは別の文字です。Microsoft Word などのワープロアプリケーションは、アポストロフィを、閉じる単一引用符に自動的に置き換える場合があります。SMS メッセージの下書きを Microsoft Word で作成して Amazon SNS に貼り付ける場合は、これらの特殊文字を削除して、アポストロフィに置き換えてください。

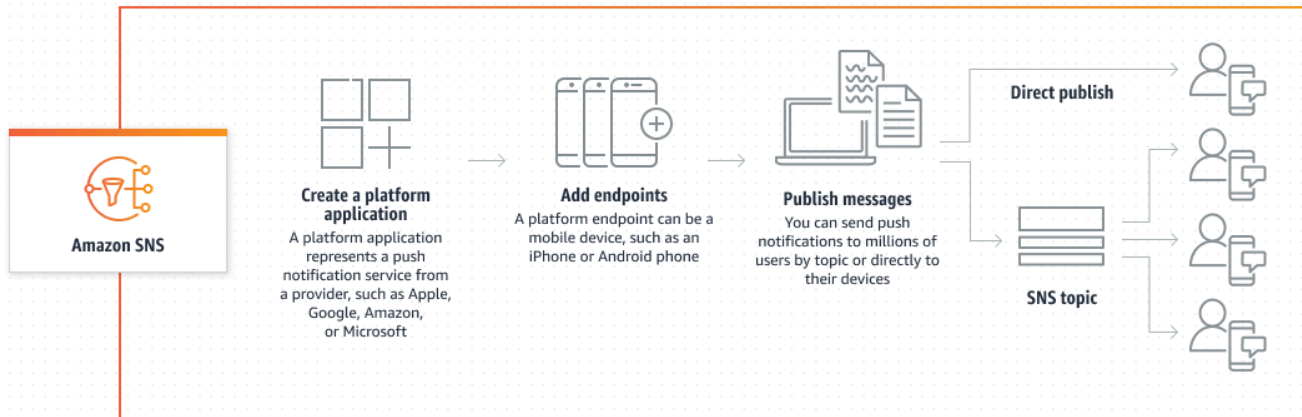
John: Your appointment with Dr. Salazar's office is scheduled for next Thursday at 4:30pm. Reply YES to confirm, NO to reschedule.

上のメッセージには 130 文字が含まれています。ただし、GSM 03.38 アルファベットに属さない閉じる単一引用符文字が含まれているため、メッセージは 2 つのメッセージパートとして送信されます。

このメッセージの閉じる単一引用符をアポストロフィ (GSM 03.38 アルファベットの一部) に置き換えると、メッセージは単一のメッセージパートとして送信されます。

Amazon SNS でモバイルプッシュ通知を送信する

Amazon SNS を使用して、プッシュ通知メッセージをモバイルデバイスのアプリに直接送信できます。モバイルエンドポイントに送信されたプッシュ通知メッセージは、メッセージアラート、バッジ更新、または音声アラートとしてモバイルアプリケーションに表示できます。



トピック

- [Amazon SNS ユーザー通知の仕組み](#)
- [Amazon SNS でプッシュ通知を設定する](#)
- [Amazon SNS でのモバイルアプリのセットアップ](#)
- [Amazon SNS を使用したモバイルプッシュ通知](#)
- [Amazon SNS モバイルアプリの属性](#)
- [モバイルアプリケーションの Amazon SNS アプリケーションイベント通知](#)
- [モバイルプッシュ API アクション](#)
- [Amazon SNS モバイルプッシュ API の一般的なエラー](#)
- [Amazon SNS のモバイルプッシュ通知で有効期限メッセージ属性を使用する](#)
- [Amazon SNS モバイルアプリケーションでサポートされるリージョン](#)
- [Amazon SNS モバイルプッシュ通知を管理するためのベストプラクティス](#)

Amazon SNS ユーザー通知の仕組み

以下のサポートされているプッシュ通知サービスのいずれかを使用して、モバイルデバイスとデスクトップの両方にプッシュ通知メッセージを送信します。

- Amazon Device Messaging (ADM)
- iOS と Mac OS Xの両方向け Apple Push Notification Service (APNs)
- Baidu Cloud Push (Baidu)
- Firebase Cloud Messaging (FCM)

- Windows Phone (MPNS) 用 Microsoft Push Notification Service (MPNS)
- Windows Push Notification Services (WNS)

APN や FCM などのプッシュ通知サービスは、各アプリケーションと、サービスを使用するために登録されている関連モバイルデバイスとの接続を維持します。アプリとモバイルデバイスの登録時に、プッシュ通知サービスはデバイストークンを返します。Amazon SNS はデバイストークンを使用して、モバイルエンドポイントを作成します。ここに直接プッシュ通知メッセージを送信できます。Amazon SNS が他のプッシュ通知サービスと通信できるようにするには、プッシュ通知サービス認証情報を Amazon SNS に送信して、代理で使えるようにします。詳細については、「[Amazon SNS でプッシュ通知を設定する](#)」を参照してください。

直接プッシュ通知メッセージを送信するだけでなく、Amazon SNS を使用して、トピックにサブスクライブされているモバイルエンドポイントにメッセージを送信することもできます。このコンセプトは、「[Amazon SNS とは](#)」に説明されているように、Amazon SQS、HTTP/S、E メール、SMS などその他のエンドポイントタイプをトピックにサブスクライブするのと同じです。違いは、Amazon SNS はサブスクライブされたモバイルエンドポイントがトピックに送信されたプッシュ通知メッセージを受信するためにプッシュ通知サービスを通じて通信を行うことです。

Amazon SNS でプッシュ通知を設定する

1. サポートするモバイルプラットフォームの[認証情報とデバイストークンを取得](#)します。
2. 認証情報を使用して、Amazon SNS でプラットフォームアプリケーションオブジェクト (PlatformApplicationArn) を作成します。詳細については、「[Amazon SNS プラットフォームアプリケーションの作成](#)」を参照してください。
3. 返された認証情報を使用して、プッシュ通知サービスから、モバイルアプリケーションおよびデバイス用のデバイストークンをリクエストします。受け取ったトークンはモバイルアプリケーションおよびデバイスの識別に使用されます。
4. デバイストークンと PlatformApplicationArn を使用し、Amazon SNS でプラットフォームエンドポイントオブジェクト (EndpointArn) を作成します。詳細については、「[モバイル通知用の Amazon SNS プラットフォームエンドポイントの設定](#)」を参照してください。
5. EndpointArn を使用して、[モバイルデバイスのアプリケーションにメッセージを発行](#)します。詳細については、『Amazon Simple Notification Service API リファレンス』の「[Amazon SNS モバイルデバイスの直接メッセージング](#)」および「[公開 API](#)」を参照してください。

Amazon SNS でのモバイルアプリのセットアップ

このトピックでは、[Amazon SNS ユーザー通知の前提条件](#) で説明されている情報を使用して、AWS マネジメントコンソール でモバイルアプリケーションをセットアップする方法について説明します。

Amazon SNS ユーザー通知の前提条件

Amazon SNS モバイルプッシュ通知の使用を開始するには、以下が必要です。

- サポートされているプッシュ通知サービスのいずれかに接続するための認証情報セット: ADM、APN、Baidu、FCM、MPNSまたは WNS。
- モバイルアプリケーションやデバイスのデバイストークンまたは登録 ID。
- モバイルエンドポイントにプッシュ通知メッセージを送信するように設定された Amazon SNS。
- 登録済みで、サポートされているいずれかのプッシュ通知サービスを使用するように設定されたモバイルアプリケーション。

アプリケーションをプッシュ通知サービスに登録するには、いくつかのステップが必要です。Amazon SNS では、モバイルエンドポイントに直接プッシュ通知メッセージを送信するために、ユーザーがプッシュ通知サービスに対して指定する情報の一部が必要です。一般的に、プッシュ通知サービスに接続するための必須認証情報、プッシュ通知サービスから受け取ったデバイストークンまたは登録 ID(モバイルデバイスおよびモバイルアプリケーションを表すもの)、およびプッシュ通知サービスに登録されているモバイルアプリケーションが必要です。

認証情報の正確な形式はモバイルプラットフォームごとに異なりますが、どの場合も、これらの認証情報はプラットフォームに接続するときに送信する必要があります。モバイルアプリケーションごとに 1 つの認証情報セットが発行され、そのアプリケーションのどのインスタンスにメッセージを送信する際もその認証情報セットを使用する必要があります。

具体的な名前は、どのプッシュ通知サービスが使用されるかによって異なります。例えば、プッシュ通知サービスとして APN を使用する場合は、デバイストークンが必要になります。また、FCM を使用する場合は、デバイストークンに相当するものが登録 ID と呼ばれます。デバイストークンまたは登録 ID は、モバイルデバイスのオペレーティングシステムによってアプリケーションに送信される文字列です。特定のモバイルデバイスで実行されるモバイルアプリケーションのインスタンスを一意に識別するため、このアプリケーション/デバイスペアの一意識別子と考えることができます。

Amazon SNS は、認証情報 (および他のいくつかの設定) をプラットフォームアプリケーションリソースとして保存します。デバイストークン (ここでもいくつかの追加設定を使用) は、プラット

フォームエンドポイントと呼ばれるオブジェクトとして表現されます。各プラットフォームエンドポイントは、1つの特定のプラットフォームアプリケーションに属し、対応するプラットフォームアプリケーションに保存された認証情報を使用してすべてのプラットフォームエンドポイントに通信できます。

以降のセクションでは、サポートされているプッシュ通知サービスごとの前提条件が示されています。前提条件情報を取得したら、AWS マネジメントコンソール または Amazon SNS モバイルプッシュ APIs を使用してプッシュ通知メッセージを送信できます。詳細については、「[Amazon SNS でプッシュ通知を設定する](#)」を参照してください。

Amazon SNS プラットフォームアプリケーションの作成

Amazon SNS からモバイルエンドポイントへ通知を送信するには、直接送信する場合でも、トピックへのサブスクリプションを介して送信する場合でも、最初にプラットフォームアプリケーションを作成する必要があります。アプリを AWS に登録したら、そのアプリとモバイルデバイスの両方についてエンドポイントを作成する必要があります。このエンドポイントにより、Amazon SNS はデバイスにメッセージを送信できます。

プラットフォームアプリケーションを作成するには

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションペインで、[プッシュ通知] を選択します。
3. [Platform applications] (プラットフォームアプリケーション) セクションで、[Create platform application] (プラットフォームアプリケーションの作成) を選択します。
4. を選択します。。AWS リージョンモバイルアプリケーションを作成できる AWS リージョンのリストについては、「[Amazon SNS モバイルアプリケーションでサポートされるリージョン](#)」を参照してください。
5. 次のアプリケーションの詳細を入力します。
 - アプリケーション名 – プラットフォームアプリケーションの名前を指定します。この名前は、1~256 文字の範囲で指定する必要があり、大文字/小文字のアルファベット、数字、アンダースコア、ハイフン、およびピリオドを含めることができます。
 - プッシュ通知プラットフォーム – アプリが登録されている適切な通知サービスを選択します (Apple Push Notification Service (APNs)、Firebase Cloud Messaging (FCM) など)。
6. 選択したプラットフォームに応じて、特定の認証情報を指定する必要があります。
 - APNs (Apple Push Notification Service) の場合 – トークンベースの認証または証明書ベースの認証のいずれかを選択します。

- トークンベースの認証の場合は、.p8 ファイル (キーチェーンアクセスを介して生成されます) をアップロードします。
 - 証明書ベースの認証の場合は、.p12 ファイル (キーチェーンアクセスからエクスポートされます) をアップロードします。
 - FCM (Firebase Cloud Messaging) の場合 – Firebase コンソールのサーバーキーを入力します。
 - 他のプラットフォーム (ADM や GCM など) の場合 – それぞれの API キーまたは認証情報を入力します。
7. 必要な詳細を入力したら、[プラットフォームアプリケーションの作成]を選択します。このアクションは、アプリを Amazon SNS に登録し、対応するプラットフォームアプリケーションオブジェクトを作成します。
 8. 作成時に、Amazon SNS は [PlatformApplicationArn](#) (Amazon リソースネーム) を生成して返します。この ARN はプラットフォームアプリケーションを一意に識別するもので、モバイルデバイスのエンドポイントを作成するときに使用されます。

モバイル通知用の Amazon SNS プラットフォームエンドポイントの設定

アプリやモバイルデバイスをプッシュ通知サービス (APNs や Firebase Cloud Messaging など) に登録するとき、プッシュ通知サービスはデバイストークンを返します。Amazon SNS はこのデバイストークンを使用してプラットフォームエンドポイントを作成します。プラットフォームエンドポイントは、直接プッシュ通知メッセージをデバイス上のアプリに送信するためのターゲットとして機能します。プラットフォームエンドポイントはブリッジとして機能し、Amazon SNS からプッシュ通知サービスに送信されるメッセージをルーティングして、対応するモバイルデバイスに配信します。詳細については、「[Amazon SNS ユーザー通知の前提条件](#)」および「[Amazon SNS でプッシュ通知を設定する](#)」を参照してください。

デバイストークンとプラットフォームエンドポイントについて

デバイストークンは、プッシュ通知サービス (APNs、Firebase Cloud Messaging など) に登録されたモバイルデバイスを一意に識別します。アプリをプッシュ通知サービスに登録するとき、そのアプリとデバイスに固有のデバイストークンが生成されます。Amazon SNS はこのデバイストークンを使用して、対応するプラットフォームアプリケーション内にプラットフォームエンドポイントを作成します。

プラットフォームエンドポイントによって、Amazon SNS はプッシュ通知サービスを介してデバイスにプッシュ通知メッセージを送信でき、その際にアプリとユーザーのデバイス間の接続を維持することができます。

プラットフォームエンドポイントの作成

Amazon SNS を使用してアプリケーションにプッシュ通知を送信するには、まずプラットフォームエンドポイントの作成アクションを呼び出すことで、そのアプリケーションのデバイストークンを Amazon SNS に登録する必要があります。このアクションは、パラメーターとしてプラットフォームアプリケーションの Amazon リソースネーム (ARN) およびデバイストークンを使用し、作成されたプラットフォームエンドポイントの ARN を返します。

[CreatePlatformEndpoint](#) アクションでは、次の処理が行われます。

- プラットフォームエンドポイントが既に存在する場合、再度作成しないでください。呼び出し元に既存のプラットフォームエンドポイントの ARN を返します。
- デバイストークンが同じだが設定が異なるプラットフォームエンドポイントが既に存在する場合、再度作成しないでください。呼び出し元に例外をスローします。
- プラットフォームエンドポイントが存在しない場合は、作成します。呼び出し元に新しく作成したプラットフォームエンドポイントの ARN を返します。

このアプローチでは、作業エンドポイントが常に提供されるとは限らないため、アプリケーションが起動するたびにプラットフォームエンドポイントの作成アクションを呼び出さないでください。これは、例えばアプリケーションがアンインストールされて同じデバイスに再インストールされ、そのアプリケーションのエンドポイントが既に存在しているが無効な場合などに発生する可能性があります。登録プロセスに成功すると、以下のことが達成されます。

- このアプリケーションデバイスの組み合わせにプラットフォームエンドポイントが存在することを確認します。
- プラットフォームエンドポイントのデバイストークンが最新の有効なデバイストークンであることを確認します。
- プラットフォームエンドポイントが有効であり、使用できる状態にあることを確認します。

擬似コード

次の擬似コードは、さまざまな開始条件において有効な最新の作業プラットフォームエンドポイントを作成するための推奨される方法について説明します。このアプローチは、アプリケーションが初めて登録されるかどうか、このアプリケーションのプラットフォームエンドポイントが既に存在するかどうか、プラットフォームエンドポイントが有効かどうか、適切なデバイストークンがあるかどうかなどに関係なく機能します。重複するプラットフォームエンドポイントが作成されたり、既に最新の

プラットフォームエンドポイントが有効になっている場合に既存のプラットフォームエンドポイントが変更されたりしないため、連続して複数回呼び出しても安全です。

```
retrieve the latest device token from the mobile operating system
if (the platform endpoint ARN is not stored)
    # this is a first-time registration
    call create platform endpoint
    store the returned platform endpoint ARN
endif

call get endpoint attributes on the platform endpoint ARN

if (while getting the attributes a not-found exception is thrown)
    # the platform endpoint was deleted
    call create platform endpoint with the latest device token
    store the returned platform endpoint ARN
else
    if (the device token in the endpoint does not match the latest one) or
        (GetEndpointAttributes shows the endpoint as disabled)
        call set endpoint attributes to set the latest device token and then enable the
        platform endpoint
    endif
endif
```

このアプローチは、アプリケーションが自身を登録または再登録するときいつでも使用できます。また、デバイストークンの変更について Amazon SNS に通知するときも使用できます。この場合、最新のデバイストークン値を持つアクションを呼び出すだけです。このアプローチについて注意が必要な点は次のとおりです。

- プラットフォームエンドポイントの作成アクションの呼び出しが考えられる 2 つのケースがあります。アプリケーションが自身のプラットフォームエンドポイント ARN を認識していない、ごく最初の時点で呼び出される可能性があります。これは初回登録時に発生します。さらに、最初の GetEndpointAttributes アクションが not-found 例外で失敗した場合にも呼び出されます。これは、アプリケーションがエンドポイント ARN を認識しているが、削除された場合に発生する可能性があります。
- GetEndpointAttributes アクションは、プラットフォームエンドポイントが作成されたばかりの場合でもプラットフォームエンドポイントの状態を確認するために呼び出されます。これは、プラットフォームエンドポイントが既に存在するが無効になっている場合に発生します。この場合、プラットフォームエンドポイントの作成アクションが成功しますが、プラットフォームエンドポイ

ントは有効にならないため、成功を返す前にプラットフォームエンドポイントの状態をもう一度確認する必要があります。

AWS SDK の例

次のコードは、AWS SDKs によって提供される Amazon SNS クライアントを使用して以前の擬似コードを実装する方法を示しています。

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、「AWS SDK とツールのリファレンスガイド」の「[共有設定ファイルと認証情報ファイル](#)」を参照してください。

CLI

AWS CLI

プラットフォームアプリケーションのエンドポイントを作成するには

次の `create-platform-endpoint` の例では、指定したトークンを使用して、指定したプラットフォームアプリケーションのエンドポイントを作成します。

```
aws sns create-platform-endpoint \
  --platform-application-arn arn:aws:sns:us-west-2:123456789012:app/GCM/  
MyApplication \
  --token EXAMPLE12345...
```

出力:

```
{
  "EndpointArn": "arn:aws:sns:us-west-2:1234567890:endpoint/GCM/MyApplication/12345678-abcd-9012-efgh-345678901234"
}
```


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.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreatePlatformEndpointRequest;
import software.amazon.awssdk.services.sns.model.CreatePlatformEndpointResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 *
 * In addition, create a platform application using the AWS Management Console.
 * See this doc topic:
 *
 * https://docs.aws.amazon.com/sns/latest/dg/mobile-push-send-register.html
 *
 * Without the values created by following the previous link, this code examples
 * does not work.
 */

public class RegistrationExample {
    public static void main(String[] args) {
        final String usage = ""

            Usage:      <token> <platformApplicationArn>

            Where:
```

token - The device token or registration ID of the mobile device. This is a unique identifier provided by the device platform (e.g., Apple Push Notification Service (APNS) for iOS devices, Firebase Cloud Messaging (FCM) for Android devices) when the mobile app is registered to receive push notifications.

platformApplicationArn - The ARN value of platform application. You can get this value from the AWS Management Console.\s

```
""";

if (args.length != 2) {
    System.out.println(usage);
    return;
}

String token = args[0];
String platformApplicationArn = args[1];
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();

createEndpoint(snsClient, token, platformApplicationArn);
}

public static void createEndpoint(SnsClient snsClient, String token, String
platformApplicationArn) {
    System.out.println("Creating platform endpoint with token " + token);
    try {
        CreatePlatformEndpointRequest endpointRequest =
CreatePlatformEndpointRequest.builder()
            .token(token)
            .platformApplicationArn(platformApplicationArn)
            .build();

        CreatePlatformEndpointResponse response =
snsClient.createPlatformEndpoint(endpointRequest);
        System.out.println("The ARN of the endpoint is " +
response.endpointArn());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
    }
}
```

```
}
```

詳細については、「[モバイルプッシュ API アクション](#)」を参照してください。

トラブルシューティング

古いデバイストークンを使用してプラットフォームエンドポイントの作成を繰り返し呼び出す

特に FCM エンドポイントの場合、アプリケーションを発行した最初のデバイストークンを保存してから、アプリケーションが起動するたびにそのデバイストークンを使用してプラットフォームエンドポイントの作成を呼び出すのが最適に思えるかもしれません。これは、アプリケーションがデバイストークンの状態を管理する必要がなくなり、Amazon SNS がサービストークンを最新の値に自動的に更新するため、適切に思えるかもしれません。しかし、この解決策には多くの深刻な問題があります。

- Amazon SNS は、有効期限切れのデバイストークンを新しいデバイストークンに更新するために FCM からのフィードバックが必要です。FCM は、古いデバイストークンの情報を当分の間保持しますが、無期限に保持するわけではありません。FCM が古いデバイストークンと新しいデバイストークンの関連性の認識を失うと、Amazon SNS はプラットフォームエンドポイントに保存されたデバイストークンを適切な値に更新することができなくなります。代わりに、プラットフォームエンドポイントが無効になるだけです。
- プラットフォームアプリケーションには、同じデバイストークンに対応する複数のプラットフォームエンドポイントが含まれます。
- Amazon SNS では、同じデバイストークンを使用して作成できるプラットフォームエンドポイントの数に限度が適用されます。最終的に、新しいエンドポイントの作成は無効なパラメータの例外によって失敗し、「This endpoint is already registered with a different token.」というエラーメッセージが表示されます。

FCM エンドポイントの管理の詳細については、「[Amazon SNS の Firebase Cloud Messaging エンドポイントの管理](#)」を参照してください。

無効なデバイストークンに関連付けられたプラットフォームエンドポイントを再度有効にする

モバイルプラットフォーム (APN や FCM など) が、発行リクエストで使用されたデバイストークンが無効であったことを Amazon SNS に通知すると、Amazon SNS はそのデバイストークンに関連付けられたプラットフォームエンドポイントを無効にします。その後、Amazon SNS はそのデバイストークンへのその後の発行を拒否します。プラットフォームエンドポイントを再度に有効にして発

行を継続すれば最適と思うかもしれませんが、ほとんどの場合これはうまくいきません。発行されるメッセージは配信されず、プラットフォームエンドポイントはその後まもなく無効になります。

これは、プラットフォームエンドポイントに関連付けられたデバイストークンが間違いなく無効であるためです。プラットフォームエンドポイントは、インストールされているどのアプリケーションにも応答しないため、配信が成功することはありません。次回発行されると、モバイルプラットフォームはデバイストークンが無効であることを Amazon SNS にもう一度通知し、Amazon SNS はプラットフォームエンドポイントをもう一度無効にします。

無効なプラットフォームエンドポイントを再度有効にするには、有効なデバイストークンに関連付けた後 (エンドポイント属性の設定アクションを呼び出して)、有効にする必要があります。その場合のみ、そのプラットフォームエンドポイントへの配信は正常に行われます。デバイストークンを更新しないでプラットフォームエンドポイントを有効にできるのは、そのエンドポイントに関連付けられたデバイストークンが無効であったが、再度有効となった場合のみです。これは、例えばアプリケーションがアンインストールされて同じモバイルデバイスに再インストールされ、同じデバイストークンを受け取った場合などに発生します。上に示したアプローチではこれが行われます。関連付けられたデバイストークンが使用可能な最新のものであると確認してから、プラットフォームエンドポイントを再度有効にすることのみ確実に行ってください。

モバイル通知用のデバイストークンと Amazon SNS の統合

Apple Push Notification Service (APNs) や Firebase Cloud Messaging (FCM) などの通知サービスに初めてアプリケーションやモバイルデバイスを登録すると、それらサービスからデバイストークンまたは登録 ID が返されます。これらのトークンや ID は Amazon SNS に追加され、[PlatformApplicationArn](#) API を使用して、アプリやデバイス向けのエンドポイントが作成されます。エンドポイントが作成されると、[EndpointArn](#) が返されます。Amazon SNS はこれを使用して、通知を適切なアプリケーションやデバイスに送信します。

次の方法を使用して、デバイストークンや登録 ID を Amazon SNS に追加できます。

- を介して 1 つのトークンを手動で追加する AWS マネジメントコンソール
- [CreatePlatformEndpoint](#) API を使用して複数のトークンをアップロードする
- 今後のデバイスのトークンを登録する

デバイストークンまたは登録 ID を手動で追加するには

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションペインで、[プッシュ通知] を選択します。

3. [プラットフォームアプリケーション] セクションでアプリケーションを選択し、[編集] を選択します。プラットフォームアプリケーションをまだ作成していない場合は、「Amazon SNS プラットフォームアプリケーションの作成」[Amazon SNS プラットフォームアプリケーションの作成](#)ガイドに従って今すぐ作成してください。
4. エンドポイントの作成 を選択します。
5. 使用している通知サービスに応じて、[エンドポイントトークン] ボックスにトークンまたは登録 ID を入力します (例: FCM 登録 ID)。
6. (オプション) [ユーザーデータ] フィールドに追加のデータを入力します。このデータは UTF-8 でエンコードされ、2 KB 未満である必要があります。
7. エンドポイントの作成 を選択します。

エンドポイントが作成されると、そのエンドポイントの対象となるモバイルデバイス、または Amazon SNS トピックにサブスクライブされているモバイルデバイスにメッセージを直接送信することができます。

CreatePlatformEndpoint API を使用して複数のトークンをアップロードするには

次の手順は、が提供するサンプル Java アプリ (bulkupload パッケージ) を使用して AWS、複数のトークン (デバイストークンまたは登録 IDs) を Amazon SNS にアップロードする方法を示しています。既存のトークンのアップロードを開始するには、このサンプルアプリを使用できます。

 Note

以下の手順では、Eclipse Java IDE を使用します。この手順では、をインストール AWS SDK for Java し、AWS のセキュリティ認証情報があることを前提としています AWS アカウント。詳細については、「[AWS SDK for Java](#)」を参照してください。認証情報の詳細については、「IAM ユーザーガイド」の「[AWS セキュリティ認証情報](#)」を参照してください。

1. [snsmobilepush.zip](#) ファイルをダウンロードして解凍します。
2. Eclipse で新しい Java プロジェクトを作成し、SNSSamples フォルダをプロジェクトにインポートします。
3. [OpenCSV library](#) をダウンロードし、ビルドパスに追加します。
4. BulkUpload.properties ファイルで、次のように指定します。
 - ApplicationArn (プラットフォームアプリケーション ARN)。

- トークンを含む CSV ファイルへの絶対パス。
- 成功したトークンと失敗したトークンのログファイル名。例えば、goodTokens.csv と badTokens.csv です。
- (オプション) 区切り文字、引用符文字、使用するスレッドの数の設定。

完了した BulkUpload.properties は次のようになります。

```
applicationarn: arn:aws:sns:us-west-2:111122223333:app/FCM/fcmpushapp
csvfilename: C:\\mytokendirectory\\mytokens.csv
goodfilename: C:\\mylogfiles\\goodtokens.csv
badfilename: C:\\mylogfiles\\badtokens.csv
delimiterchar: ','
quotechar: '"'
numofthreads: 5
```

5. BatchCreatePlatformEndpointSample.java アプリケーションを実行してトークンを Amazon SNS にアップロードします。正常にアップロードされたトークンは goodTokens.csv に記録され、不正な形式のトークンは badTokens.csv に記録されます。

今後のアプリケーションインストールのためにデバイスからトークンを登録するには

このプロセスには 2 つのオプションがあります。

Amazon Cognito サービスを使用する

モバイルアプリは、一時的なセキュリティ認証情報を使用してエンドポイントを作成できます。一時的な認証情報を生成するには、Amazon Cognito をお勧めします。詳細については、「[Amazon Cognito デベロッパーガイド](#)」を参照してください。

アプリの登録を追跡するには、Amazon SNS イベントを使用して、新しいエンドポイント ARNs の作成時に通知を受信します。

また、[ListEndpointByPlatformApplication](#) API を使用して、登録されたエンドポイントのリストを取得することもできます。

プロキシサーバーの使用

アプリケーションのインフラストラクチャがインストール時のデバイス登録を既にサポートしている場合は、お使いのサーバーをプロキシとして使用できます。デバイストークンは [CreatePlatformEndpoint](#) API を介して Amazon SNS に転送されます。

Amazon SNS によって作成されたエンドポイント ARN が返され、今後のメッセージを発行するためにサーバーで保存できます。

Amazon SNS Apple プッシュ通知認証方法

アプリのデベロッパーであることを識別する情報を提供することで、Amazon SNS に iOS または macOS アプリへのプッシュ通知の送信を許可できます。認証するには、[プラットフォームアプリケーションを作成するとき](#)にキーまたは証明書 のいずれかを提供します。どちらも Apple Developer アカウントから取得できます。

トークン署名キー

Amazon SNS が Apple Push Notification Service (APN) 認証トークンの署名に使用するプライベート署名キー。

署名キーを提供する場合、Amazon SNS はトークンを使用して、送信するプッシュ通知ごとに APN を使用して認証を行います。署名キーを使用すると、APN 本番環境とサンドボックス環境に通知を送信できます。

署名キーの有効期限は失効しないため、複数のアプリに対して同じ署名キーを使用できます。詳しくは、Apple ウェブサイトの [デベロッパーアカウントヘルプ セクションの「認証トークンを使用した APN との通信」](#)を参照してください。

証明書

プッシュ通知を送信するときに Amazon SNS が APN と認証するために使用する TLS 証明書です。証明書は Apple デベロッパーアカウントから取得できます。

証明書は 1 年後に失効します。この場合、新しい証明書を作成し、それを Amazon SNS に提供する必要があります。詳細については、Apple デベロッパーウェブサイトの「[APN への証明書ベースの接続の確立](#)」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て APNs 設定を管理するには

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションペインで、[プッシュ通知] を選択します。
3. [プラットフォームアプリケーション] セクションで、APNs の設定を編集するアプリケーションを選択し、[編集] を選択します。プラットフォームアプリケーションをまだ作成していない

場合は、「Amazon SNS プラットフォームアプリケーションの作成」[Amazon SNS プラットフォームアプリケーションの作成](#)ガイドに従って今すぐ作成してください。

4. [編集] を選択して、プラットフォームアプリケーションの設定を変更します。
5. [認証タイプ] セクションで、次のいずれかのオプションを選択します。
 - トークンベースの認証 (最新の APNs 統合に推奨)
 - 証明書ベースの認証 (従来の方法)
6. 認証タイプに基づいて認証情報を設定します。
 - トークンベースの認証の場合:
 - Apple デベロッパーアカウントからダウンロードした認証トークン署名キーである .p8 ファイルをアップロードします。
 - Apple デベロッパーアカウントで確認できる署名キー ID を入力します。[証明書]、[ID とプロファイル]、[キー] の順に移動し、使用するキーを選択します。
 - Apple デベロッパーアカウントのチーム識別子 を指定します。この識別子は、[メンバーシップ] ページで確認できます。
 - アプリに割り当てられたバンドル識別子を入力します。この識別子は、[証明書]、[ID とプロファイル]、[アプリ ID] で確認できます。
 - 証明書ベースの認証の場合:
 - TLS 証明書の .p12 ファイルをアップロードします。このファイルは、Apple デベロッパーアカウントから証明書をダウンロードした後で、macOS のキーチェーンアクセスからエクスポートできます。
 - パスワードを p12 証明書に割り当てている場合は、そのパスワードをここに入力します。
7. 必要な認証情報を入力したら、[変更の保存] を選択して設定を更新します。

Amazon SNS と Firebase Cloud Messaging の認証設定の統合

このトピックでは、AWS API および で使用するために必要な FCM API (HTTP v1) 認証情報を Google から取得する方法について説明します AWS CLI AWS マネジメントコンソール。

Important

2024 年 3 月 26 日、Amazon SNS は、Apple デバイスと Webpush 送信先用の FCM HTTP v1 API をサポートします。アプリケーションの中断を避けるため、2024 年 6 月 1 日または

それ以前に、既存のモバイルプッシュアプリケーションを最新の FCM HTTP v1 API に移行することをお勧めします。

2024 年 1 月 18 日、Amazon SNS は、Android デバイスへのモバイルプッシュ通知配信の FCM HTTP v1 API のサポートを導入しました。

2023 年 6 月 20 日、Google は Firebase Cloud Messaging (FCM) のレガシー HTTP API を廃止しました。Amazon SNS は、FCM HTTP v1 API を使用したすべてのデバイスタイプへの配信をサポートするようになりました。中断を避けるため、2024 年 6 月 1 日またはそれ以前に、既存のモバイルプッシュアプリケーションを最新の FCM HTTP v1 API に移行することをお勧めします。

アプリケーションのデベロッパーであることを識別する情報を提供することで、Amazon SNS にアプリケーションへのプッシュ通知の送信を許可できます。認証するには、[プラットフォームアプリケーションの作成時](#)に API キーまたはトークンのいずれかを指定します。次の情報は [Firebase アプリケーションコンソール](#)から取得できます。

API キー

API キーは Firebase のレガシー API を呼び出すときに使用される認証情報です。FCM レガシー API は 2024 年 6 月 20 日に Google によって廃止されます。現在 API キーをプラットフォーム認証情報として使用している場合は、オプションとして [トークン] を選択し、Firebase アプリケーションに関連する JSON ファイルをアップロードすることで、プラットフォーム認証情報を更新できます。

Token

HTTP v1 API を呼び出す際には、有効期間の短いアクセストークンが使用されます。これは Firebase が推奨するプッシュ通知の送信用の API です。Firebase はアクセストークンを生成するために、プライベートキーファイル (service.json ファイルとも呼ばれます) の形式でデベロッパーに認証情報のセットを提供します。

前提条件

Amazon SNS で FCM の設定の管理を開始する前に、FCM service.json 認証情報を取得する必要があります。service.json 認証情報を取得するには、Google Firebase ドキュメントの「[以前の HTTP から HTTP v1 に移行する](#)」を参照してください。

CLI を使用した FCM の設定の管理

AWS API を使用して FCM プッシュ通知を作成できます。AWS アカウント内の Amazon SNS リソースの数とサイズは限られています。詳細については、「AWS 全般のリファレンス ガイド」の「[Amazon Simple Notification Service のエンドポイントとクォータ](#)」を参照してください。

Amazon SNS トピック (API) とともに FCM プッシュ通知を作成するには AWS

キー認証情報を使用する場合、PlatformCredential は API key です。トークン認証情報を使用する場合、PlatformCredential は JSON 形式のプライベートキーファイルです。

- [CreatePlatformApplication](#)

既存の Amazon SNS トピックの FCM 認証情報タイプを取得するには (AWS API)

認証情報タイプ "AuthenticationMethod": "Token" または "AuthenticationMethod": "Key" を取得します。

- [GetPlatformApplicationAttributes](#)

既存の Amazon SNS トピックの FCM 属性を設定するには (AWS API)

FCM 属性を設定します。

- [SetPlatformApplicationAttributes](#)

コンソールを使用して FCM の設定を管理する

(AWS Command Line Interface CLI) を使用して FCM プッシュ通知を作成できます。AWS アカウント内の Amazon SNS リソースの数とサイズは限られています。詳細については、「[Amazon Simple Notification Service のエンドポイントとクォータ](#)」を参照してください。

Amazon SNS トピックと共に FCM プッシュ通知を作成するには (AWS CLI)

キー認証情報を使用する場合、PlatformCredential は API key です。トークン認証情報を使用する場合、PlatformCredential は JSON 形式のプライベートキーファイルです。AWS CLI を使用する場合、ファイルは文字列形式であり、特殊文字は無視する必要があります。ファイルを正しくフォーマットするために、Amazon SNS では次のコマンドを使用することを推奨しています:

```
SERVICE_JSON=`jq @json <<< cat service.json`:
```

- [create-platform-application](#)

既存の Amazon SNS トピックの FCM 認証情報タイプを取得するには (AWS CLI)

認証情報タイプ "AuthenticationMethod": "Token" または "AuthenticationMethod": "Key" を取得します。

- [get-platform-application-attributes](#)

既存の Amazon SNS トピックの FCM 属性を設定するには (AWS CLI)

FCM 属性を設定します。

- [set-platform-application-attributes](#)

FCM の設定の管理 (コンソール)

Amazon SNS で Firebase Cloud Messaging (FCM) 認証情報を入力および管理するには、次の手順を実行します。

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションペインで、[プッシュ通知] を選択します。
3. [プラットフォームアプリケーション] セクションで、認証情報を編集する FCM プラットフォームアプリケーションを選択し、[編集]を選択します。
4. [Firebase Cloud Messaging の認証情報] セクションで、次のいずれかのオプションを選択します。
 - トークンベースの認証 (推奨方法) – Firebase コンソールからダウンロードしたプライベートキーファイル (JSON) をアップロードします。このファイルには、FCM 通知の有効期間の短いアクセストークンを生成するために必要な認証情報が含まれています。このファイルを取得するには、次を実行します。
 1. [Firebase アプリケーションコンソール](#)に移動します。
 2. [プロジェクト設定]で、[Cloud Messaging] を選択します。
 3. プライベートキー JSON ファイルをダウンロードします (トークンベースの認証方法で利用)。
 - API キー認証 – 古い API キー認証方法を使用する場合は、指定されたフィールドに Google API キーを入力します。このファイルを取得するには、次を実行します。

1. [Firebase アプリケーションコンソール](#)に移動します。
 2. [プロジェクト設定]で、[Cloud Messaging] を選択します。
 3. 通知の送信に使用するサーバーキー (API キー) をコピーします。
5. 変更が完了したら、[変更の保存] を選択します。

関連トピック

- [Amazon SNS での Google Firebase Cloud Messaging v1 ペイロードの使用](#)

Amazon SNS の Firebase Cloud Messaging エンドポイントの管理

デバイストークンの管理と維持

以下の手順に従って、モバイルアプリケーションのプッシュ通知の配信可能性を確保できます。

1. すべてのデバイストークン、対応する Amazon SNS エンドポイントの ARN、タイムスタンプを、アプリケーションサーバーに保存します。
2. すべての古いトークンを削除し、対応する Amazon SNS エンドポイント ARN を削除します。

アプリの初回起動時に、デバイスのデバイストークン (登録トークンとも呼ばれます) を受け取ります。このデバイストークンは、デバイスのオペレーティングシステムによって作成され、FCM アプリケーションに関連付けられています。このデバイストークンを受け取ったら、プラットフォームエンドポイントとして Amazon SNS に登録できます。デバイストークン、Amazon SNS プラットフォームエンドポイント ARN、タイムスタンプを、アプリケーションサーバーまたは別の永続ストアに保存することをお勧めします。FCM アプリケーションをセットアップし、デバイストークンを取得して保存するには、Google の Firebase ドキュメントの[登録トークンの取得と保存](#)を参照してください。

最新のトークンを維持することが重要です。ユーザーのデバイストークンは、次の場合に変更される可能性があります。

1. モバイルアプリケーションが、新しいデバイスで復元されます。
2. ユーザーにより、アプリケーションがアンインストールまたは更新されます。
3. ユーザーが、アプリケーションデータを消去します。

デバイストークンが変更された場合、対応する Amazon SNS エンドポイントを新しいトークンで更新することをお勧めします。これにより、Amazon SNS は登録デバイスとの通信を継続できます。これを行うには、モバイルアプリケーションに次の擬似コードを実装します。有効なプラットフォームエンドポイントを作成および維持するための推奨プラクティスについて説明します。このアプローチは、モバイルアプリケーションが開始されるたびに実行することも、バックグラウンドでスケジュールされたジョブとして実行することもできます。

擬似コード

次の FCM 擬似コードを使用して、デバイストークンを管理および維持します。

```
retrieve the latest token from the mobile OS
if (endpoint arn not stored)
    # first time registration
    call CreatePlatformEndpoint
    store returned endpoint arn
endif

call GetEndpointAttributes on the endpoint arn

if (getting attributes encountered NotFound exception)
    #endpoint was deleted
    call CreatePlatformEndpoint
    store returned endpoint arn
else
    if (token in endpoint does not match latest) or
        (GetEndpointAttributes shows endpoint as disabled)
        call SetEndpointAttributes to set the
            latest token and enable the endpoint
    endif
endif
```

トークンの更新要件の詳細については、Google の Firebase ドキュメントの「[定期的なトークンの更新](#)」を参照してください。

無効なトークンの検出

無効なデバイストークンを使用してメッセージが FCM v1 エンドポイントに配信されると、Amazon SNS は次の例外のいずれかを受け取ります。

- UNREGISTERED (HTTP 404) – Amazon SNS がこの例外を受信すると、InvalidPlatformToken の FailureType を含む配信失敗イベント、および「工

エンドポイントに関連付けられたプラットフォームトークンが有効ではありません」という `FailureMessage` を受け取ります。Amazon SNS は、この例外で配信が失敗すると、プラットフォームエンドポイントを無効にします。

- `INVALID_ARGUMENT` (HTTP 400) – Amazon SNS がこの例外を受信すると、デバイストークンまたはメッセージペイロードが無効であることを意味します。詳細については、Google の Firebase ドキュメントの「[ErrorCode](#)」を参照してください。

`INVALID_ARGUMENT` はこれらのいずれの場合でも返される可能性があるため、Amazon SNS は `InvalidNotification` の `FailureType`、および「通知本文が無効です」という `FailureMessage` を返します。このエラーが表示されたら、ペイロードが正しいことを確認します。ペイロードが正しい場合は、デバイストークンが最新であることを確認します。この例外で配信が失敗しても、Amazon SNS はプラットフォームエンドポイントを無効にしません。

`InvalidPlatformToken` 配信失敗イベントが発生するもう 1 つのケースは、登録されたデバイストークンが、そのメッセージを送信しようとするアプリケーションにない場合です。この場合、Google は `SENDER_ID_MISMATCH` エラーを返します。Amazon SNS は、この例外で配信が失敗すると、プラットフォームエンドポイントを無効にします。

FCM v1 API から受信したすべての観測されたエラーコードは、アプリケーションの[配信ステータス ログ](#)を設定するときに、CloudWatch で利用できます。

アプリケーションの配信イベントを受け取るには、「[使用可能なアプリケーションイベント](#)」を参照してください。

古いトークンの削除

エンドポイントデバイスへのメッセージ配信が失敗すると、トークンは古いとみなされます。Amazon SNS は、これらの古いトークンをプラットフォームアプリケーションの無効なエンドポイントとして設定します。無効になったエンドポイントに発行すると、Amazon SNS は `EventDeliveryFailure` イベントと `EndpointDisabled` の `FailureType`、および「エンドポイントが無効です」という `FailureMessage` を返します。アプリケーションの配信イベントを受け取るには、「[使用可能なアプリケーションイベント](#)」を参照してください。

Amazon SNS からこのエラーを受け取った場合、プラットフォームアプリケーションで古いトークンを削除または更新する必要があります。

Amazon SNS を使用したモバイルプッシュ通知

このセクションでは、モバイルプッシュ通知を送信する方法について説明します。

トピックへ発行する

Amazon SNS を使用して、トピックにサブスクリプションしたモバイルエンドポイントにメッセージを送信することもできます。このコンセプトは、「[Amazon SNS とは](#)」に説明されているように、Amazon SQS、HTTP/S、E メール、SMS などその他のエンドポイントタイプをトピックにサブスクライブするのと同じです。違いは、Amazon SNS は Apple Push Notification Service (APNS) や Google Firebase Cloud Messaging (FCM) などの通知サービスを通じて通信することです。通知サービスを通じて、サブスクライブしたモバイルエンドポイントがトピックに送信された通知を受信します。

Amazon SNS モバイルデバイスの直接メッセージング

モバイルデバイスのアプリケーションを表すエンドポイントに Amazon SNS プッシュ通知メッセージを直接送信できます。

直接メッセージを送信するには

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[Push notifications] (プッシュ通知) を選択します。
3. [Mobile push notifications] (モバイルプッシュ通知) ページの [Platform applications] (プラットフォームアプリケーション) セクションで、アプリケーションの名前 (**MyApp** など) を選択します。
4. **MyApp** ページの [エンドポイント] セクションで、エンドポイントを選択し、[メッセージの発行] を選択します。
5. [エンドポイントへのメッセージの発行] ページで、モバイルデバイスのアプリケーションに表示されるメッセージを入力し、[メッセージの発行] を選択します。

Amazon SNS によって、プラットフォーム通知サービスに通知メッセージが送信され、プラットフォーム通知サービスはアプリケーションにメッセージを送信します。

プラットフォーム固有のペイロードを使用した Amazon SNS 通知の発行

AWS マネジメントコンソール または Amazon SNS API を使用して、プラットフォーム固有のペイロードを含むカスタムメッセージをモバイルデバイスに送信できます。APIs Amazon SNS API の使用については、[モバイルプッシュ API アクション](#) および [snsmobilepush.zip](#) の SNSMobilePush.java ファイルを参照してください。

JSON 形式のメッセージの送信

プラットフォーム固有のペイロードを送信する場合、データは引用符をエスケープした、JSON キーと値のペア文字列である必要があります。

次の例では、FCM プラットフォーム用のカスタムメッセージを示します。

```
{
  "GCM": "{\"fcmV1Message\": {\"message\": {\"notification\": {\"title\": \"Hello\",
    \"body\": \"This is a test.\"}, \"data\": {\"dataKey\": \"example\"}}}}"
```

プラットフォーム固有のメッセージを送信する

カスタムデータをキーと値のペアとして送信することに加えて、プラットフォーム固有のキーと値のペアを送信できます。

次の例では、FCM data パラメータのカスタムデータのキーと値のペアの後に、FCM パラメータ `time_to_live` および `collapse_key` を含めています。

```
{
  "GCM": "{\"fcmV1Message\": {\"message\": {\"notification\": {\"title\": \"TitleTest\",
    \"body\": \"Sample message for Android or iOS endpoints.\"}, \"data\": {\"time_to_live
    \": 3600, \"collapse_key\": \"deals\"}}}}"
```

Amazon SNS でサポートされている各プッシュ通知サービスでサポートされているキーと値のペアのリストについては次を参照してください。

Important

Amazon SNS は、Android デバイスにモバイルプッシュ通知を送信するために、Firebase Cloud Messaging (FCM) HTTP v1 API をサポートするようになりました。

2024 年 3 月 26 日、Amazon SNS は、Apple デバイスと Webpush 送信先用の FCM HTTP v1 API をサポートします。アプリケーションの中断を避けるため、2024 年 6 月 1 日またはそれ以前に、既存のモバイルプッシュアプリケーションを最新の FCM HTTP v1 API に移行することをお勧めします。

- APN ドキュメントの [ペイロードキーのリファレンス](#)

- FCM ドキュメントの[Firebase Cloud Messaging HTTP プロトコル](#)
- ADM ドキュメントの[メッセージの送信](#)

複数プラットフォーム上のアプリケーションへのメッセージ送信

FCM や APN など、複数プラットフォーム向けのデバイスにインストールされたアプリケーションにメッセージを送信するには、Amazon SNS のトピックにモバイルエンドポイントをサブスクリプションしてから、トピックにメッセージを発行します。

次の例は、APN、FCM、および ADM でサブスクリプションしたモバイルエンドポイントに送信するメッセージを示しています。

```
{
  "default": "This is the default message which must be present when publishing a
message to a topic. The default message will only be used if a message is not present
for
one of the notification platforms.",
  "APNS": "{\"aps\":{\"alert\": \"Check out these awesome deals!\",\"url\":
\\\"www.amazon.com\\\"} }",
  "GCM": "{\"data\":{\"message\": \"Check out these awesome deals!\",\"url\":
\\\"www.amazon.com\\\"}}",
  "ADM": "{\"data\":{\"message\": \"Check out these awesome deals!\",\"url\":
\\\"www.amazon.com\\\"}}"
}
```

アラートまたはバックグラウンド通知としてのメッセージを APN に送信する

Amazon SNS は APN にメッセージを alert または background 通知 (詳細については、APN のドキュメントの「[バックグラウンド更新をアプリケーションにプッシュする](#)」を参照してください)。

- alert APN 通知では、ユーザーに通知する方法として、アラートメッセージを表示するか、サウンドを鳴らすか、アプリケーションのアイコンにバッジを追加します。
- background APN 通知は、ユーザーに通知することなく、ウェイクアップするか、通知の内容に基づいて動作するようにアプリケーションに指示します。

APN ヘッダーのカスタム値を指定する

Amazon SNS Publish API アクション、AWS SDKs、または `awscli` を使用して、`AWS.SNS.MOBILE.APNS.PUSH_TYPE` [予約済みメッセージ属性](#)のカスタム値を指定することを

お勧めします AWS CLI。次の CLI の例では、指定したトピックの `content-available` を 1 に設定し、`apns-push-type` を `background` に設定します。

```
aws sns publish \
--endpoint-url https://sns.us-east-1.amazonaws.com \
--target-arn arn:aws:sns:us-east-1:123456789012:endpoint/APNS_PLATFORM/MYAPP/1234a567-bc89-012d-3e45-6fg7h890123i \
--message '{"APNS_PLATFORM":{"aps":{"content-available":1}}}' \
--message-attributes '{ \
  "AWS.SNS.MOBILE.APNS.TOPIC": \
{"DataType":"String","StringValue":"com.amazon.mobile.messaging.myapp"}, \
  "AWS.SNS.MOBILE.APNS.PUSH_TYPE":{"DataType":"String","StringValue":"background"}, \
  "AWS.SNS.MOBILE.APNS.PRIORITY":{"DataType":"String","StringValue":"5"}}' \
--message-structure json
```

Note

JSON 構造が有効であることを確認します。キーと値の各ペアの後 (最後のペア以外) にカンマを追加します。

ペイロードからの APN プッシュタイプヘッダーを推測する

`apns-push-type` APN ヘッダーを設定しないと、JSON 形式の APN ペイロードに設定されている `aps` デイクシヨナリ内の `content-available` キーに応じて、Amazon SNS がヘッダーを `alert` または `background` に設定します。

Note

`apns-push-type` ヘッダーは他の値に設定できますが、Amazon SNS が推測できるのは `alert` ヘッダーまたは `background` ヘッダーのみです。

- `apns-push-type` は、`alert` に設定されます。
- `aps` デイクシヨナリで `content-available` が 1 に設定されていて、ユーザーの操作をトリガーする 1 つ以上のキーが含まれている場合。
- `aps` デイクシヨナリで `content-available` が 0 に設定されているか、または `content-available` キーが存在しない場合。
- `content-available` キーの値が整数またはブール値でない場合。

- apns-push-type は、background に設定されます。
- aps ディクショナリに 1 に設定された content-available のみが含まれ、ユーザーの操作をトリガーする他のキーが含まれていない場合。

Important

Amazon SNSが APN の raw 設定オブジェクトをバックグラウンドのみの通知として送信する場合は、content-available を 1 に設定して aps ディクショナリに含める必要があります。カスタムキーを含めることもできますが、ユーザー操作をトリガーするキー (アラート、バッジ、サウンドなど) を aps ディクショナリに含めることはできません。

raw 設定オブジェクトの例を次に示します。

```
{
  "APNS": "{\"aps\":{\"content-available\":1},\"Foo1\":{\"Bar\"},\"Foo2\":123}"
}
```

この例の場合、Amazon SNS はメッセージの apns-push-type APN ヘッダーを background に設定します。Amazon SNS が apn ディクショナリが 1 に設定された content-available キーを含み、ユーザー操作をトリガーする他のキーを含まないことを検出すると、Amazon SNS はヘッダーを background に設定します。

Amazon SNS での Google Firebase Cloud Messaging v1 ペイロードの使用

Amazon SNS で、FCM HTTP v1 API を使用して、Android、iOS、および Webpush の送信先に通知を送信できます。このトピックでは、CLI または Amazon SNS API を使用してモバイルプッシュ通知を発行する際のペイロード構造の例を示します。

FCM 通知を送信するときに、ペイロードに次のメッセージタイプを含めることができます。

- データメッセージ – データメッセージはクライアントアプリによって処理され、カスタムのキーと値のペアが含まれます。データメッセージを作成するときは、JSON オブジェクトの data キーを値として含め、カスタムのキーと値のペアを入力する必要があります。
- 通知メッセージまたは表示メッセージ – 通知メッセージには、FCM SDK によって処理される事前定義されたキーのセットが含まれます。これらのキーは、配信先のデバイスタイプによって異なります。プラットフォーム固有の通知キーの詳細については、以下を参照してください。

- [Android 通知キー](#)
- [APNS 通知キー](#)
- [Webpush 通知キー](#)

FCM メッセージタイプの詳細については、Google の「Firebase」ドキュメントの「[メッセージタイプ](#)」を参照してください。

FCM v1 ペイロード構造を使用したメッセージの送信

初めて FCM アプリケーションを作成する場合、または FCM v1 機能を活用する場合は、FCM v1 形式のペイロードの送信をオプトインできます。これを行うには、最上位キー `fcmV1Message` を含める必要があります。FCM v1 ペイロードの構築の詳細については、Google の「Firebase」ドキュメントの「[レガシー FCM API から HTTP v1 への移行](#)」および「[プラットフォーム間でのメッセージのカスタマイズ](#)」を参照してください。

Amazon SNS に送信された FCM v1 ペイロードの例:

Note

次の例で使用されている GCM キー値は、Amazon SNS を使用して通知を発行するときに文字列としてエンコードする必要があります。

```
{
  "GCM": "{
    \"fcmV1Message\": {
      \"validate_only\": false,
      \"message\": {
        \"notification\": {
          \"title\": \"string\",
          \"body\": \"string\"
        },
        \"data\": {
          \"dataGen\": \"priority message\"
        },
        \"android\": {
          \"priority\": \"high\",
          \"notification\": {
            \"body_loc_args\": [\"string\"],
            \"title_loc_args\": [\"string\"],
```

```
\sound\": \"string\",
\"title_loc_key\": \"string\",
\"title\": \"string\",
\"body\": \"string\",
\"click_action\": \"clicky_clacky\",
\"body_loc_key\": \"string\"
},
\"data\": {
  \"dataAndroid\": \"priority message\"
},
\"ttl\": \"10023.32s\"
},
\"apns\": {
  \"payload\": {
    \"aps\": {
      \"alert\": {
        \"subtitle\": \"string\",
        \"title-loc-args\": [\"string\"],
        \"title-loc-key\": \"string\",
        \"loc-args\": [\"string\"],
        \"loc-key\": \"string\",
        \"title\": \"string\",
        \"body\": \"string\"
      },
      \"category\": \"Click\",
      \"content-available\": 0,
      \"sound\": \"string\",
      \"badge\": 5
    }
  }
},
\"webpush\": {
  \"notification\": {
    \"badge\": \"5\",
    \"title\": \"string\",
    \"body\": \"string\"
  },
  \"data\": {
    \"dataWeb\": \"priority message\"
  }
}
}
```

```
}
```

JSON ペイロードを送信するときは、必ずリクエストに `message-structure` 属性を含めて `json` に設定します。

CLI の例:

```
aws sns publish --topic $TOPIC_ARN --message '{"GCM": {"\fcmV1Message\": {"\message\": {"\notification\": {"\title\": "\string\","body\": "\string\"},"android\": {"\priority\": "\high\","notification\": {"\title\": "\string\","body\": "\string\"},"data\": {"\customAndroidDataKey\": "\custom key value\"},"ttl\": "\0s\"},"apns\": {"\payload\": {"\aps\": {"\alert\": {"\title\": "\string\","body\": "\string\"},"content-available\": 1,\badge\": 5}}}, "\webpush\": {"\notification\": {"\badge\": "\URL\","body\": "\Test\"},"data\": {"\customWebpushDataKey\": "\priority message\"},"data\": {"\customGeneralDataKey\": "\priority message\}}}}', "default": {"\notification\": {"\title\": "\test\}}}' --region $REGION --message-structure json
```

FCM v1 形式のペイロードの送信の詳細については、Google の「[Firebase](#)」ドキュメントの以下を参照してください。

- [レガシー FCM API から HTTP v1 への移行](#)
- [FCM メッセージについて](#)
- [REST リソース: `projects.messages`](#)

レガシーペイロード構造を使用した FCM v1 API へのメッセージ送信

FCM v1 に移行する場合、レガシー認証情報に使用していたペイロード構造を変更する必要はありません。Amazon SNS はペイロードを新しい FCM v1 ペイロード構造に変換し、Google に送信します。

入力メッセージのペイロード形式:

```
{
  "GCM": {"\notification\": {"\title\": "\string\","body\": "\string\","
    "\android_channel_id\": "\string\","body_loc_args\": ["\string\"],"body_loc_key\": "\string\","
    "\click_action\": "\string\","color\": "\string\","icon\": "\string\","
    "\sound\": "\string\","tag\": "\string\","title_loc_args\": ["\string\"],"
    "\title_loc_key\": "\string\"},"data\": {"\message\": "\priority message\}}"}
}
```

Google に送信されたメッセージ:

```
{
  "message": {
    "token": "****",
    "notification": {
      "title": "string",
      "body": "string"
    },
    "android": {
      "priority": "high",
      "notification": {
        "body_loc_args": [
          "string"
        ],
        "title_loc_args": [
          "string"
        ],
        "color": "string",
        "sound": "string",
        "icon": "string",
        "tag": "string",
        "title_loc_key": "string",
        "title": "string",
        "body": "string",
        "click_action": "string",
        "channel_id": "string",
        "body_loc_key": "string"
      },
      "data": {
        "message": "priority message"
      }
    },
    "apns": {
      "payload": {
        "aps": {
          "alert": {
            "title-loc-args": [
              "string"
            ],
            "title-loc-key": "string",
            "loc-args": [
              "string"
            ],

```

```
        "loc-key": "string",
        "title": "string",
        "body": "string"
    },
    "category": "string",
    "sound": "string"
}
},
"webpush": {
    "notification": {
        "icon": "string",
        "tag": "string",
        "body": "string",
        "title": "string"
    },
    "data": {
        "message": "priority message"
    }
},
"data": {
    "message": "priority message"
}
}
}
```

潜在的なリスク

- レガシーから v1 へのマッピングでは、Apple Push Notification Service (APNS) headers または fcm_options キーをサポートしていません。これらのフィールドを使用する場合は、FCM v1 ペイロードを送信します。
- 場合によっては、APN デバイスにサイレント通知を送信するには、FCM v1 でメッセージヘッダーが必要です。現在 APN デバイスにサイレント通知を送信している場合、レガシーアプローチでは動作しません。予期しない問題を避けるため、FCM v1 ペイロードを使用することをお勧めします。APN ヘッダーのリストとそれらの使用目的を確認するには、「[Apple デベロッパーガイド](#)」の「[APN との通信](#)」を参照してください。
- 通知の送信時に TTL Amazon SNS 属性を使用している場合は、android フィールドでのみ更新されます。TTL APNS 属性を設定する場合は、FCM v1 ペイロードを使用します。
- android、apns、および webpush キーはマッピングされ、提供されたすべての関連キーが入力されます。例えば、title は 3 つのプラットフォームすべてで共有されているキーであり、これ

を指定すると、FCM v1 マッピングによって、3 つのプラットフォームすべてに指定したタイトルが入力されます。

- プラットフォーム間で共有される一部のキーでは、値のタイプが異なります。例えば、apns に渡される badge キーは整数値、webpush に渡される badge キーは文字列値である必要があります。badge キーが指定された場合、FCM v1 マッピングは、有効な値が指定されたキーのみを入力します。

FCM 配信失敗イベント

次の表に、Amazon SNS 障害タイプを示します。これは、FCM v1 通知リクエストで Google から受信したエラー/ステータスコードに対応しています。FCM v1 API から受信したすべての観測されたエラーコードは、アプリケーションの[配信ステータスログ](#)を設定するときに、CloudWatch で利用できます。

FCM エラー/ステータスコード	Amazon SNS 障害タイプ	障害メッセージ	原因と緩和策
UNREGISTERED	InvalidPlatformToken	エンドポイントに関連付けられたプラットフォームトークンが無効です。	エンドポイントにアタッチされたデバイストークンが古くなったか、無効です。Amazon SNS はエンドポイントを無効にしました。Amazon SNS エンドポイントを最新のデバイストークンに更新してください。
INVALID_ARGUMENT	InvalidNotification	通知本文が無効です。	デバイストークンまたはメッセージペイロードが無効である可能性があります。メッセージペイロードが有効であることを確認してください。メッセージペイロ

FCM エラー/ステータスコード	Amazon SNS 障害タイプ	障害メッセージ	原因と緩和策
			ードが有効な場合は、Amazon SNS エンドポイントを最新のデバイストークンに更新してください。
SENDER_ID_MISMATCH	InvalidPlatformToken	エンドポイントに関連付けられたプラットフォームトークンが無効です。	デバイストークンに関連付けられたプラットフォームアプリケーションには、デバイストークンに送信するアクセス許可がありません。Amazon SNS プラットフォームアプリケーションで正しい FCM 認証情報を使用していることを確認してください。
UNAVAILABLE	DependencyUnavailable	依存関係は使用できません。	FCM はリクエストを時間内に処理できませんでした。Amazon SNS で実行されたすべての再試行が失敗しました。これらのメッセージをデッドレターキュー (DLQ) に保存して、後で再処理できます。

FCM エラー/ステータスコード	Amazon SNS 障害タイプ	障害メッセージ	原因と緩和策
INTERNAL	UnexpectedFailure	予期しない失敗です。Amazon にお問い合わせください。失敗フレーズ [内部エラー]。	リクエストの処理中に FCM サーバーでエラーが発生しました。Amazon SNS で実行されたすべての再試行が失敗しました。これらのメッセージをデッドレターキュー (DLQ) に保存して、後で再処理できます。
THIRD_PARTY_AUTH_ERROR	InvalidCredentials	プラットフォームアプリケーションの認証情報が無効です。	iOS デバイスまたは Webpush デバイスをターゲットにしたメッセージを送信できませんでした。開発用と本番用の認証情報が有効であることを確認してください。
QUOTA_EXCEEDED	Throttled	リクエストは [gcm] によってスロットリングされました。	メッセージレート、デバイスメッセージレート、またはトピックメッセージレートがクォータを超過しています。この問題を解決する方法については、Google の「Firebase」ドキュメントの「 ErrorCode 」を参照してください。

FCM エラー/ステータスコード	Amazon SNS 障害タイプ	障害メッセージ	原因と緩和策
PERMISSION_DENIED	InvalidNotification	通知本文が無効です。	PERMISSION_DENIED 例外の場合、発信者 (FCM アプリケーション) にペイロードで指定されたオペレーションを実行するアクセス許可がありません。FCM コンソールに移動し、認証情報で必要な API アクションが有効になっていることを確認してください。

Amazon SNS モバイルアプリの属性

Amazon Simple Notification Service (Amazon SNS) では、プッシュ通知メッセージの配信ステータスの記録がサポートされています。アプリケーション属性を設定した後、Amazon SNS からモバイルエンドポイントに送信されたメッセージのログエントリが CloudWatch Logs に送信されます。メッセージの配信ステータスを記録することは、以下のように運用をよりよく把握するのに役立ちます。

- プッシュ通知メッセージが Amazon SNS からプッシュ通知サービスに配信されたかがわかります。
- プッシュ通知サービスから Amazon SNS に送信されたレスポンスを特定します。
- メッセージのドウェル時間 (発行のタイムスタンプからプッシュ通知サービスへの配信直前までの時間) を決定します。

メッセージ配信ステータスのアプリケーション属性を設定するには、AWS ソフトウェア開発キット (SDKs) AWS マネジメントコンソール、またはクエリ API を使用できます。

を使用したメッセージ配信ステータス属性の設定 AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで [モバイル]、[プッシュ通知] を選択します。
3. [プラットフォームアプリケーション]セクションで、CloudWatch Logs を受け取るエンドポイントを含むアプリケーションを選択します。
4. [アプリケーションアクション]、[配信ステータス] の順に選択します。
5. [配信ステータス] ダイアログボックスで、[IAM ロールの作成] を選択します。

IAM コンソールにリダイレクトされます。

6. [許可] を選択して、お客様に代わって CloudWatch Logs を使用する書き込みアクセス許可を Amazon SNS に付与します。
7. ここで、[配信ステータス] ダイアログボックスに戻り、[Percentage of Success to Sample (0-100)] フィールドに、CloudWatch Logs を受信するために送信される正常なメッセージの割合を数字で入力します。

Note

メッセージの配信ステータスのアプリケーション属性を設定した後は、メッセージの配信に失敗すると、必ず CloudWatch Logs が生成されます。

8. 最後に、[設定の保存] を選択します。これで、メッセージの配信ステータスを含む CloudWatch Logs を参照して解析できます。CloudWatch の使用方法の詳細については、「[CloudWatch のドキュメント](#)」を参照してください。

Amazon SNS メッセージ配信ステータス CloudWatch ログの例

アプリケーションエンドポイントのメッセージの配信ステータスの属性を設定した後は、CloudWatch Logs が生成されます。以下は、JSON 形式のログの例です。

SUCCESS

```
{
  "status": "SUCCESS",
  "notification": {
    "timestamp": "2015-01-26 23:07:39.54",
    "messageId": "9655abe4-6ed6-5734-89f7-e6a6a42de02a"
  },
}
```

```

"delivery": {
  "statusCode": 200,
  "dwellTimeMs": 65,
  "token": "Examplei7fFachkJ1xjlqT64RaBkcGHochmf1VQAr9k-
IBJtKjp7fedYPzEwT_Pq3Tu0lroqro1cwWJUvgkcPPYcaXCpPWmG3Bqn-
wiqIEzp5zZ7y_jsM0PKPxKhddCzx6paEsyay9Zn3D4wNUJb8m6HXrBf9dqaEw",
  "attempts": 1,
  "providerResponse": "{\"multicast_id\":5138139752481671853,\"success
\":1,\"failure\":0,\"canonical_ids\":0,\"results\": [{\"message_id\":
\"0:1422313659698010%d6ba8edff9fd7ecd\"}]}",
  "destination": "arn:aws:sns:us-east-2:111122223333:endpoint/FCM/FCMPushApp/
c23e42de-3699-3639-84dd-65f84474629d"
}
}

```

FAILURE

```

{
  "status": "FAILURE",
  "notification": {
    "timestamp": "2015-01-26 23:29:35.678",
    "messageId": "c3ad79b0-8996-550a-8bfa-24f05989898f"
  },
  "delivery": {
    "statusCode": 8,
    "dwellTimeMs": 1451,
    "token": "examp1e29z6j5c4df46f80189c4c83fjcgf7f6257e98542d2j3395kj73",
    "attempts": 1,
    "providerResponse": "NotificationErrorResponse(command=8, status=InvalidToken,
id=1, cause=null)",
    "destination": "arn:aws:sns:us-east-2:111122223333:endpoint/APNS_SANDBOX/
APNSPushApp/986cb8a1-4f6b-34b1-9a1b-d9e9cb553944"
  }
}

```

プッシュ通知サービスのレスポンスコードの一覧は、「[プラットフォームのレスポンスコード](#)」を参照してください。

AWS SDKs を使用したメッセージ配信ステータス属性の設定

[AWS SDK](#) には、Amazon SNS でメッセージの配信ステータスの属性を使用するための API がいくつかの言語で用意されています。

次の Java の例は、SetPlatformApplicationAttributes API を使用して、プッシュ通知メッセージの配信ステータスのアプリケーション属性を設定する方法を示しています。メッセージの配信ステータスには、SuccessFeedbackRoleArn、FailureFeedbackRoleArn、および SuccessFeedbackSampleRate の属性を使用できます。SuccessFeedbackRoleArn および FailureFeedbackRoleArn 属性は、お客様に代わって CloudWatch Logs を使用する書き込みアクセス許可を Amazon SNS に付与するために使用します。SuccessFeedbackSampleRate 属性は、正常な配信メッセージのサンプルレートの割合 (0〜100) を指定するためのものです。FailureFeedbackRoleArn 属性を設定した後は、メッセージの配信に失敗すると、必ず CloudWatch Logs が生成されます。

```
SetPlatformApplicationAttributesRequest setPlatformApplicationAttributesRequest = new
    SetPlatformApplicationAttributesRequest();
Map<String, String> attributes = new HashMap<>();
attributes.put("SuccessFeedbackRoleArn", "arn:aws:iam::111122223333:role/SNS_CWlogs");
attributes.put("FailureFeedbackRoleArn", "arn:aws:iam::111122223333:role/SNS_CWlogs");
attributes.put("SuccessFeedbackSampleRate", "5");
setPlatformApplicationAttributesRequest.withAttributes(attributes);
setPlatformApplicationAttributesRequest.setPlatformApplicationArn("arn:aws:sns:us-
west-2:111122223333:app/FCM/FCMPushApp");
sns.setPlatformApplicationAttributes(setPlatformApplicationAttributesRequest);
```

Java 用 SDL の詳細については、「[AWS SDK for Javaの開始方法](#)」を参照してください。

プラットフォームのレスポンスコード

以下は、プッシュ通知サービスのレスポンスコードへのリンクの一覧です。

プッシュ通知サービス	レスポンスコード
Amazon Device Messaging (ADM)	ADM のドキュメントの「 レスポンスの形式 」を参照してください。
Apple Push Notification Service (APNS)	『ローカルおよびリモート通知プログラミングガイド』の「 APN との通信 におけるAPN からの HTTP/2 レスポンス」を参照してください。
Firebase Cloud Messaging (FCM)	Firebase Cloud Messaging ドキュメントの「 Downstream Message Error Response Codes 」を参照してください。

プッシュ通知サービス	レスポンスコード
Windows Phone (MPNS) 用 Microsoft Push Notification Service (MPNS)	Windows 8 開発ドキュメントの「 Push Notification Service Response Codes for Windows Phone 8 」を参照してください。
Windows Push Notification Services (WNS)	Windows 8 開発ドキュメントで「 Push Notification Service Request and Response Headers (Windows Runtime Apps) 」の「Response codes」を参照してください。

モバイルアプリケーションの Amazon SNS アプリケーションイベント通知

Amazon SNS は、特定のアプリケーションイベントが発生する際のトリガー通知をサポートします。その場合、そのイベントでプログラムによるいくつかのアクションを実行することができます。アプリケーションには、Apple Push Notification Service (APN)、Firebase Cloud Messaging (FCM)、Windows Push Notification Service (WNS) などのプッシュ通知サービスのサポートが含まれている必要があります。Amazon SNS コンソール AWS CLI、または AWS SDKs を使用してアプリケーションイベント通知を設定します。

使用可能なアプリケーションイベント

アプリケーションイベント通知は、個々のプラットフォームエンドポイントがいつ作成、削除、更新、配信エラーとなったかを追跡します。アプリケーションイベントの属性名は次のとおりです。

属性名	通知トリガー
EventEndpointCreated	新しいエンドポイントがアプリケーションに追加されます。
EventEndpointDeleted	アプリケーションに関連付けられたすべてのプラットフォームエンドポイントが削除されます。
EventEndpointUpdated	アプリケーションに関連付けられたプラットフォームエンドポイントのすべての属性が変更されます。
EventDeliveryFailure	アプリケーションに関連付けられた任意のプラットフォームエンドポイントへの配信は、永続的にエラーとなります。

属性名	通知トリガー
	 Note プラットフォームアプリケーション側で配信エラーを追跡するには、アプリケーションのメッセージ配信ステータスイベントにサブスクライブします。詳細については、「メッセージの配信ステータスの Amazon SNS アプリケーション属性を使用する」を参照してください。

アプリケーションには任意の属性を関連付けることができ、これでそのイベント通知を受け取れるようになります。

モバイルプッシュ通知を送信する

アプリケーションイベント通知を送信するには、各タイプのイベントの通知を受信できるよう、トピックを指定します。Amazon SNS が通知を送信する場合、トピックは、プログラムによるアクションを取るエンドポイントにそれらをルーティングすることができます。

Important

大容量アプリケーションは多数のアプリケーションイベント通知 (例えば、数万回) を作成するので、E メールアドレス、電話番号、モバイルアプリケーションなど、人間が使用するためのエンドポイントに負荷をかけます。アプリケーションイベント通知をトピックに送信する場合は、以下のガイドラインを検討してください。

- 通知を受け取る各トピックには、HTTP または HTTPS エンドポイント、Amazon SQS キュー、AWS Lambda 関数などのプログラムによるエンドポイントのサブスクリプションのみを含める必要があります。
- 通知によってトリガーされる処理量を減らすために、各トピックのサブスクリプションを少数 (例えば、5 以下) に制限します。

Amazon SNS コンソール、(AWS CLI)、AWS Command Line Interface または AWS SDKs を使用して、アプリケーションイベント通知を送信できます。

AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで [モバイル]、[プッシュ通知] を選択します。
3. [モバイルプッシュ通知] ページの [プラットフォームアプリケーション] セクションでアプリケーションを選択したら、[編集] を選択します。
4. [イベント通知] セクションを展開します。
5. [アクション]、[イベントの設定] を選択します。
6. 次のイベントで使用される ARN を入力します。
 - 作成されたエンドポイント
 - 削除されたエンドポイント
 - 更新されたエンドポイント
 - 配信失敗
7. [Save changes] (変更の保存) をクリックします。

AWS CLI

[set-platform-application-attributes](#) コマンドを実行します。

次の例では、4 つのアプリケーションすべてに対して同じ Amazon SNS トピックを設定します。

```
aws sns set-platform-application-attributes
--platform-application-arn arn:aws:sns:us-east-1:12345EXAMPLE:app/FCM/
MyFCMPlatformApplication
--attributes EventEndpointCreated="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventEndpointDeleted="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventEndpointUpdated="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents",
EventDeliveryFailure="arn:aws:sns:us-
east-1:12345EXAMPLE:MyFCMPlatformApplicationEvents"
```

AWS SDKs

AWS SDK を使用して Amazon SNS API で `SetPlatformApplicationAttributes` リクエストを送信して、アプリケーションイベント通知を設定します。

開始方法や以前のバージョンに関する情報など、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 での Amazon SNS の使用](#)。

モバイルプッシュ API アクション

Amazon SNS モバイルプッシュ API を使用するには、まず、プッシュ通知サービス (Apple Push Notification (APN)、Firebase Cloud Messaging (FCM) など) の前提条件を満たす必要があります。前提条件の詳細については、「[Amazon SNS ユーザー通知の前提条件](#)」を参照してください。

API を使用してモバイルアプリケーションおよびデバイスにプッシュ通知メッセージを送信するには、まず、`CreatePlatformApplication` アクションを使用する必要があります。このアクションは、`PlatformApplicationArn` 属性を返します。この `PlatformApplicationArn` 属性は、`CreatePlatformEndpoint` によって使用され、`EndpointArn` 属性が返されます。この `EndpointArn` 属性と `Publish` アクションを使用して、モバイルアプリケーションやデバイスに通知メッセージを送信することができます。また、`EndpointArn` 属性と `Subscribe` アクションを使って、トピックへのサブスクリプションを行うこともできます。詳細については、「[Amazon SNS でプッシュ通知を設定する](#)」を参照してください。

Amazon SNS モバイルプッシュ API は次のとおりです。

[CreatePlatformApplication](#)

デバイスやモバイルアプリケーションを登録できる、サポートされているいずれかのプッシュ通知サービス (例: APNS、GCM) のプラットフォームアプリケーションオブジェクトを作成します。`PlatformApplicationArn` 属性を返します。この属性は、`CreatePlatformEndpoint` アクションで使用されます。

[CreatePlatformEndpoint](#)

サポートされているプッシュ通知サービスのいずれかでデバイスおよびモバイルアプリケーションのエンドポイントを作成します。`CreatePlatformEndpoint` は、`CreatePlatformApplication` アクションから返される `PlatformApplicationArn` 属性を使用します。`CreatePlatformEndpoint` を使用したときに返される `EndpointArn` 属性は、モバイルアプリケーションとデバイスに通知メッセージを送信するために `Publish` アクションで使用されます。

[CreateTopic](#)

メッセージが発行されるトピックを作成します。

[DeleteEndpoint](#)

サポートされているプッシュ通知サービスのいずれかでデバイスおよびモバイルアプリケーションのエンドポイントを削除します。

[DeletePlatformApplication](#)

プラットフォームアプリケーションオブジェクトを削除します。

[DeleteTopic](#)

トピックとすべてのサブスクリプションを削除します。

[GetEndpointAttributes](#)

デバイスおよびモバイルアプリケーションのエンドポイント属性を取得します。

[GetPlatformApplicationAttributes](#)

プラットフォームアプリケーションオブジェクトの属性を取得します。

[ListEndpointsByPlatformApplication](#)

サポートされているプッシュ通知サービスでのデバイスおよびモバイルアプリケーションのエンドポイントとエンドポイント属性を一覧表示します。

[ListPlatformApplications](#)

サポートされているプッシュ通知サービス用のプラットフォームアプリケーションオブジェクトを一覧表示します。

[Publish](#)

トピックのサブスクライブされたエンドポイントすべてに通知メッセージを送信します。

[SetEndpointAttributes](#)

デバイスおよびモバイルアプリケーションのエンドポイントの属性を設定します。

[SetPlatformApplicationAttributes](#)

プラットフォームアプリケーションオブジェクトの属性を設定します。

[Subscribe](#)

エンドポイントに確認メッセージを送信して、エンドポイントのサブスクライブに備えます。実際にサブスクリプションを作成するには、エンドポイントの所有者は確認メッセージからトークンと共に ConfirmSubscription アクションを呼び出す必要があります。

[Unsubscribe](#)

サブスクリプションを削除します。

Amazon SNS モバイルプッシュ API の一般的なエラー

Amazon SNS API から返されるモバイルプッシュに関するエラーを、次の表に示します。モバイルプッシュ用 Amazon SNS API の詳細については、「[モバイルプッシュ API アクション](#)」を参照してください。

エラー	説明	HTTPS ステータスコード	API アクション
Application Name is null string	必須のアプリケーション名が null に設定されています。	400	CreatePlatformApplication
Platform Name is null string	必須のプラットフォーム名が null に設定されています。	400	CreatePlatformApplication
Platform Name is invalid	プラットフォーム名に無効な値または範囲外の値が入力されました。	400	CreatePlatformApplication
APN - Principal is not a valid certificate	APN プリンシパル (SSL 証明書) に無効な証明書が指定されました。詳細については、Amazon Simple Notification Service API リファレンスの「 CreatePlatformApplication 」を参照してください。	400	CreatePlatformApplication

エラー	説明	HTTPS ステータスコード	API アクション
APN - Principal is a valid cert but not in a .pem format	SSL プリンシパル (SSL 証明書) に、.pem 形式ではない有効な証明書が指定されました。	400	CreatePlatformApplication
APNs — Principal is an expired certificate	APN プリンシパル (SSL 証明書) に失効した証明書が指定されました。	400	CreatePlatformApplication
APNs — Principal is not an Apple issued certificate	APN プリンシパル (SSL 証明書) に Apple によって発行されたものではない証明書が指定されました。	400	CreatePlatformApplication
APNs — Principal is not provided	APN プリンシパル (SSL 証明書) が指定されませんでした。	400	CreatePlatformApplication
APNs — Credential is not provided	APN 認証情報 (プライベートキー) が指定されませんでした。詳細については、Amazon Simple Notification Service API リファレンスの「 CreatePlatformApplication 」を参照してください。	400	CreatePlatformApplication

エラー	説明	HTTPS ステータスコード	API アクション
APNs — Credential are not in a valid .pem format	APN 認証情報 (プライベートキー) が有効な .pem 形式ではありません。	400	CreatePlatformApplication
FCM — serverAPIKey is not provided	FCM 認証情報 (API キー) が指定されませんでした。詳細については、Amazon Simple Notification Service API リファレンスの「 CreatePlatformApplication 」を参照してください。	400	CreatePlatformApplication
FCM — serverAPIKey is empty	FCM 認証情報 (API キー) が空です。	400	CreatePlatformApplication
FCM — serverAPIKey is a null string	FCM 認証情報 (API キー) が null です。	400	CreatePlatformApplication
FCM — serverAPIKey is invalid	API 認証情報 (API キー) が無効です。	400	CreatePlatformApplication
ADM — clientsecret is not provided	必須のクライアントシークレットが提供されていません。	400	CreatePlatformApplication
ADM — clientsecret is a null string	クライアントシークレットの必須文字列が null です。	400	CreatePlatformApplication

エラー	説明	HTTPS ステータスコード	API アクション
ADM — client_secret is empty string	クライアントシークレットの必須文字列が空です。	400	CreatePlatformApplication
ADM — client_secret is not valid	クライアントシークレットの必須文字列が無効です。	400	CreatePlatformApplication
ADM — client_id is empty string	クライアント ID の必須文字列が空です。	400	CreatePlatformApplication
ADM — clientId is not provided	クライアント ID の必須文字列が提供されていません。	400	CreatePlatformApplication
ADM — clientid is a null string	クライアント ID の必須文字列が null です。	400	CreatePlatformApplication
ADM — client_id is not valid	クライアント ID の必須文字列が無効です。	400	CreatePlatformApplication
EventEndpointCreated has invalid ARN format	EventEndpointCreated が無効な ARN 形式です。	400	CreatePlatformApplication
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted が無効な ARN 形式です。	400	CreatePlatformApplication
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated が無効な ARN 形式です。	400	CreatePlatformApplication

エラー	説明	HTTPS ステータスコード	API アクション
EventDeliveryAttemptFailure has invalid ARN format	EventDeliveryAttemptFailure が無効な ARN 形式です。	400	CreatePlatformApplication
EventDeliveryFailure has invalid ARN format	EventDeliveryFailure が無効な ARN 形式です。	400	CreatePlatformApplication
EventEndpointCreated is not an existing Topic	EventEndpointCreated は既存のトピックではありません。	400	CreatePlatformApplication
EventEndpointDeleted is not an existing Topic	EventEndpointDeleted は既存のトピックではありません。	400	CreatePlatformApplication
EventEndpointUpdated is not an existing Topic	EventEndpointUpdated は既存のトピックではありません。	400	CreatePlatformApplication
EventDeliveryAttemptFailure is not an existing Topic	EventDeliveryAttemptFailure は既存のトピックではありません。	400	CreatePlatformApplication
EventDeliveryFailure is not an existing Topic	EventDeliveryFailure は既存のトピックではありません。	400	CreatePlatformApplication
Platform ARN is invalid	プラットフォーム ARN が無効です。	400	SetPlatformAttributes

エラー	説明	HTTPS ステータスコード	API アクション
Platform ARN is valid but does not belong to the user	プラットフォーム ARN は有効ですが、このユーザーに属していません。	400	SetPlatformAttributes
APN - Principal is not a valid certificate	APN プリンシパル (SSL 証明書) に無効な証明書が指定されました。詳細については、Amazon Simple Notification Service API リファレンスの「 CreatePlatformApplication 」を参照してください。	400	SetPlatformAttributes
APN - Principal is a valid cert but not in a .pem format	SSL プリンシパル (SSL 証明書) に、.pem 形式ではない有効な証明書が指定されました。	400	SetPlatformAttributes
APNs — Principal is an expired certificate	APN プリンシパル (SSL 証明書) に失効した証明書が指定されました。	400	SetPlatformAttributes
APNs — Principal is not an Apple issued certificate	APN プリンシパル (SSL 証明書) に Apple によって発行されたものではない証明書が指定されました。	400	SetPlatformAttributes

エラー	説明	HTTPS ステータスコード	API アクション
APNs — Principal is not provided	APN プリンシパル (SSL 証明書) が指定されませんでした。	400	SetPlatformAttributes
APNs — Credential is not provided	APN 認証情報 (プライベートキー) が指定されませんでした。詳細については、Amazon Simple Notification Service API リファレンスの「 CreatePlatformApplication 」を参照してください。	400	SetPlatformAttributes
APNs — Credential are not in a valid .pem format	APN 認証情報 (プライベートキー) が有効な .pem 形式ではありません。	400	SetPlatformAttributes
FCM — serverAPIKey is not provided	FCM 認証情報 (API キー) が指定されませんでした。詳細については、Amazon Simple Notification Service API リファレンスの「 CreatePlatformApplication 」を参照してください。	400	SetPlatformAttributes
FCM — serverAPIKey is a null string	FCM 認証情報 (API キー) が null です。	400	SetPlatformAttributes

エラー	説明	HTTPS ステータスコード	API アクション
ADM — clientId is not provided	クライアント ID の必須文字列が提供されていません。	400	SetPlatformAttributes
ADM — clientId is a null string	クライアント ID の必須文字列が null です。	400	SetPlatformAttributes
ADM — clientsecret is not provided	必須のクライアントシークレットが提供されていません。	400	SetPlatformAttributes
ADM — clientsecret is a null string	クライアントシークレットの必須文字列が null です。	400	SetPlatformAttributes
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated が無効な ARN 形式です。	400	SetPlatformAttributes
EventEndpointDeleted has invalid ARN format	EventEndpointDeleted が無効な ARN 形式です。	400	SetPlatformAttributes
EventEndpointUpdated has invalid ARN format	EventEndpointUpdated が無効な ARN 形式です。	400	SetPlatformAttributes
EventDeliveryAttemptFailure has invalid ARN format	EventDeliveryAttemptFailure が無効な ARN 形式です。	400	SetPlatformAttributes
EventDeliveryFailure has invalid ARN format	EventDeliveryFailure が無効な ARN 形式です。	400	SetPlatformAttributes

エラー	説明	HTTPS ステータスコード	API アクション
EventEndpointCreated is not an existing Topic	EventEndpointCreated は既存のトピックではありません。	400	SetPlatformAttributes
EventEndpointDeleted is not an existing Topic	EventEndpointDeleted は既存のトピックではありません。	400	SetPlatformAttributes
EventEndpointUpdated is not an existing Topic	EventEndpointUpdated は既存のトピックではありません。	400	SetPlatformAttributes
EventDeliveryAttemptFailure is not an existing Topic	EventDeliveryAttemptFailure は既存のトピックではありません。	400	SetPlatformAttributes
EventDeliveryFailure is not an existing Topic	EventDeliveryFailure は既存のトピックではありません。	400	SetPlatformAttributes
Platform ARN is invalid	プラットフォーム ARN が無効です。	400	GetPlatformApplicationAttributes
Platform ARN is valid but does not belong to the user	プラットフォーム ARN は有効ですが、このユーザーに属していません。	403	GetPlatformApplicationAttributes
Token specified is invalid	指定されたトークンが無効です。	400	ListPlatformApplications

エラー	説明	HTTPS ステータスコード	API アクション
Platform ARN is invalid	プラットフォーム ARN が無効です。	400	ListEndpointsByPlatformApplication
Platform ARN is valid but does not belong to the user	プラットフォーム ARN は有効ですが、このユーザーに属していません。	404	ListEndpointsByPlatformApplication
Token specified is invalid	指定されたトークンが無効です。	400	ListEndpointsByPlatformApplication
Platform ARN is invalid	プラットフォーム ARN が無効です。	400	DeletePlatformApplication
Platform ARN is valid but does not belong to the user	プラットフォーム ARN は有効ですが、このユーザーに属していません。	403	DeletePlatformApplication
Platform ARN is invalid	プラットフォーム ARN が無効です。	400	CreatePlatformEndpoint
Platform ARN is valid but does not belong to the user	プラットフォーム ARN は有効ですが、このユーザーに属していません。	404	CreatePlatformEndpoint
Token is not specified	トークンが指定されていません。	400	CreatePlatformEndpoint

エラー	説明	HTTPS ステータスコード	API アクション
Token is not of correct length	トークンの長さが正しくありません。	400	CreatePlatformEndpoint
Customer User data is too large	顧客のユーザーデータは UTF-8 エンコードで 2048 バイトを超えることはできません。	400	CreatePlatformEndpoint
Endpoint ARN is invalid	エンドポイント ARN が無効です。	400	DeleteEndpoint
Endpoint ARN is valid but does not belong to the user	エンドポイント ARN は有効ですが、このユーザーに属していません。	403	DeleteEndpoint
Endpoint ARN is invalid	エンドポイント ARN が無効です。	400	SetEndpointAttributes
Endpoint ARN is valid but does not belong to the user	エンドポイント ARN は有効ですが、このユーザーに属していません。	403	SetEndpointAttributes
Token is not specified	トークンが指定されていません。	400	SetEndpointAttributes
Token is not of correct length	トークンの長さが正しくありません。	400	SetEndpointAttributes

エラー	説明	HTTPS ステータスコード	API アクション
Customer User data is too large	顧客のユーザーデータは UTF-8 エンコードで 2048 バイトを超えることはできません。	400	SetEndpointAttributes
Endpoint ARN is invalid	エンドポイント ARN が無効です。	400	GetEndpointAttributes
Endpoint ARN is valid but does not belong to the user	エンドポイント ARN は有効ですが、このユーザーに属していません。	403	GetEndpointAttributes
Target ARN is invalid	ターゲット ARN が無効です。	400	Publish
Target ARN is valid but does not belong to the user	ターゲット ARN は有効ですが、このユーザーに属していません。	403	Publish
Message format is invalid	メッセージの形式が無効です。	400	Publish
Message size is larger than supported by protocol/end-service	メッセージサイズがプロトコル/エンドサービスでサポートされているより大きくなっています。	400	Publish

Amazon SNS のモバイルプッシュ通知で有効期限メッセージ属性を使用する

Amazon Simple Notification Service (Amazon SNS) では、モバイルプッシュ通知サービスの有効期限 (TTL) メッセージ属性の設定がサポートされています。この属性は、モバイルプッシュ通知サービス (Amazon Device Messaging (ADM)、Firebase Cloud Messaging (FCM、Android への送信時) など) の Amazon SNS メッセージ本文内に TTL を設定する既存の機能に追加されたものです。

TTL メッセージ属性は、メッセージの有効期限メタデータを指定するために使用します。この属性を使用して、Apple Push Notification Service (APNs) や FCM などのプッシュ通知サービスによってエンドポイントにメッセージが配信される時間を指定できます。何らかの理由 (モバイルデバイスがオフになっているなど) で、指定した TTL 内にメッセージが配信されなかった場合、そのメッセージは破棄され、以降その配信は試みられません。メッセージ属性内で TTL を指定するには、AWS ソフトウェア開発キット (SDKs) AWS マネジメントコンソール、またはクエリ API を使用できます。

プッシュ通知サービスの TTL メッセージ属性

以下は、AWS SDKs またはクエリ API の使用時に設定できるプッシュ通知サービスの TTL メッセージ属性のリストです。

プッシュ通知サービス	TTL メッセージ属性
Amazon Device Messaging (ADM)	<code>AWS.SNS.MOBILE.ADM.TTL</code>
Apple Push Notification Service (APNS)	<code>AWS.SNS.MOBILE.APNS.TTL</code>
Apple Push Notification Service Sandbox (APNs_SANDBOX)	<code>AWS.SNS.MOBILE.APNS_SANDBOX.TTL</code>
Baidu Cloud Push (Baidu)	<code>AWS.SNS.MOBILE.BAIDU.TTL</code>
Firebase Cloud Messaging (FCM、Android への送信時)	<code>AWS.SNS.MOBILE.FCM.TTL</code>
Windows Push Notification Services (WNS)	<code>AWS.SNS.MOBILE.WNS.TTL</code>

各プッシュ通知サービスは TTL を個別に処理します。Amazon SNS では、すべてのプッシュ通知サービスに対する TTL の抽象ビューが提供されるため、TTL の指定がより簡単になります。を使用して TTL AWS マネジメントコンソール を指定する場合 (秒単位)、TTL 値を 1 回入力するだけで、Amazon SNS はメッセージの発行時に選択したプッシュ通知サービスごとに TTL を計算します。

TTL は発行時間を基準にします。特定のプッシュ通知サービスにプッシュ通知メッセージを発行する前に、Amazon SNS はプッシュ通知のドウェル時間 (発行のタイムスタンプからプッシュ通知サービスへの発行直前までの時間) を計算し、残りの TTL をそのプッシュ通知サービスに渡します。TTL がドウェル時間よりも短い場合、Amazon SNS はプッシュ通知メッセージの発行を試みません。

プッシュ通知メッセージに TTL を指定する場合、TTL 値は正の整数でなければなりません。ただし、0 の値がプッシュ通知サービスで固有の意味を持つ場合 (APN や FCM (Android への送信時) など) は除きます。TTL 値を 0 に設定した場合、プッシュ通知サービスに対して 0 に特定の意味がないと、Amazon SNS はメッセージを破棄します。APN を使用した場合に 0 に設定した TTL パラメータの詳細については、[バイナリプロバイダー API](#) ドキュメントの表 A-3 リモート通知のアイテム識別子を参照してください。

TTL を決定するための優先順位

Amazon SNS がプッシュ通知メッセージの TTL を決定するための優先順位は以下の順序に基づきます。最も小さい番号が最も高い優先順位を表しています。

1. メッセージ属性の TTL
2. メッセージ本文の TTL
3. プッシュ通知サービスのデフォルト TTL (サービスごとに異なる)
4. Amazon SNS のデフォルト TTL (4 週間)

同じメッセージに対して (メッセージ属性とメッセージ本文で) 異なる TTL 値を設定した場合、Amazon SNS はメッセージ属性の TTL に一致するようにメッセージ本文の TTL を変更します。

を使用した TTL の指定 AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで [モバイル]、[プッシュ通知] を選択します。
3. [モバイルプッシュ通知] ページの [プラットフォームアプリケーション] セクションで、アプリケーションを選択します。

4. **[MyApplication]** ページの [エンドポイント] セクションでエンドポイントを選択したら、[メッセージの発行] を選択します。
5. [メッセージの詳細] セクションで、TTL (プッシュ通知サービスがエンドポイントに配信される秒数) を入力します。
6. [メッセージの発行] を選択します。

Amazon SNS モバイルアプリケーションでサポートされるリージョン

現在、モバイルアプリケーションは、次のリージョンで作成できます。

- 米国東部(オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (ミラノ)
- 欧州 (スペイン)
- 欧州 (パリ)
- ヨーロッパ (ストックホルム)
- 中東 (バーレーン)

- 中東 (アラブ首長国連邦)
- 南米 (サンパウロ)
- AWS GovCloud (米国西部)

Amazon SNS モバイルプッシュ通知を管理するためのベストプラクティス

このセクションでは、顧客エンゲージメントを高めるのに役立つ可能性があるベストプラクティスについて説明します。

エンドポイントの管理

デバイス上でのユーザーのアクション (例えば、アプリをデバイスに再インストールするなど) や、特定の iOS バージョンで実行されているデバイスに影響を与える [証明書を更新](#) によってデバイストークンが変更された場合、配信に関する問題が発生する可能性があります。Apple が推奨するベストプラクティスは、アプリを起動するたびに APN を使用して [登録](#) することです。

デバイストークンは、ユーザーがアプリを開くたびに変更されるわけではないので、べき等 [CreatePlatformEndpoint](#) API を使用できます。ただし、トークン自体が無効な場合、またはエンドポイントは有効なものの無効になっている場合 (例えば、本番環境とサンドボックス環境の不一致など)、同じデバイスに対して重複が生じることがあります。

デバイストークン管理メカニズム ([擬似コードのものなど](#)) が使用できます。

FCM v1 デバイストークンの管理とメンテナンスの詳細については、「[Amazon SNS の Firebase Cloud Messaging エンドポイントの管理](#)」を参照してください。

配信ステータスのログ記録

プッシュ通知の配信ステータスをモニタリングするには、Amazon SNS プラットフォームアプリケーションの配信ステータスログ記録を有効にすることをお勧めします。ログにはプッシュプラットフォームサービスから返されたプロバイダ [レスポンスコード](#) が含まれているため、配信が失敗したときのトラブルシューティングに役立ちます。配信ステータスログ記録の有効化の詳細については、「[プッシュ通知のために Amazon SNS トピック配信ログにアクセスするにはどうすればよいですか](#)」を参照してください。

イベント通知

イベント駆動方式でエンドポイントを管理するには、[イベント通知](#) 機能が使用できます。これにより、設定された Amazon SNS トピックは、エンドポイントの作成、削除、更新、および配信失敗の

プラットフォームアプリケーションイベントについて、Lambda 関数などのサブスクライバーにイベントをファンアウトできます。

Amazon SNS E メールサブスクリプションの設定と管理

E [メールアドレス](#)を Amazon SNS トピックにサブスクライブするには AWS マネジメントコンソール、AWS SDK for Java、または [AWS SDK for .NET](#) を使用します。

注意事項

- E メールメッセージ本文のカスタマイズはサポートされていません。E メール配信機能は、マーケティングメッセージではなく、内部システムアラートを提供することを目的としています。
- E メールエンドポイントの直接的なサブスクライブは、標準トピックのみでサポートされています。
- E メール配信スループットはスロットリングされます。詳細については、[Amazon SNS クォータ](#)を参照してください。

Important

- Amazon SNS トピック E メールメーリングリストの受信者全員に対して、受信者登録を解除できないようにするには、AWS サポートからの[登録解除に認証が必要な E メール受信登録を設定する](#)を参照してください。
- E メールまたは E メールJSON エンドポイントの 1 秒あたり 10 メッセージ (TPS) の制限を超えると、サブスクリプションの自動停止がトリガーされ、保留中の確認ステータスになります。サブスクリプションは 30 日間 PendingConfirmation 状態のままになります。その後、アクションが実行されない限り、サブスクリプションは自動的に削除されます。

を使用して Amazon SNS トピックに E メールアドレスをサブスクライブする AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. 左のナビゲーションペインで、[サブスクリプション] を選択します。

3. [サブスクリプション] ページで [サブスクリプションの作成] を選択します。
4. [サブスクリプションの作成] ページの [詳細] セクションで、以下を実行します。
 - a. [トピック ARN] で、トピックの Amazon リソースネーム (ARN) を選択します。
 - b. [プロトコル] で [E メール] を選択します。
 - c. [エンドポイント] に E メールアドレスを入力します。
 - d. (オプション) フィルターポリシーを設定するには、[サブスクリプションのフィルターポリシー] セクションを展開します。詳細については、「[Amazon SNS サブスクリプションフィルターポリシー](#)」を参照してください。
 - e. (オプション) ペイロードベースのフィルタリングを有効にするには、Filter Policy Scope を MessageBody に設定します。詳細については、「[Amazon SNS サブスクリプションフィルターポリシーの範囲](#)」を参照してください。
 - f. (オプション) サブスクリプションのデッドレターキューを設定するには、Redrive ポリシー (デッドレターキュー) を展開します。詳細については、「[Amazon SNS デッドレターキュー](#)」を参照してください。
 - g. [サブスクリプションの作成] を選択します。

コンソールがサブスクリプションを作成し、サブスクリプションの [詳細] ページが開きます。

E メールアドレスがメッセージの受信を開始する前に、サブスクリプションを確認する必要があります。

サブスクリプションを確認するには

1. Eメールの受信トレイを確認し、Amazon SNS からの E メールで [サブスクリプションの確認] を選択します。
2. Amazon SNS がウェブブラウザを開き、サブスクリプション ID とともにサブスクリプションの確認を表示します。

AWS SDK を使用して E メールアドレスを Amazon SNS トピックにサブスクライブする

AWS SDK を使用するには、認証情報を使用して設定する必要があります。詳細については、「AWS SDK とツールのリファレンスガイド」の「[共有設定ファイルと認証情報ファイル](#)」を参照してください。

次のサンプルコードは、Subscribe を使用する方法を説明しています。

.NET

SDK for .NET

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。

E メールアドレスをトピックにサブスクライブします。

```
/// <summary>
/// Creates a new subscription to a topic.
/// </summary>
/// <param name="client">The initialized Amazon SNS client object, used
/// to create an Amazon SNS subscription.</param>
/// <param name="topicArn">The ARN of the topic to subscribe to.</param>
/// <returns>A SubscribeResponse object which includes the subscription
/// ARN for the new subscription.</returns>
public static async Task<SubscribeResponse> TopicSubscribeAsync(
    IAmazonSimpleNotificationService client,
    string topicArn)
{
    SubscribeRequest request = new SubscribeRequest()
    {
        TopicArn = topicArn,
        ReturnSubscriptionArn = true,
        Protocol = "email",
        Endpoint = "recipient@example.com",
    };
}
```

```
        var response = await client.SubscribeAsync(request);

        return response;
    }
}
```

オプションのフィルターでトピックにキューをサブスクライブします。

```
/// <summary>
/// Subscribe a queue to a topic with optional filters.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <returns>The ARN of the new subscription.</returns>
public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
{
    var subscribeRequest = new SubscribeRequest()
    {
        TopicArn = topicArn,
        Protocol = "sqs",
        Endpoint = queueArn
    };

    if (!string.IsNullOrEmpty(filterPolicy))
    {
        subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
    }

    var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
    return subscribeResponse.SubscriptionArn;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
delivery to an email address.
/*!
\param topicARN: An SNS topic Amazon Resource Name (ARN).
\param emailAddress: An email address.
\param clientConfiguration: AWS client configuration.
\return bool: Function succeeded.
*/
bool AwsDoc::SNS::subscribeEmail(const Aws::String &topicARN,
                                const Aws::String &emailAddress,
                                const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("email");
    request.SetEndpoint(emailAddress);

    const Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN " <<
outcome.GetResult().GetSubscriptionArn()
        << "." << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
outcome.GetError().GetMessage()
```

```

        << std::endl;
    }

    return outcome.IsSuccess();
}

```

モバイルアプリケーションをトピックにサブスクライブします。

```

//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
delivery to a mobile app.
/*!
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
    \param endpointARN: The ARN for a mobile app or device endpoint.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool
AwsDoc::SNS::subscribeApp(const Aws::String &topicARN,
                        const Aws::String &endpointARN,
                        const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("application");
    request.SetEndpoint(endpointARN);

    const Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
outcome.GetResult().GetSubscriptionArn()
        << "'" << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
outcome.GetError().GetMessage()
        << std::endl;
    }
}

```

```
    return outcome.IsSuccess();  
}
```

Lambda 関数をトピックにサブスクライブします。

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with  
delivery to an AWS Lambda function.  
/*!  
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.  
    \param lambdaFunctionARN: The ARN for an AWS Lambda function.  
    \param clientConfiguration: AWS client configuration.  
    \return bool: Function succeeded.  
*/  
bool AwsDoc::SNS::subscribeLambda(const Aws::String &topicARN,  
                                   const Aws::String &lambdaFunctionARN,  
                                   const Aws::Client::ClientConfiguration  
&clientConfiguration) {  
    Aws::SNS::SNSClient snsClient(clientConfiguration);  
  
    Aws::SNS::Model::SubscribeRequest request;  
    request.SetTopicArn(topicARN);  
    request.SetProtocol("lambda");  
    request.SetEndpoint(lambdaFunctionARN);  
  
    const Aws::SNS::Model::SubscribeOutcome outcome =  
        snsClient.Subscribe(request);  
  
    if (outcome.IsSuccess()) {  
        std::cout << "Subscribed successfully." << std::endl;  
        std::cout << "Subscription ARN '" <<  
outcome.GetResult().GetSubscriptionArn()  
            << "'" << std::endl;  
    }  
    else {  
        std::cerr << "Error while subscribing " <<  
outcome.GetError().GetMessage()  
            << std::endl;  
    }  
  
    return outcome.IsSuccess();  
}
```

```
}
```

SQS キューをトピックにサブスクライブします。

```
Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("sqs");
    request.SetEndpoint(queueARN);

    Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
        std::cout << "The queue '" << queueName
                    << "' has been subscribed to the topic '"
                    << "'" << topicName << "'" << std::endl;
        std::cout << "with the subscription ARN '" << subscriptionARN <<
". "
                    << std::endl;
        subscriptionARNS.push_back(subscriptionARN);
    }
    else {
        std::cerr << "Error with TopicsAndQueues::Subscribe. "
                    << outcome.GetError().GetMessage()
                    << std::endl;

        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }
}
```

トピックにフィルターでサブスクライブします。

```
static const Aws::String TONE_ATTRIBUTE("tone");
static const Aws::Vector<Aws::String> TONES = {"cheerful", "funny",
"serious",
                                                    "sincere"};

Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::SNS::SNSClient snsClient(clientConfiguration);

Aws::SNS::Model::SubscribeRequest request;
request.SetTopicArn(topicARN);
request.SetProtocol("sqs");
request.SetEndpoint(queueARN);
if (isFifoTopic) {
    if (first) {
        std::cout << "Subscriptions to a FIFO topic can have
filters."
                    << std::endl;
        std::cout
            << "If you add a filter to this subscription, then
only the filtered messages "
            << "will be received in the queue." << std::endl;
        std::cout << "For information about message filtering, "
            << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
            << std::endl;
        std::cout << "For this example, you can filter messages by a
\""
                    << TONE_ATTRIBUTE << "\" attribute." << std::endl;
    }

    std::ostringstream ostream;
    ostream << "Filter messages for \"" << queueName
        << "\"'s subscription to the topic \""
        << topicName << "\"? (y/n)";

    // Add filter if user answers yes.
```

```

        if (askYesNoQuestion(ostringstream.str())) {
            Aws::String jsonPolicy = getFilterPolicyFromUser();
            if (!jsonPolicy.empty()) {
                filteringMessages = true;

                std::cout << "This is the filter policy for this
subscription."
                        << std::endl;
                std::cout << jsonPolicy << std::endl;

                request.AddAttributes("FilterPolicy", jsonPolicy);
            }
            else {
                std::cout
                    << "Because you did not select any attributes, no
filter "
                    << "will be added to this subscription." <<
std::endl;
            }
        }
        // if (isFifoTopic)
        Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

        if (outcome.IsSuccess()) {
            Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
            std::cout << "The queue '" << queueName
                    << "' has been subscribed to the topic '"
                    << "'" << topicName << "'" << std::endl;
            std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
                    << std::endl;
            subscriptionARNS.push_back(subscriptionARN);
        }
        else {
            std::cerr << "Error with TopicsAndQueues::Subscribe. "
                    << outcome.GetError().GetMessage()
                    << std::endl;

            cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,

```

```

        sqsClient);

        return false;
    }

    //! Routine that lets the user select attributes for a subscription filter
    policy.
    /*!
    \sa getFilterPolicyFromUser()
    \return Aws::String: The filter policy as JSON.
    */
    Aws::String AwsDoc::TopicsAndQueues::getFilterPolicyFromUser() {
        std::cout
            << "You can filter messages by one or more of the following \""
            << TONE_ATTRIBUTE << "\" attributes." << std::endl;

        std::vector<Aws::String> filterSelections;
        int selection;
        do {
            for (size_t j = 0; j < TONES.size(); ++j) {
                std::cout << "  " << (j + 1) << ". " << TONES[j]
                    << std::endl;
            }
            selection = askQuestionForIntRange(
                "Enter a number (or enter zero to stop adding more). ",
                0, static_cast<int>(TONES.size()));

            if (selection != 0) {
                const Aws::String &selectedTone(TONES[selection - 1]);
                // Add the tone to the selection if it is not already added.
                if (std::find(filterSelections.begin(),
                    filterSelections.end(),
                    selectedTone)
                    == filterSelections.end()) {
                    filterSelections.push_back(selectedTone);
                }
            }
        } while (selection != 0);

        Aws::String result;
        if (!filterSelections.empty()) {
            std::ostringstream jsonPolicyStream;
            jsonPolicyStream << "{ \"" << TONE_ATTRIBUTE << "\": [";

```

```
    for (size_t j = 0; j < filterSelections.size(); ++j) {
        jsonPolicyStream << "\"" << filterSelections[j] << "\"";
        if (j < filterSelections.size() - 1) {
            jsonPolicyStream << ",";
        }
    }
    jsonPolicyStream << "] ]";

    result = jsonPolicyStream.str();
}

return result;
}
```

- API の詳細については、AWS SDK for C++ API リファレンスの「[Subscribe](#)」を参照してください。

CLI

AWS CLI

トピックにサブスクライブするには

次の subscribe コマンドは、指定したトピックに E メールアドレスをサブスクライブします。

```
aws sns subscribe \
  --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic \
  --protocol email \
  --notification-endpoint my-email@example.com
```

出力:

```
{
  "SubscriptionArn": "pending confirmation"
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[Subscribe](#)」を参照してください。

Go

SDK for Go V2

 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。

オプションのフィルターでトピックにキューをサブスクライブします。

```
import (
    "context"
    "encoding/json"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/sns"
    "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type SnsActions struct {
    SnsClient *sns.Client
}

// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
// an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
// policy
// so that messages are only sent to the queue when the message has the specified
// attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
    queueArn string, filterMap map[string][]string) (string, error) {
    var subscriptionArn string
    var attributes map[string]string
    if filterMap != nil {
```

```
filterBytes, err := json.Marshal(filterMap)
if err != nil {
    log.Printf("Couldn't create filter policy, here's why: %v\n", err)
    return "", err
}
attributes = map[string]string{"FilterPolicy": string(filterBytes)}
}
output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
    Protocol:          aws.String("sqs"),
    TopicArn:          aws.String(topicArn),
    Attributes:         attributes,
    Endpoint:          aws.String(queueArn),
    ReturnSubscription: true,
})
if err != nil {
    log.Printf("Couldn't subscribe queue %v to topic %v. Here's why: %v\n",
        queueArn, topicArn, err)
} else {
    subscriptionArn = *output.SubscriptionArn
}

return subscriptionArn, err
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * 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
 */
public class SubscribeEmail {
    public static void main(String[] args) {
        final String usage = ""
            Usage:      <topicArn> <email>

            Where:
                topicArn - The ARN of the topic to subscribe.
                email - The email address to use.
            "";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String email = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        subEmail(snsClient, topicArn, email);
        snsClient.close();
    }

    public static void subEmail(SnsClient snsClient, String topicArn, String
email) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                .protocol("email")
```

```

        .endpoint(email)
        .returnSubscriptionArn(true)
        .topicArn(topicArn)
        .build();

    SubscribeResponse result = snsClient.subscribe(request);
    System.out.println("Subscription ARN: " + result.subscriptionArn() +
"\n\n Status is "
        + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

HTTP エンドポイントをトピックにサブスクライブします。

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * 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
 */
public class SubscribeHTTPS {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicArn> <url>

            Where:
                topicArn - The ARN of the topic to subscribe.

```

```

        url - The HTTPS endpoint that you want to receive
        notifications.
        """;

    if (args.length < 2) {
        System.out.println(usage);
        System.exit(1);
    }

    String topicArn = args[0];
    String url = args[1];
    SnsClient snsClient = SnsClient.builder()
        .region(Region.US_EAST_1)
        .build();

    subHTTPS(snsClient, topicArn, url);
    snsClient.close();
}

public static void subHTTPS(SnsClient snsClient, String topicArn, String url)
{
    try {
        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("https")
            .endpoint(url)
            .returnSubscriptionArn(true)
            .topicArn(topicArn)
            .build();

        SubscribeResponse result = snsClient.subscribe(request);
        System.out.println("Subscription ARN is " + result.subscriptionArn()
+ "\n\n Status is "
            + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

Lambda 関数をトピックにサブスクライブします。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * 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
 */
public class SubscribeLambda {

    public static void main(String[] args) {

        final String usage = ""

            Usage:    <topicArn> <lambdaArn>

            Where:
                topicArn - The ARN of the topic to subscribe.
                lambdaArn - The ARN of an AWS Lambda function.
            "";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String lambdaArn = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        String arnValue = subLambda(snsClient, topicArn, lambdaArn);
        System.out.println("Subscription ARN: " + arnValue);
        snsClient.close();
    }
}
```

```
public static String subLambda(SnsClient snsClient, String topicArn, String
lambdaArn) {
    try {
        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("lambda")
            .endpoint(lambdaArn)
            .returnSubscriptionArn(true)
            .topicArn(topicArn)
            .build();

        SubscribeResponse result = snsClient.subscribe(request);
        return result.subscriptionArn();

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[Subscribe](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
it blank
```

```
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm
 * a subscription.
 * @param {string} emailAddress - The email address that is subscribed to the
 * topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
};
```

モバイルアプリケーションをトピックにサブスクライブします。


```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing
 * to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint
 * is created
 *                               when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

Lambda 関数をトピックにサブスクライブします。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
```

```
* @param {string} topicArn - The ARN of the topic the subscriber is subscribing
to.
* @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
*/
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
  return response;
};
```

SQS キューをトピックにサブスクライブします。

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
```

```

    Protocol: "sqs",
    Endpoint: queueArn,
  });

const response = await client.send(command);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
// }
return response;
};

```

トピックにフィルターでサブスクライブします。

```

import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute
      set to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

```

```
    },  
  });  
  
  const response = await client.send(command);  
  console.log(response);  
  // {  
  //   '$metadata': {  
  //     httpStatusCode: 200,  
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',  
  //     extendedRequestId: undefined,  
  //     cfId: undefined,  
  //     attempts: 1,  
  //     totalRetryDelay: 0  
  //   },  
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-  
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'  
  // }  
  return response;  
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
suspend fun subEmail(  
    topicArnVal: String,  
    email: String,  
): String {
```

```
val request =
    SubscribeRequest {
        protocol = "email"
        endpoint = email
        returnSubscriptionArn = true
        topicArn = topicArnVal
    }

SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
    val result = snsClient.subscribe(request)
    return result.subscriptionArn.toString()
}
}
```

Lambda 関数をトピックにサブスクライブします。

```
suspend fun subLambda(
    topicArnVal: String?,
    lambdaArn: String?,
) {
    val request =
        SubscribeRequest {
            protocol = "lambda"
            endpoint = lambdaArn
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        println(" The subscription Arn is ${result.subscriptionArn}")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[Subscribe](#)」を参照してください。

PHP

SDK for PHP

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。

E メールアドレスをトピックにサブスクライブします。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 * message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$protocol = 'email';
$endpoint = 'sample@example.com';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
```

```
]);  
var_dump($result);  
} catch (AwsException $e) {  
    // output error message if fails  
    error_log($e->getMessage());  
}
```

HTTP エンドポイントをトピックにサブスクライブします。

```
require 'vendor/autoload.php';  
  
use Aws\Exception\AwsException;  
use Aws\Sns\SnsClient;  
  
/**  
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation  
 * message.  
 *  
 * This code expects that you have AWS credentials set up per:  
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/  
guide\_credentials.html  
 */  
  
$SnSClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);  
  
$protocol = 'https';  
$endpoint = 'https://';  
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';  
  
try {  
    $result = $SnSClient->subscribe([  
        'Protocol' => $protocol,  
        'Endpoint' => $endpoint,  
        'ReturnSubscriptionArn' => true,  
        'TopicArn' => $topic,  
    ]);
```

```
var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- API の詳細については、AWS SDK for PHP API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def subscribe(topic, protocol, endpoint):
        """
        Subscribes an endpoint to the topic. Some endpoint types, such as email,
        must be confirmed before their subscriptions are active. When a
        subscription
        is not confirmed, its Amazon Resource Number (ARN) is set to
        'PendingConfirmation'.
```



```

        :param topic: The topic to subscribe to.
        :param protocol: The protocol of the endpoint, such as 'sms' or 'email'.
        :param endpoint: The endpoint that receives messages, such as a phone
number
                        (in E.164 format) for SMS messages, or an email address
for
                        email messages.
        :return: The newly added subscription.
        """
        try:
            subscription = topic.subscribe(
                Protocol=protocol, Endpoint=endpoint, ReturnSubscriptionArn=True
            )
            logger.info("Subscribed %s %s to topic %s.", protocol, endpoint,
topic.arn)
        except ClientError:
            logger.exception(
                "Couldn't subscribe %s %s to topic %s.", protocol, endpoint,
topic.arn
            )
            raise
        else:
            return subscription

```

オプションのフィルターでトピックにキューをサブスクライブします。

```

class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

```

```

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

def subscribe_queue_to_topic(
    self,
    topic_arn: str,
    queue_arn: str,
    filter_policy: Optional[str] = None
) -> str:
    """
    Subscribe an SQS queue to an SNS topic.

    :param topic_arn: The ARN of the SNS topic.
    :param queue_arn: The ARN of the SQS queue.
    :param filter_policy: Optional JSON filter policy for message filtering.
    :return: The ARN of the subscription.
    :raises ClientError: If the subscription fails.
    """
    try:
        attributes = {}
        if filter_policy:
            attributes['FilterPolicy'] = filter_policy

        response = self.sns_client.subscribe(
            TopicArn=topic_arn,
            Protocol='sqs',
            Endpoint=queue_arn,
            Attributes=attributes
        )

        subscription_arn = response['SubscriptionArn']
        logger.info(f"Subscribed queue {queue_arn} to topic {topic_arn}")
        return subscription_arn

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error subscribing queue to topic: {error_code} - {e}")
        raise

```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[Subscribe](#)」を参照してください。

Ruby

SDK for Ruby

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。

E メールアドレスをトピックにサブスクライブします。

```
require 'aws-sdk-sns'
require 'logger'

# Represents a service for creating subscriptions in Amazon Simple Notification
# Service (SNS)
class SubscriptionService
  # Initializes the SubscriptionService with an SNS client
  #
  # @param sns_client [Aws::SNS::Client] The SNS client
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Attempts to create a subscription to a topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param protocol [String] The subscription protocol (e.g., email)
  # @param endpoint [String] The endpoint that receives the notifications (email
  # address)
  # @return [Boolean] true if subscription was successfully created, false
  # otherwise
  def create_subscription(topic_arn, protocol, endpoint)
    @sns_client.subscribe(topic_arn: topic_arn, protocol: protocol, endpoint:
    endpoint)
```

```
@logger.info('Subscription created successfully.')
true
rescue Aws::SNS::Errors::ServiceError => e
  @logger.error("Error while creating the subscription: #{e.message}")
  false
end
end

# Main execution if the script is run directly
if $PROGRAM_NAME == __FILE__
  protocol = 'email'
  endpoint = 'EMAIL_ADDRESS' # Should be replaced with a real email address
  topic_arn = 'TOPIC_ARN'     # Should be replaced with a real topic ARN

  sns_client = Aws::SNS::Client.new
  subscription_service = SubscriptionService.new(sns_client)

  @logger.info('Creating the subscription.')
  unless subscription_service.create_subscription(topic_arn, protocol, endpoint)
    @logger.error('Subscription creation failed. Stopping program.')
    exit 1
  end
end
end
```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for Ruby API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`", topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;

    println!("Published message: {:?}", rsp);

    Ok(())
}
```

- APIの詳細については、AWS SDK for Rust API リファレンスの「[Subscribe](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

E メールアドレスをトピックにサブスクライブします。

```
TRY.
    oo_result = lo_sns->subscribe(
        iv_topicarn = iv_topic_arn
        iv_protocol = 'email'
        iv_endpoint = iv_email_address
        iv_returnsubscriptionarn = abap_true ).
    MESSAGE 'Email address subscribed to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
CATCH /aws1/cx_snssubscriptionlmt00.
    MESSAGE 'Unable to create subscriptions. You have reached the maximum
number of subscriptions allowed.' TYPE 'E'.
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[Subscribe](#)」を参照してください。

Swift

SDK for Swift

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。

E メールアドレスをトピックにサブスクライブします。

```
import AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

let output = try await snsClient.subscribe(
    input: SubscribeInput(
        endpoint: email,
```

```
        protocol: "email",
        returnSubscriptionArn: true,
        topicArn: arn
    )
)

guard let subscriptionArn = output.subscriptionArn else {
    print("No subscription ARN received from Amazon SNS.")
    return
}

print("Subscription \(subscriptionArn) created.")
```

SMS で通知を受け取るには、電話番号をトピックに登録します。

```
import AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

let output = try await snsClient.subscribe(
    input: SubscribeInput(
        endpoint: phone,
        protocol: "sms",
        returnSubscriptionArn: true,
        topicArn: arn
    )
)

guard let subscriptionArn = output.subscriptionArn else {
    print("No subscription ARN received from Amazon SNS.")
    return
}

print("Subscription \(subscriptionArn) created.")
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[Subscribe](#)」を参照してください。

SDK を使用した Amazon SNS のコード例 AWS SDKs

次のコード例は、AWS Software Development Kit (SDK) で Amazon SNS を使用する方法を示しています。

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。

シナリオは、1つのサービス内から、または他の AWS のサービスと組み合わせて複数の関数を呼び出し、特定のタスクを実行する方法を示すコード例です。

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 での Amazon SNS 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 を使用した Amazon SNS の基本的な例](#)
 - [Hello Amazon SNS](#)
 - [SDK を使用した Amazon SNS のアクション AWS SDKs](#)
 - [AWS SDK または CLI CheckIfPhoneNumberIsOptedOutで を使用する](#)
 - [AWS SDK または CLI ConfirmSubscriptionで を使用する](#)
 - [AWS SDK または CLI CreateTopicで を使用する](#)
 - [AWS SDK または CLI DeleteTopicで を使用する](#)
 - [AWS SDK または CLI GetSMSAttributesで を使用する](#)
 - [AWS SDK または CLI GetTopicAttributesで を使用する](#)
 - [AWS SDK または CLI ListPhoneNumbersOptedOutで を使用する](#)
 - [AWS SDK または CLI ListSubscriptionsで を使用する](#)
 - [AWS SDK または CLI ListTopicsで を使用する](#)
 - [AWS SDK または CLI Publishで を使用する](#)
 - [AWS SDK または CLI SetSMSAttributesで を使用する](#)
 - [AWS SDK または CLI SetSubscriptionAttributesで を使用する](#)
 - [AWS SDK SetSubscriptionAttributesRedrivePolicyで を使用する](#)
 - [AWS SDK または CLI SetTopicAttributesで を使用する](#)

- [AWS SDK または CLI Subscribeで を使用する](#)
- [AWS SDK または CLI TagResourceで を使用する](#)
- [AWS SDK または CLI Unsubscribeで を使用する](#)
- [SDK を使用した Amazon SNS のシナリオ AWS SDKs](#)
 - [DynamoDB テーブルにデータを送信するアプリケーションを構築する](#)
 - [メッセージを翻訳する公開およびサブスクリプションアプリケーションを構築する](#)
 - [AWS SDK を使用して Amazon SNS プッシュ通知用のプラットフォームエンドポイントを作成する](#)
 - [ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成](#)
 - [Amazon Textract エクスプローラーアプリケーションを作成する](#)
 - [AWS SDK を使用して FIFO Amazon SNS トピックを作成して公開する](#)
 - [AWS SDK を使用して Amazon Rekognition でビデオ内の人物とオブジェクトを検出する](#)
 - [AWS SDK を使用して Amazon SNS トピックに SMS メッセージを発行する](#)
 - [AWS SDK を使用して Amazon S3 で Amazon SNS Amazon S3に大規模なメッセージを発行する](#)
 - [AWS SDK を使用して Amazon SNS SMS テキストメッセージを発行する](#)
 - [AWS SDK を使用して Amazon SNS メッセージを Amazon SQS キューに発行する](#)
 - [API Gateway を使用して Lambda 関数を呼び出す](#)
 - [スケジュールされたイベントを使用した Lambda 関数の呼び出し](#)
- [Amazon SNS のサーバーレス例](#)
 - [Amazon SNS トリガーから Lambda 関数を呼び出す](#)

AWS SDKs を使用した Amazon SNS の基本的な例

次のコード例は、AWS SDK を使用した Amazon Simple Notification Service の基本的な使用方法を説明しています。

例

- [Hello Amazon SNS](#)
- [SDK を使用した Amazon SNS のアクション AWS SDKs](#)
 - [AWS SDK または CLI CheckIfPhoneNumberIsOptedOutで を使用する](#)
 - [AWS SDK または CLI ConfirmSubscriptionで を使用する](#)

- [AWS SDK または CLI CreateTopic で 使用する](#)
- [AWS SDK または CLI DeleteTopic で 使用する](#)
- [AWS SDK または CLI GetSMSAttributes で 使用する](#)
- [AWS SDK または CLI GetTopicAttributes で 使用する](#)
- [AWS SDK または CLI ListPhoneNumbersOptedOut で 使用する](#)
- [AWS SDK または CLI ListSubscriptions で 使用する](#)
- [AWS SDK または CLI ListTopics で 使用する](#)
- [AWS SDK または CLI Publish で 使用する](#)
- [AWS SDK または CLI SetSMSAttributes で 使用する](#)
- [AWS SDK または CLI SetSubscriptionAttributes で 使用する](#)
- [AWS SDK SetSubscriptionAttributesRedrivePolicy で 使用する](#)
- [AWS SDK または CLI SetTopicAttributes で 使用する](#)
- [AWS SDK または CLI Subscribe で 使用する](#)
- [AWS SDK または CLI TagResource で 使用する](#)
- [AWS SDK または CLI Unsubscribe で 使用する](#)

Hello Amazon SNS

以下のコード例は、Amazon SNS の使用を開始する方法を示しています。

.NET

SDK for .NET

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。

```
using Amazon.SimpleNotificationService;  
using Amazon.SimpleNotificationService.Model;  
  
namespace SNSActions;
```

```
public static class HelloSNS
{
    static async Task Main(string[] args)
    {
        var snsClient = new AmazonSimpleNotificationServiceClient();

        Console.WriteLine($"Hello Amazon SNS! Following are some of your
topics:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get a list of topics.
        var response = await snsClient.ListTopicsAsync(
            new ListTopicsRequest());

        foreach (var topic in response.Topics)
        {
            Console.WriteLine($"\\tTopic ARN: {topic.TopicArn}");
            Console.WriteLine();
        }
    }
}
```

- APIの詳細については、AWS SDK for .NET API リファレンスの「[ListTopics](#)」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。

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 sns)

# Set this project's name.
project("hello_sns")

# 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_sns.cpp)

target_link_libraries(${PROJECT_NAME}
  ${AWSSDK_LINK_LIBRARIES})
```

hello_sns.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/sns/SNSClient.h>
#include <aws/sns/model/ListTopicsRequest.h>
#include <iostream>

/*
 * A "Hello SNS" starter application which initializes an Amazon Simple
 Notification
 * Service (Amazon SNS) client and lists the SNS topics in the current account.
 *
 * main function
 *
 * Usage: 'hello_sns'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
    // options.loggingOptions.logLevel = 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::SNS::SNSClient snsClient(clientConfig);

        Aws::Vector<Aws::SNS::Model::Topic> allTopics;
        Aws::String nextToken; // Next token is used to handle a paginated
        response.
        do {
            Aws::SNS::Model::ListTopicsRequest request;

            if (!nextToken.empty()) {
                request.SetNextToken(nextToken);
            }

            const Aws::SNS::Model::ListTopicsOutcome outcome =
            snsClient.ListTopics(
                request);
```

```

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::SNS::Model::Topic> &paginatedTopics =
                outcome.GetResult().GetTopics();
            if (!paginatedTopics.empty()) {
                allTopics.insert(allTopics.cend(), paginatedTopics.cbegin(),
                                paginatedTopics.cend());
            }
        }
        else {
            std::cerr << "Error listing topics " <<
outcome.GetError().GetMessage()
                << std::endl;
            return 1;
        }

        nextToken = outcome.GetResult().GetNextToken();
    } while (!nextToken.empty());

    std::cout << "Hello Amazon SNS! You have " << allTopics.size() << "
topic"
                << (allTopics.size() == 1 ? "" : "s") << " in your account."
                << std::endl;

    if (!allTopics.empty()) {
        std::cout << "Here are your topic ARNs." << std::endl;
        for (const Aws::SNS::Model::Topic &topic: allTopics) {
            std::cout << "  * " << topic.GetTopicArn() << 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「[ListTopics](#)」を参照してください。

Go

SDK for Go V2

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。

```
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/sns"
    "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
// Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    snsClient := sns.NewFromConfig(sdkConfig)
    fmt.Println("Let's list the topics for your account.")
    var topics []types.Topic
    paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
    for paginator.HasMorePages() {
```

```
output, err := paginator.NextPage(ctx)
if err != nil {
    log.Printf("Couldn't get topics. Here's why: %v\n", err)
    break
} else {
    topics = append(topics, output.Topics...)
}
}
if len(topics) == 0 {
    fmt.Println("You don't have any topics!")
} else {
    for _, topic := range topics {
        fmt.Printf("\t%v\n", *topic.TopicArn)
    }
}
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[ListTopics](#)」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。

```
package com.example.sns;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.paginators.ListTopicsIterable;

public class HelloSNS {
    public static void main(String[] args) {
```



```
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();

listSNSTopics(snsClient);
snsClient.close();
}

public static void listSNSTopics(SnsClient snsClient) {
    try {
        ListTopicsIterable listTopics = snsClient.listTopicsPaginator();
        listTopics.stream()
            .flatMap(r -> r.topics().stream())
            .forEach(content -> System.out.println(" Topic ARN: " +
content.topicArn()));

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[ListTopics](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

SNS クライアントを初期化し、アカウントのトピックを一覧表示します。

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";

export const helloSns = async () => {
```

```
// The configuration object ( `{}` ) is required. If the region and credentials
// are omitted, the SDK uses your local configuration if it exists.
const client = new SNSClient({});

// You can also use `ListTopicsCommand`, but to use that command you must
// handle the pagination yourself. You can do that by sending the
`ListTopicsCommand`
// with the `NextToken` parameter from the previous request.
const paginatedTopics = paginateListTopics({ client }, {});
const topics = [];

for await (const page of paginatedTopics) {
  if (page.Topics?.length) {
    topics.push(...page.Topics);
  }
}

const suffix = topics.length === 1 ? "" : "s";

console.log(
  `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your
  account.` ,
);
console.log(topics.map((t) => ` * ${t.TopicArn}`).join("\n"));
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「[ListTopics](#)」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.sns.SnsClient
```

```
import aws.sdk.kotlin.services.sns.model.ListTopicsRequest
import aws.sdk.kotlin.services.sns.paginators.listTopicsPaginated
import kotlinx.coroutines.flow.transform

/**
Before running this Kotlin 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-kotlin/latest/developer-guide/setup.html
*/
suspend fun main() {
    listTopicsPag()
}

suspend fun listTopicsPag() {
    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient
            .listTopicsPaginated(ListTopicsRequest { })
            .transform { it.topics?.forEach { topic -> emit(topic) } }
            .collect { topic ->
                println("The topic ARN is ${topic.topicArn}")
            }
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[ListTopics](#)」を参照してください。

Swift

SDK for Swift

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。

Package.swift ファイル。

```
import PackageDescription

let package = Package(
    name: "sns-basics",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/aws-labs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
        // or a test suite.
        // Targets can depend on other targets in this package and products
        // from dependencies.
        .executableTarget(
            name: "sns-basics",
            dependencies: [
                .product(name: "AWSSNS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")
    ]
)
```

メイン Swift プログラム。

```
import ArgumentParser
import AWSClientRuntime
import AWSSNS
import Foundation
```

```
struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use (default: us-east-1)")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "sns-basics",
        abstract: ""
        This example shows how to list all of your available Amazon SNS topics.
        "",
        discussion: ""
        ""
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await SNSClient.SNSClientConfiguration(region: region)
        let snsClient = SNSClient(config: config)

        var topics: [String] = []
        let outputPages = snsClient.listTopicsPaginated(
            input: ListTopicsInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let topicList = output.topics else {
                print("Unable to get a page of Amazon SNS topics.")
                return
            }

            // Iterate over the topics listed on this page, adding their ARNs
            // to the `topics` array.

            for topic in topicList {
                guard let arn = topic.topicArn else {
                    print("Topic has no ARN.")
                    return
                }
                topics.append(arn)
            }
        }
    }
}
```

```
        print("You have \(topics.count) topics:")
        for topic in topics {
            print("    \(topic)")
        }
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[ListTopics](#)」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 での Amazon SNS 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。

SDK を使用した Amazon SNS のアクション AWS SDKs

次のコード例は、AWS SDKs を使用して個々の Amazon SNS 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。

これらは Amazon SNS API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。アクションは [SDK を使用した Amazon SNS のシナリオ AWS SDKs](#) のコンテキスト内で確認できます。

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。リストの詳細については、「[Amazon Simple Notification Service API リファレンス](#)」を参照してください。

例

- [AWS SDK または CLI `CheckIfPhoneNumberIsOptedOut`で を使用する](#)
- [AWS SDK または CLI `ConfirmSubscription`で を使用する](#)
- [AWS SDK または CLI `CreateTopic`で を使用する](#)
- [AWS SDK または CLI `DeleteTopic`で を使用する](#)
- [AWS SDK または CLI `GetSMSAttributes`で を使用する](#)
- [AWS SDK または CLI `GetTopicAttributes`で を使用する](#)
- [AWS SDK または CLI `ListPhoneNumbersOptedOut`で を使用する](#)
- [AWS SDK または CLI `ListSubscriptions`で を使用する](#)
- [AWS SDK または CLI `ListTopics`で を使用する](#)
- [AWS SDK または CLI `Publish`で を使用する](#)
- [AWS SDK または CLI `SetSMSAttributes`で を使用する](#)
- [AWS SDK または CLI `SetSubscriptionAttributes`で を使用する](#)
- [AWS SDK `SetSubscriptionAttributesRedrivePolicy`で を使用する](#)
- [AWS SDK または CLI `SetTopicAttributes`で を使用する](#)
- [AWS SDK または CLI `Subscribe`で を使用する](#)
- [AWS SDK または CLI `TagResource`で を使用する](#)
- [AWS SDK または CLI `Unsubscribe`で を使用する](#)

AWS SDK または CLI `CheckIfPhoneNumberIsOptedOut`で を使用する

次のサンプルコードは、`CheckIfPhoneNumberIsOptedOut` を使用する方法を説明しています。

.NET

SDK for .NET

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。

```
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use the Amazon Simple Notification Service
/// (Amazon SNS) to check whether a phone number has been opted out.
/// </summary>
public class IsPhoneNumOptedOut
{
    public static async Task Main()
    {
        string phoneNumber = "+15551112222";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await CheckIfOptedOutAsync(client, phoneNumber);
    }

    /// <summary>
    /// Checks to see if the supplied phone number has been opted out.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS Client object used
    /// to check if the phone number has been opted out.</param>
    /// <param name="phoneNumber">A string representing the phone number
    /// to check.</param>
    public static async Task
CheckIfOptedOutAsync(IAmazonSimpleNotificationService client, string
phoneNumber)
    {
        var request = new CheckIfPhoneNumberIsOptedOutRequest
        {
            PhoneNumber = phoneNumber,
        };

        try
        {
            var response = await
client.CheckIfPhoneNumberIsOptedOutAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
```



```
        {
            string optOutStatus = response.IsOptedOut ? "opted out" :
            "not opted out.";
            Console.WriteLine($"The phone number: {phoneNumber} is
            {optOutStatus}");
        }
    }
    catch (AuthorizationErrorException ex)
    {
        Console.WriteLine($"{ex.Message}");
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[CheckIfPhoneNumberIsOptedOut](#)」を参照してください。

CLI

AWS CLI

電話番号での SMS メッセージのオプトアウトを確認するには

次の `check-if-phone-number-is-opted-out` 例では、指定された電話番号が現在の AWS アカウントからの SMS メッセージの受信をオプトアウトされているかどうかを確認します。

```
aws sns check-if-phone-number-is-opted-out \
    --phone-number +1555550100
```

出力:

```
{
    "isOptedOut": false
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[CheckIfPhoneNumberIsOptedOut](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import
    software.amazon.awssdk.services.sns.model.CheckIfPhoneNumberIsOptedOutRequest;
import
    software.amazon.awssdk.services.sns.model.CheckIfPhoneNumberIsOptedOutResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class CheckOptOut {
    public static void main(String[] args) {

        final String usage = ""

            Usage:    <phoneNumber>

            Where:
                phoneNumber - The mobile phone number to look up (for example,
+1XXX5550100).

            "";

        if (args.length != 1) {
            System.out.println(usage);
        }
    }
}
```

```
        System.exit(1);
    }

    String phoneNumber = args[0];
    SnsClient snsClient = SnsClient.builder()
        .region(Region.US_EAST_1)
        .build();

    checkPhone(snsClient, phoneNumber);
    snsClient.close();
}

public static void checkPhone(SnsClient snsClient, String phoneNumber) {
    try {
        CheckIfPhoneNumberIsOptedOutRequest request =
            CheckIfPhoneNumberIsOptedOutRequest.builder()
                .phoneNumber(phoneNumber)
                .build();

        CheckIfPhoneNumberIsOptedOutResponse result =
            snsClient.checkIfPhoneNumberIsOptedOut(request);
        System.out.println(
            result.isOptedOut() + "Phone Number " + phoneNumber + " has
Opted Out of receiving sns messages." +
            "\n\nStatus was " +
            result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[CheckIfPhoneNumberIsOptedOut](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
//      totalRetryDelay: 0
//    },
//    isOptedOut: false
//  }
  return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの [CheckIfPhoneNumberIsOptedOut](#) を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Indicates whether the phone number owner has opted out of receiving SMS
 * messages from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnsClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
```

```
]);

$phone = '+1XXX5550100';

try {
    $result = $SnSClient->checkIfPhoneNumberIsOptedOut([
        'phoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの [CheckIfPhoneNumberIsOptedOut](#) 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 での Amazon SNS 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 **ConfirmSubscription**で 使用する

次のサンプルコードは、ConfirmSubscription を使用する方法を説明しています。

CLI

AWS CLI

サブスクリプションを確認するには

次の confirm-subscription コマンドは、my-topic という名前の SNS トピックをサブスクライブしたときに開始された確認プロセスを完了します。--token パラメータは、subscribe の呼び出しで指定した通知エンドポイントに送信される確認メッセージから取得されます。

```
aws sns confirm-subscription \
    --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic \
```

```
--  
token 2336412f37fb687f5d51e6e241d7700ae02f7124d8268910b858cb4db727ceeb2474bb937929d3bdd7c
```

出力:

```
{  
  "SubscriptionArn": "arn:aws:sns:us-west-2:123456789012:my-  
topic:8a21d249-4329-4871-acc6-7be709c6ea7f"  
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ConfirmSubscription](#)」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。

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sns.SnsClient;  
import software.amazon.awssdk.services.sns.model.ConfirmSubscriptionRequest;  
import software.amazon.awssdk.services.sns.model.ConfirmSubscriptionResponse;  
import software.amazon.awssdk.services.sns.model.SnsException;  
  
/**  
 * 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  
 */  
public class ConfirmSubscription {  
    public static void main(String[] args) {  
        final String usage = ""
```

```
Usage:    <subscriptionToken> <topicArn>

Where:
    subscriptionToken - A short-lived token sent to an endpoint
during the Subscribe action.
    topicArn - The ARN of the topic.\s
    """;

if (args.length != 2) {
    System.out.println(usage);
    System.exit(1);
}

String subscriptionToken = args[0];
String topicArn = args[1];
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();

confirmSub(snsClient, subscriptionToken, topicArn);
snsClient.close();
}

public static void confirmSub(SnsClient snsClient, String subscriptionToken,
String topicArn) {
    try {
        ConfirmSubscriptionRequest request =
ConfirmSubscriptionRequest.builder()
            .token(subscriptionToken)
            .topicArn(topicArn)
            .build();

        ConfirmSubscriptionResponse result =
snsClient.confirmSubscription(request);
        System.out.println("\n\nStatus was " +
result.sdkHttpResponse().statusCode() + "\n\nSubscription Arn: \n\n"
            + result.subscriptionArn());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```



```
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[ConfirmSubscription](#)」参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                        that are not AWS services (HTTP/S, email) need to be
 *                        confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm
 *                        a subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
```

```
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while
      unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
  xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの [ConfirmSubscription](#) を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Verifies an endpoint owner's intent to receive messages by
 * validating the token sent to the endpoint by an earlier Subscribe action.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$subscription_token = 'arn:aws:sns:us-east-1:111122223333:MyTopic:123456-
abcd-12ab-1234-12ba3dc1234a';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->confirmSubscription([
        'Token' => $subscription_token,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- APIの詳細については、AWS SDK for PHP API リファレンスの「[ConfirmSubscription](#)」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 での Amazon SNS 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 **CreateTopic**で を使用する

次のサンプルコードは、CreateTopic を使用する方法を説明しています。

アクション例は、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。次のコード例で、このアクションのコンテキストを確認できます。

- [FIFO トピックを作成して発行する](#)
- [メッセージをキューに発行する](#)

.NET

SDK for .NET

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。

トピックを作成して、個別の名前を付けます。

```
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example shows how to use Amazon Simple Notification Service
/// (Amazon SNS) to add a new Amazon SNS topic.
/// </summary>
public class CreateSNSTopic
{
    public static async Task Main()
    {
        string topicName = "ExampleSNSTopic";
```

```

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var topicArn = await CreateSNSTopicAsync(client, topicName);
        Console.WriteLine($"New topic ARN: {topicArn}");
    }

    /// <summary>
    /// Creates a new SNS topic using the supplied topic name.
    /// </summary>
    /// <param name="client">The initialized SNS client object used to
    /// create the new topic.</param>
    /// <param name="topicName">A string representing the topic name.</param>
    /// <returns>The Amazon Resource Name (ARN) of the created topic.</
returns>
    public static async Task<string>
CreateSNSTopicAsync(IAmazonSimpleNotificationService client, string topicName)
    {
        var request = new CreateTopicRequest
        {
            Name = topicName,
        };

        var response = await client.CreateTopicAsync(request);

        return response.TopicArn;
    }
}

```

名前と特定の FIFO および重複除外属性を使用して新しいトピックを作成します。

```

    /// <summary>
    /// Create a new topic with a name and specific FIFO and de-duplication
    attributes.
    /// </summary>
    /// <param name="topicName">The name for the topic.</param>
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>
    /// <param name="useContentBasedDeduplication">True to use content-based de-
    duplication.</param>
    /// <returns>The ARN of the new topic.</returns>

```

```
public async Task<string> CreateTopicWithName(string topicName, bool
useFifoTopic, bool useContentBasedDeduplication)
{
    var createTopicRequest = new CreateTopicRequest()
    {
        Name = topicName,
    };

    if (useFifoTopic)
    {
        // Update the name if it is not correct for a FIFO topic.
        if (!topicName.EndsWith(".fifo"))
        {
            createTopicRequest.Name = topicName + ".fifo";
        }

        // Add the attributes from the method parameters.
        createTopicRequest.Attributes = new Dictionary<string, string>
        {
            { "FifoTopic", "true" }
        };
        if (useContentBasedDeduplication)
        {
            createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
        }
    }

    var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
    return createResponse.TopicArn;
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[CreateTopic](#)」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 Amazon Simple Notification Service (Amazon SNS) topic.
/*!
    \param topicName: An Amazon SNS topic name.
    \param topicARNResult: String to return the Amazon Resource Name (ARN) for the
    topic.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::createTopic(const Aws::String &topicName,
                              Aws::String &topicARNResult,
                              const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::CreateTopicRequest request;
    request.SetName(topicName);

    const Aws::SNS::Model::CreateTopicOutcome outcome =
snsClient.CreateTopic(request);

    if (outcome.IsSuccess()) {
        topicARNResult = outcome.GetResult().GetTopicArn();
        std::cout << "Successfully created an Amazon SNS topic " << topicName
                    << " with topic ARN '" << topicARNResult
                    << "'." << std::endl;
    }
    else {
        std::cerr << "Error creating topic " << topicName << ":" <<
                    outcome.GetError().GetMessage() << std::endl;
        topicARNResult.clear();
    }
}
```

```
    return outcome.IsSuccess();  
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「[CreateTopic](#)」を参照してください。

CLI

AWS CLI

SNS トピックを作成するには

次の create-topic の例では、my-topic という名前の SNS トピックを作成します。

```
aws sns create-topic \  
  --name my-topic
```

出力:

```
{  
  "ResponseMetadata": {  
    "RequestId": "1469e8d7-1642-564e-b85d-a19b4b341f83"  
  },  
  "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic"  
}
```

詳細については、[AWS 「コマンドラインインターフェイスユーザーガイド」のAmazon SQS および Amazon SNS でのコマンドラインインターフェイスの使用](#)を参照してください。

AWS

- API の詳細については、「AWS CLI コマンドリファレンス」の「[CreateTopic](#)」を参照してください。

Go

SDK for Go V2

 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 (  
    "context"  
    "encoding/json"  
    "log"  
  
    "github.com/aws/aws-sdk-go-v2/aws"  
    "github.com/aws/aws-sdk-go-v2/service/sns"  
    "github.com/aws/aws-sdk-go-v2/service/sns/types"  
)  
  
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)  
// actions  
// used in the examples.  
type SnsActions struct {  
    SnsClient *sns.Client  
}  
  
// CreateTopic creates an Amazon SNS topic with the specified name. You can  
// optionally  
// specify that the topic is created as a FIFO topic and whether it uses content-  
// based  
// deduplication instead of ID-based deduplication.  
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,  
    isFifoTopic bool, contentBasedDeduplication bool) (string, error) {  
    var topicArn string  
    topicAttributes := map[string]string{}  
    if isFifoTopic {  
        topicAttributes["FifoTopic"] = "true"  
    }  
}
```

```
if contentBasedDeduplication {
    topicAttributes["ContentBasedDeduplication"] = "true"
}
topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
    Name:      aws.String(topicName),
    Attributes: topicAttributes,
})
if err != nil {
    log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
} else {
    topicArn = *topic.TopicArn
}

return topicArn, err
}
```

- APIの詳細については、「AWS SDK for Go API リファレンス」の「[CreateTopic](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
*/
public class CreateTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicName>

            Where:
                topicName - The name of the topic to create (for example,
mytopic).

            "";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicName = args[0];
        System.out.println("Creating a topic with name: " + topicName);
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        String arnVal = createSNSTopic(snsClient, topicName);
        System.out.println("The topic ARN is" + arnVal);
        snsClient.close();
    }

    public static String createSNSTopic(SnsClient snsClient, String topicName) {
        CreateTopicResponse result;
        try {
            CreateTopicRequest request = CreateTopicRequest.builder()
                .name(topicName)
                .build();

            result = snsClient.createTopic(request);
            return result.topicArn();

        } catch (SnsException e) {
```

```
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「[CreateTopic](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
    const response = await snsClient.send(
```

```
        new CreateTopicCommand({ Name: topicName })),
    );
    console.log(response);
    // {
    //   '$metadata': {
    //     httpStatusCode: 200,
    //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
    //     extendedRequestId: undefined,
    //     cfId: undefined,
    //     attempts: 1,
    //     totalRetryDelay: 0
    //   },
    //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
    // }
    return response;
  };
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「[CreateTopic](#)」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 createSNSTopic(topicName: String): String {
    val request =
        CreateTopicRequest {
            name = topicName
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn.toString()
    }
}
```

```
}  
}
```

- APIの詳細については、「AWS SDK for Kotlin API リファレンス」の「[CreateTopic](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';  
  
use Aws\Exception\AwsException;  
use Aws\Sns\SnsClient;  
  
/**  
 * Create a Simple Notification Service topics in your AWS account at the  
 * requested region.  
 *  
 * This code expects that you have AWS credentials set up per:  
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/  
 * guide_credentials.html  
 */  
  
$SnsClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);  
  
$topicname = 'myTopic';  
  
try {
```

```

$result = $SnSClient->createTopic([
    'Name' => $topicname,
]);
var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}

```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for PHP API リファレンス」の「[CreateTopic](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def create_topic(self, name):
        """
        Creates a notification topic.

        :param name: The name of the topic to create.
        :return: The newly created topic.

```

```

"""
try:
    topic = self.sns_resource.create_topic(Name=name)
    logger.info("Created topic %s with ARN %s.", name, topic.arn)
except ClientError:
    logger.exception("Couldn't create topic %s.", name)
    raise
else:
    return topic

```

```

class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def create_topic(
        self,
        topic_name: str,
        is_fifo: bool = False,
        content_based_deduplication: bool = False
    ) -> str:
        """
        Create an SNS topic.

```



```
        :param topic_name: The name of the topic to create.
        :param is_fifo: Whether to create a FIFO topic.
        :param content_based_deduplication: Whether to use content-based
deduplication for FIFO topics.
        :return: The ARN of the created topic.
        :raises ClientError: If the topic creation fails.
        """
    try:
        # Add .fifo suffix for FIFO topics
        if is_fifo and not topic_name.endswith('.fifo'):
            topic_name += '.fifo'

        attributes = {}
        if is_fifo:
            attributes['FifoTopic'] = 'true'
            if content_based_deduplication:
                attributes['ContentBasedDeduplication'] = 'true'

        response = self.sns_client.create_topic(
            Name=topic_name,
            Attributes=attributes
        )

        topic_arn = response['TopicArn']
        logger.info(f"Created topic: {topic_name} with ARN: {topic_arn}")
        return topic_arn

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error creating topic {topic_name}: {error_code} -
{e}")
        raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[CreateTopic](#)」を参照してください。

Ruby

SDK for Ruby

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。

```
# This class demonstrates how to create an Amazon Simple Notification Service
(SNS) topic.
class SNSTopicCreator
  # Initializes an SNS client.
  #
  # Utilizes the default AWS configuration for region and credentials.
  def initialize
    @sns_client = Aws::SNS::Client.new
  end

  # Attempts to create an SNS topic with the specified name.
  #
  # @param topic_name [String] The name of the SNS topic to create.
  # @return [Boolean] true if the topic was successfully created, false
  # otherwise.
  def create_topic(topic_name)
    @sns_client.create_topic(name: topic_name)
    puts "The topic '#{topic_name}' was successfully created."
    true
  rescue Aws::SNS::Errors::ServiceError => e
    # Handles SNS service errors gracefully.
    puts "Error while creating the topic named '#{topic_name}': #{e.message}"
    false
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_name = 'YourTopicName' # Replace with your topic name
  sns_topic_creator = SNSTopicCreator.new

  puts "Creating the topic '#{topic_name}'..."
end
```

```
unless sns_topic_creator.create_topic(topic_name)
  puts 'The topic was not created. Stopping program.'
  exit 1
end
end
```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、「AWS SDK for Ruby API リファレンス」の「[CreateTopic](#)」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 make_topic(client: &Client, topic_name: &str) -> Result<(), Error> {
    let resp = client.create_topic().name(topic_name).send().await?;

    println!(
        "Created topic with ARN: {}",
        resp.topic_arn().unwrap_or_default()
    );

    Ok(())
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「[CreateTopic](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.  
    oo_result = lo_sns->createtopic( iv_name = iv_topic_name ). " oo_result  
is returned for testing purposes. "  
    MESSAGE 'SNS topic created' TYPE 'I'.  
    CATCH /aws1/cx_snstopiclimitexcdex.  
        MESSAGE 'Unable to create more topics. You have reached the maximum  
number of topics allowed.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「[CreateTopic](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS  
  
let config = try await SNSClient.SNSClientConfiguration(region: region)  
let snsClient = SNSClient(config: config)  
  
let output = try await snsClient.createTopic(  
    input: CreateTopicInput(name: name)
```

```
)

guard let arn = output.topicArn else {
    print("No topic ARN returned by Amazon SNS.")
    return
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[CreateTopic](#)」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 での Amazon SNS 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 **DeleteTopic**で を使用する

次のサンプルコードは、DeleteTopic を使用する方法を説明しています。

アクション例は、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。次のコード例で、このアクションのコンテキストを確認できます。

- [メッセージをキューに発行する](#)

.NET

SDK for .NET

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。

トピック ARN でトピックを削除します。

```
/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
```

```

/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}

```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[DeleteTopic](#)」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 an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
 */
bool AwsDoc::SNS::deleteTopic(const Aws::String &topicARN,
                              const Aws::Client::ClientConfiguration
                              &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::DeleteTopicRequest request;
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::DeleteTopicOutcome outcome =
        snsClient.DeleteTopic(request);
}

```

```
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted the Amazon SNS topic " << topicARN <<
std::endl;
    }
    else {
        std::cerr << "Error deleting topic " << topicARN << ":" <<
        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「[DeleteTopic](#)」を参照してください。

CLI

AWS CLI

SNS トピックを削除するには

次の delete-topic の例では、指定した SNS トピックを削除します。

```
aws sns delete-topic \
  --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic"
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[DeleteTopic](#)」を参照してください。

Go

SDK for Go V2

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 (  
    "context"  
    "encoding/json"  
    "log"  
  
    "github.com/aws/aws-sdk-go-v2/aws"  
    "github.com/aws/aws-sdk-go-v2/service/sns"  
    "github.com/aws/aws-sdk-go-v2/service/sns/types"  
)  
  
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)  
actions  
// used in the examples.  
type SnsActions struct {  
    SnsClient *sns.Client  
}  
  
// DeleteTopic delete an Amazon SNS topic.  
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {  
    _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{  
        TopicArn: aws.String(topicArn)})  
    if err != nil {  
        log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)  
    }  
    return err  
}
```

- APIの詳細については、AWS SDK for Go API リファレンスの「[DeleteTopic](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class DeleteTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:      <topicArn>

            Where:
                topicArn - The ARN of the topic to delete.
            "";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
```

```
        .region(Region.US_EAST_1)
        .build();

        System.out.println("Deleting a topic with name: " + topicArn);
        deleteSNSTopic(snsClient, topicArn);
        snsClient.close();
    }

    public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
        try {
            DeleteTopicRequest request = DeleteTopicRequest.builder()
                .topicArn(topicArn)
                .build();

            DeleteTopicResponse result = snsClient.deleteTopic(request);
            System.out.println("\n\nStatus was " +
                result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[DeleteTopic](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave
it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[DeleteTopic](#)」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 deleteSNSTopic(topicArnVal: String) {
    val request =
        DeleteTopicRequest {
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was successfully deleted.")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[DeleteTopic](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;
```

```
/**
 * Deletes an SNS topic and all its subscriptions.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->deleteTopic([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- APIの詳細については、AWS SDK for PHP API リファレンスの「[DeleteTopic](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def delete_topic(topic):
        """
        Deletes a topic. All subscriptions to the topic are also deleted.
        """
        try:
            topic.delete()
            logger.info("Deleted topic %s.", topic.arn)
        except ClientError:
            logger.exception("Couldn't delete topic %s.", topic.arn)
            raise

```

```

class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')

```

```
return cls(sns_client)

def delete_topic(self, topic_arn: str) -> bool:
    """
    Delete an SNS topic.

    :param topic_arn: The ARN of the topic to delete.
    :return: True if successful.
    :raises ClientError: If the topic deletion fails.
    """
    try:
        self.sns_client.delete_topic(TopicArn=topic_arn)

        logger.info(f"Deleted topic: {topic_arn}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')

        if error_code == 'NotFound':
            logger.warning(f"Topic not found: {topic_arn}")
            return True # Already deleted
        else:
            logger.error(f"Error deleting topic: {error_code} - {e}")
            raise
```

- APIの詳細については、AWS SDK for Python (Boto3) API リファレンスの「[DeleteTopic](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.  
    lo_sns->deletetopic( iv_topicarn = iv_topic_arn ).  
    MESSAGE 'SNS topic deleted.' TYPE 'I'.  
CATCH /aws1/cx_snsnotfoundexception.  
    MESSAGE 'Topic does not exist.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[DeleteTopic](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS  
  
let config = try await SNSClient.SNSClientConfiguration(region: region)  
let snsClient = SNSClient(config: config)  
  
_ = try await snsClient.deleteTopic(  
    input: DeleteTopicInput(topicArn: arn)  
)
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[DeleteTopic](#)」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 での Amazon SNS 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 **GetSMSAttributes**で を使用する

次のサンプルコードは、GetSMSAttributes 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。

```
//! Retrieve the default settings for sending SMS messages from your AWS account
    by using
    //! Amazon Simple Notification Service (Amazon SNS).
    /*!
        \param clientConfiguration: AWS client configuration.
        \return bool: Function succeeded.
    */
    bool
    AwsDoc::SNS::getSMSType(const Aws::Client::ClientConfiguration
        &clientConfiguration) {
        Aws::SNS::SNSClient snsClient(clientConfiguration);

        Aws::SNS::Model::GetSMSAttributesRequest request;
        //Set the request to only retrieve the DefaultSMSType setting.
        //Without the following line, GetSMSAttributes would retrieve all settings.
        request.AddAttributes("DefaultSMSType");

        const Aws::SNS::Model::GetSMSAttributesOutcome outcome =
            snsClient.GetSMSAttributes(
                request);

        if (outcome.IsSuccess()) {
            const Aws::Map<Aws::String, Aws::String> attributes =
                outcome.GetResult().GetAttributes();
            if (!attributes.empty()) {
                for (auto const &att: attributes) {
                    std::cout << att.first << ": " << att.second << std::endl;
                }
            }
        }
    }
```

```
    }
    else {
        std::cout
            << "AwsDoc::SNS::getSMSType - an empty map of attributes was
retrieved."
            << std::endl;
    }
}
else {
    std::cerr << "Error while getting SMS Type: '"
        << 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「[GetSMSAttributes](#)」を参照してください。

CLI

AWS CLI

デフォルトの SMS メッセージ属性を一覧表示するには

次の `get-sms-attributes` の例では、SMS メッセージを送信するためのデフォルト属性を一覧表示しています。

```
aws sns get-sms-attributes
```

出力:

```
{
  "attributes": {
    "DefaultSenderId": "MyName"
  }
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[GetSMSAttributes](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import
    software.amazon.awssdk.services.sns.model.GetSubscriptionAttributesRequest;
import
    software.amazon.awssdk.services.sns.model.GetSubscriptionAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.Iterator;
import java.util.Map;

/**
 * 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
 */
public class GetSMSAttributes {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicArn>

            Where:
                topicArn - The ARN of the topic from which to retrieve
attributes.

            """;

        if (args.length != 1) {
            System.out.println(usage);
        }
    }
}
```

```
        System.exit(1);
    }

    String topicArn = args[0];
    SnsClient snsClient = SnsClient.builder()
        .region(Region.US_EAST_1)
        .build();

    getSNSAttrutes(snsClient, topicArn);
    snsClient.close();
}

public static void getSNSAttrutes(SnsClient snsClient, String topicArn) {
    try {
        GetSubscriptionAttributesRequest request =
        GetSubscriptionAttributesRequest.builder()
            .subscriptionArn(topicArn)
            .build();

        // Get the Subscription attributes
        GetSubscriptionAttributesResponse res =
        snsClient.getSubscriptionAttributes(request);
        Map<String, String> map = res.attributes();

        // Iterate through the map
        Iterator iter = map.entrySet().iterator();
        while (iter.hasNext()) {
            Map.Entry entry = (Map.Entry) iter.next();
            System.out.println("[Key] : " + entry.getKey() + " [Value] : " +
            entry.getValue());
        }

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }

    System.out.println("\n\nStatus was good");
}
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[GetSMSAttributes](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
```

```
//      extendedRequestId: undefined,
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    attributes: { DefaultSMSType: 'Transactional' }
//  }
return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[GetSMSAttributes](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Get the type of SMS Message sent by default from the AWS SNS service.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnsClient = new SnsClient([
    'profile' => 'default',
```

```
'region' => 'us-east-1',
'version' => '2010-03-31'
]);

try {
    $result = $SnSClient->getSMSAttributes([
        'attributes' => ['DefaultSMSType'],
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[GetSMSAttributes](#)」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 での Amazon SNS 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 **GetTopicAttributes**で を使用する

次のサンプルコードは、GetTopicAttributes を使用する方法を説明しています。

.NET

SDK for .NET

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。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

```

using Amazon.SimpleNotificationService;

/// <summary>
/// This example shows how to retrieve the attributes of an Amazon Simple
/// Notification Service (Amazon SNS) topic.
/// </summary>
public class GetTopicAttributes
{
    public static async Task Main()
    {
        string topicArn = "arn:aws:sns:us-
west-2:000000000000:ExampleSNSTopic";
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        var attributes = await GetTopicAttributesAsync(client, topicArn);
        DisplayTopicAttributes(attributes);
    }

    /// <summary>
    /// Given the ARN of the Amazon SNS topic, this method retrieves the
topic
    /// attributes.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to retrieve the attributes for the Amazon SNS topic.</param>
    /// <param name="topicArn">The ARN of the topic for which to retrieve
    /// the attributes.</param>
    /// <returns>A Dictionary of topic attributes.</returns>
    public static async Task<Dictionary<string, string>>
GetTopicAttributesAsync(
        IAmazonSimpleNotificationService client,
        string topicArn)
    {
        var response = await client.GetTopicAttributesAsync(topicArn);

        return response.Attributes;
    }

    /// <summary>
    /// This method displays the attributes for an Amazon SNS topic.
    /// </summary>
    /// <param name="topicAttributes">A Dictionary containing the
    /// attributes for an Amazon SNS topic.</param>

```



```
public static void DisplayTopicAttributes(Dictionary<string, string>
topicAttributes)
{
    foreach (KeyValuePair<string, string> entry in topicAttributes)
    {
        Console.WriteLine($"{entry.Key}: {entry.Value}\n");
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[GetTopicAttributes](#)」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。

```
//! Retrieve the properties of an Amazon Simple Notification Service (Amazon SNS)
topic.
/*!
 \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
 \param clientConfiguration: AWS client configuration.
 \return bool: Function succeeded.
 */
bool AwsDoc::SNS::getTopicAttributes(const Aws::String &topicARN,
                                     const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);
    Aws::SNS::Model::GetTopicAttributesRequest request;
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::GetTopicAttributesOutcome outcome =
    snsClient.GetTopicAttributes(
```

```

        request);

    if (outcome.IsSuccess()) {
        std::cout << "Topic Attributes:" << std::endl;
        for (auto const &attribute: outcome.GetResult().GetAttributes()) {
            std::cout << "    * " << attribute.first << " : " << attribute.second
                << std::endl;
        }
    }
    else {
        std::cerr << "Error while getting Topic attributes "
            << 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「[GetTopicAttributes](#)」を参照してください。

CLI

AWS CLI

トピックの属性を取得するには

次の `get-topic-attributes` の例では、指定したトピックの属性を表示します。

```

aws sns get-topic-attributes \
    --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic"

```

出力:

```

{
  "Attributes": {
    "SubscriptionsConfirmed": "1",
    "DisplayName": "my-topic",
    "SubscriptionsDeleted": "0",
    "EffectiveDeliveryPolicy": "{\"http\":{\"defaultHealthyRetryPolicy\":"
    "\"minDelayTarget\":20,\"maxDelayTarget\":20,\"numRetries\":3,

```

```

    \"numMaxDelayRetries\":0,\"numNoDelayRetries\":0,\"numMinDelayRetries\":0,
    \"backoffFunction\": \"linear\"},\"disableSubscriptionOverrides\":false}}\",
    \"Owner\": \"123456789012\",
    \"Policy\": \"{\\\"Version\\\":\\\"2008-10-17\\\",\\\"Id\\\":\\\"__default_policy_ID\\\",\\\"Statement\\\":[\\\"Sid\\\":\\\"__default_statement_ID\\\",\\\"Effect\\\":\\\"Allow\\\",\\\"Principal\\\":{\\\"AWS\\\":\\\"*\\\"},\\\"Action\\\":[\\\"SNS:Subscribe\\\",\\\"SNS:ListSubscriptionsByTopic\\\",\\\"SNS>DeleteTopic\\\",\\\"SNS:GetTopicAttributes\\\",\\\"SNS:Publish\\\",\\\"SNS:RemovePermission\\\",\\\"SNS:AddPermission\\\",\\\"SNS:SetTopicAttributes\\\"],\\\"Resource\\\":\\\"arn:aws:sns:us-west-2:123456789012:my-topic\\\",\\\"Condition\\\":{\\\"StringEquals\\\":{\\\"AWS:SourceOwner\\\":\\\"0123456789012\\\"}}]}]}\",
    \"TopicArn\": \"arn:aws:sns:us-west-2:123456789012:my-topic\",
    \"SubscriptionsPending\": \"0\"
  }
}

```

- APIの詳細については、「AWS CLI コマンドリファレンス」の「[GetTopicAttributes](#)」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。

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.GetTopicAttributesRequest;
import software.amazon.awssdk.services.sns.model.GetTopicAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
*/
public class GetTopicAttributes {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicArn>

            Where:
                topicArn - The ARN of the topic to look up.
            "";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        System.out.println("Getting attributes for a topic with name: " +
            topicArn);
        getSNSTopicAttributes(snsClient, topicArn);
        snsClient.close();
    }

    public static void getSNSTopicAttributes(SnsClient snsClient, String
        topicArn) {
        try {
            GetTopicAttributesRequest request =
                GetTopicAttributesRequest.builder()
                    .topicArn(topicArn)
                    .build();

            GetTopicAttributesResponse result =
                snsClient.getTopicAttributes(request);
            System.out.println("\n\nStatus is " +
                result.sdkHttpResponse().statusCode() + "\n\nAttributes: \n\n"
                    + result.attributes());

        } catch (SnsException e) {
```

```

        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[GetTopicAttributes](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```

import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});

```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```

import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
    const response = await snsClient.send(

```


SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set region
AWS.config.update({ region: "REGION" });

// Create promise and SNS service object
var getTopicAttribsPromise = new AWS.SNS({ apiVersion: "2010-03-31" })
    .getTopicAttributes({ TopicArn: "TOPIC_ARN" })
    .promise();

// Handle promise's fulfilled/rejected states
getTopicAttribsPromise
    .then(function (data) {
        console.log(data);
    })
    .catch(function (err) {
        console.error(err, err.stack);
    });
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[GetTopicAttributes](#)」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 getSnsTopicAttributes(topicArnVal: String) {
    val request =
        GetTopicAttributesRequest {
            topicArn = topicArnVal
        }
```

```
    }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.getTopicAttributes(request)
        println("${result.attributes}")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[GetTopicAttributes](#)」を参照してください。

PHP

SDK for PHP

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。

```
$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->getTopicAttributes([
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```


- API の詳細については、AWS SDK for PHP API リファレンスの「[GetTopicAttributes](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.  
    oo_result = lo_sns->gettopicattributes( iv_topicarn = iv_topic_arn ). "  
oo_result is returned for testing purposes. "  
    DATA(lt_attributes) = oo_result->get_attributes( ).  
    MESSAGE 'Retrieved attributes/properties of a topic.' TYPE 'I'.  
CATCH /aws1/cx_snsnotfoundexception.  
    MESSAGE 'Topic does not exist.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[GetTopicAttributes](#)」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 での Amazon SNS 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 **ListPhoneNumbersOptedOut**で 使用する

次のサンプルコードは、ListPhoneNumbersOptedOut を使用する方法を説明しています。

CLI

AWS CLI

SMS メッセージのオプトアウトを一覧表示するには

次の `list-phone-numbers-opted-out` の例では、SMS メッセージの受信をオプトアウトした電話番号を一覧表示しています。

```
aws sns list-phone-numbers-opted-out
```

出力:

```
{
  "phoneNumbers": [
    "+15555550100"
  ]
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ListPhoneNumbersOptedOut](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListPhoneNumbersOptedOutRequest;
import
  software.amazon.awssdk.services.sns.model.ListPhoneNumbersOptedOutResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
*/
public class ListOptOut {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listOpts(snsClient);
        snsClient.close();
    }

    public static void listOpts(SnsClient snsClient) {
        try {
            ListPhoneNumbersOptedOutRequest request =
ListPhoneNumbersOptedOutRequest.builder().build();
            ListPhoneNumbersOptedOutResponse result =
snsClient.listPhoneNumbersOptedOut(request);
            System.out.println("Status is " +
result.sdkHttpResponse().statusCode() + "\n\nPhone Numbers: \n\n"
+ result.phoneNumbers());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[ListPhoneNumbersOptedOut](#)」を参照してください。

PHP

SDK for PHP

 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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Returns a list of phone numbers that are opted out of receiving SMS messages
 * from your AWS SNS account.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSClient->listPhoneNumbersOptedOut();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。

- API の詳細については、AWS SDK for PHP API リファレンスの「[ListPhoneNumbersOptedOut](#)」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 での Amazon SNS 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 **ListSubscriptions**で を使用する

次のサンプルコードは、ListSubscriptions を使用する方法を説明しています。

.NET

SDK for .NET

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。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example will retrieve a list of the existing Amazon Simple
/// Notification Service (Amazon SNS) subscriptions.
/// </summary>
public class ListSubscriptions
{
    public static async Task Main()
    {
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        Console.WriteLine("Enter a topic ARN to list subscriptions for a
specific topic, " +
```

```

        "or press Enter to list subscriptions for all
topics.");
        var topicArn = Console.ReadLine();
        Console.WriteLine();

        var subscriptions = await GetSubscriptionsListAsync(client,
topicArn);

        DisplaySubscriptionList(subscriptions);
    }

    /// <summary>
    /// Gets a list of the existing Amazon SNS subscriptions, optionally by
specifying a topic ARN.
    /// </summary>
    /// <param name="client">The initialized Amazon SNS client object used
    /// to obtain the list of subscriptions.</param>
    /// <param name="topicArn">The optional ARN of a specific topic. Defaults
to null.</param>
    /// <returns>A list containing information about each subscription.</
returns>
    public static async Task<List<Subscription>>
GetSubscriptionsListAsync(IAmazonSimpleNotificationService client, string
topicArn = null)
    {
        var results = new List<Subscription>();

        if (!string.IsNullOrEmpty(topicArn))
        {
            var paginateByTopic = client.Paginators.ListSubscriptionsByTopic(
                new ListSubscriptionsByTopicRequest()
                {
                    TopicArn = topicArn,
                });

            // Get the entire list using the paginator.
            await foreach (var subscription in paginateByTopic.Subscriptions)
            {
                results.Add(subscription);
            }
        }
        else
        {

```

```
        var paginateAllSubscriptions =
client.Paginators.ListSubscriptions(new ListSubscriptionsRequest());

        // Get the entire list using the paginator.
        await foreach (var subscription in
paginateAllSubscriptions.Subscriptions)
        {
            results.Add(subscription);
        }

        return results;
    }

    /// <summary>
    /// Display a list of Amazon SNS subscription information.
    /// </summary>
    /// <param name="subscriptionList">A list containing details for existing
    /// Amazon SNS subscriptions.</param>
    public static void DisplaySubscriptionList(List<Subscription>
subscriptionList)
    {
        foreach (var subscription in subscriptionList)
        {
            Console.WriteLine($"Owner: {subscription.Owner}");
            Console.WriteLine($"Subscription ARN:
{subscription.SubscriptionArn}");
            Console.WriteLine($"Topic ARN: {subscription.TopicArn}");
            Console.WriteLine($"Endpoint: {subscription.Endpoint}");
            Console.WriteLine($"Protocol: {subscription.Protocol}");
            Console.WriteLine();
        }
    }
}
```

- APIの詳細については、AWS SDK for .NET API リファレンスの「[ListSubscriptions](#)」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。

```
//! Retrieve a list of Amazon Simple Notification Service (Amazon SNS)
subscriptions.
/*!
 \param clientConfiguration: AWS client configuration.
 \return bool: Function succeeded.
 */
bool AwsDoc::SNS::listSubscriptions(
    const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::String nextToken; // Next token is used to handle a paginated response.
    bool result = true;
    Aws::Vector<Aws::SNS::Model::Subscription> subscriptions;
    do {
        Aws::SNS::Model::ListSubscriptionsRequest request;

        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        const Aws::SNS::Model::ListSubscriptionsOutcome outcome =
            snsClient.ListSubscriptions(
                request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::SNS::Model::Subscription> &newSubscriptions =
                outcome.GetResult().GetSubscriptions();
            subscriptions.insert(subscriptions.cend(), newSubscriptions.begin(),
                                newSubscriptions.end());
        }
        else {
            std::cerr << "Error listing subscriptions "

```



```
        << outcome.GetError().GetMessage()
        <<
        std::endl;
        result = false;
        break;
    }

    nextToken = outcome.GetResult().GetNextToken();
} while (!nextToken.empty());

if (result) {
    if (subscriptions.empty()) {
        std::cout << "No subscriptions found" << std::endl;
    }
    else {
        std::cout << "Subscriptions list:" << std::endl;
        for (auto const &subscription: subscriptions) {
            std::cout << "    * " << subscription.GetSubscriptionArn() <<
std::endl;
        }
    }
}
return result;
}
```

- API の詳細については、AWS SDK for C++ API リファレンスの「[ListSubscriptions](#)」を参照してください。

CLI

AWS CLI

SNS サブスクリプションを一覧表示するには

次の `list-subscriptions` 例では、AWS アカウントの SNS サブスクリプションのリストを表示します。

```
aws sns list-subscriptions
```

出力:

```
{
  "Subscriptions": [
    {
      "Owner": "123456789012",
      "Endpoint": "my-email@example.com",
      "Protocol": "email",
      "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic",
      "SubscriptionArn": "arn:aws:sns:us-west-2:123456789012:my-topic:8a21d249-4329-4871-acc6-7be709c6ea7f"
    }
  ]
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ListSubscriptions](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListSubscriptionsRequest;
import software.amazon.awssdk.services.sns.model.ListSubscriptionsResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
```

```
public class ListSubscriptions {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listSNSSubscriptions(snsClient);
        snsClient.close();
    }

    public static void listSNSSubscriptions(SnsClient snsClient) {
        try {
            ListSubscriptionsRequest request = ListSubscriptionsRequest.builder()
                .build();

            ListSubscriptionsResponse result =
snsClient.listSubscriptions(request);
            System.out.println(result.subscriptions());

        } catch (SnsException e) {

            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[ListSubscriptions](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 * subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyl@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
}
```

```
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[ListSubscriptions](#)」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 listSNSSubscriptions() {  
    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->  
        val response = snsClient.listSubscriptions(ListSubscriptionsRequest {})  
        response.subscriptions?.forEach { sub ->  
            println("Sub ARN is ${sub.subscriptionArn}")  
            println("Sub protocol is ${sub.protocol}")  
        }  
    }  
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[ListSubscriptions](#)」を参照してください。

PHP

SDK for PHP

 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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Returns a list of Amazon SNS subscriptions in the requested region.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSClient->listSubscriptions();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- API の詳細については、AWS SDK for PHP API リファレンスの「[ListSubscriptions](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def list_subscriptions(self, topic=None):
        """
        Lists subscriptions for the current account, optionally limited to a
        specific topic.

        :param topic: When specified, only subscriptions to this topic are
        returned.
        :return: An iterator that yields the subscriptions.
        """
        try:
            if topic is None:
                subs_iter = self.sns_resource.subscriptions.all()
            else:
                subs_iter = topic.subscriptions.all()
            logger.info("Got subscriptions.")
        except ClientError:
            logger.exception("Couldn't get subscriptions.")
            raise
        else:
            return subs_iter
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[ListSubscriptions](#)」を参照してください。

Ruby

SDK for Ruby

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。

```
# This class demonstrates how to list subscriptions to an Amazon Simple
# Notification Service (SNS) topic
class SnsSubscriptionLister
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Lists subscriptions for a given SNS topic
  # @param topic_arn [String] The ARN of the SNS topic
  # @return [Types::ListSubscriptionsResponse] subscriptions: The response object
  def list_subscriptions(topic_arn)
    @logger.info("Listing subscriptions for topic: #{topic_arn}")
    subscriptions = @sns_client.list_subscriptions_by_topic(topic_arn: topic_arn)
    subscriptions.subscriptions.each do |subscription|
      @logger.info("Subscription endpoint: #{subscription.endpoint}")
    end
    subscriptions
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Error listing subscriptions: #{e.message}")
    raise
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  sns_client = Aws::SNS::Client.new
  topic_arn = 'SNS_TOPIC_ARN' # Replace with your SNS topic ARN
```



```

lister = SnsSubscriptionLister.new(sns_client)

begin
  lister.list_subscriptions(topic_arn)
rescue StandardError => e
  puts "Failed to list subscriptions: #{e.message}"
  exit 1
end
end
end

```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for Ruby API リファレンスの「[ListSubscriptions](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```

TRY.
  oo_result = lo_sns->listsuscriptions( ).
  " oo_result is
returned for testing purposes. "
  DATA(lt_subscriptions) = oo_result->get_subscriptions( ).
  MESSAGE 'Retrieved list of subscribers.' TYPE 'I'.
CATCH /aws1/cx_rt_generic.
  MESSAGE 'Unable to list subscribers.' TYPE 'E'.
ENDTRY.

```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[ListSubscriptions](#)」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 での Amazon SNS 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 **ListTopics**で を使用する

次のサンプルコードは、ListTopics を使用する方法を説明しています。

.NET

SDK for .NET

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。

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// Lists the Amazon Simple Notification Service (Amazon SNS)
/// topics for the current account.
/// </summary>
public class ListSNSTopics
{
    public static async Task Main()
    {
        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await GetTopicListAsync(client);
    }

    /// <summary>
    /// Retrieves the list of Amazon SNS topics in groups of up to 100
    /// topics.
    /// </summary>
```

```
/// <param name="client">The initialized Amazon SNS client object used
/// to retrieve the list of topics.</param>
public static async Task
GetTopicListAsync(IAmazonSimpleNotificationService client)
{
    // If there are more than 100 Amazon SNS topics, the call to
    // ListTopicsAsync will return a value to pass to the
    // method to retrieve the next 100 (or less) topics.
    string nextToken = string.Empty;

    do
    {
        var response = await client.ListTopicsAsync(nextToken);
        DisplayTopicsList(response.Topics);
        nextToken = response.NextToken;
    }
    while (!string.IsNullOrEmpty(nextToken));
}

/// <summary>
/// Displays the list of Amazon SNS Topic ARNs.
/// </summary>
/// <param name="topicList">The list of Topic ARNs.</param>
public static void DisplayTopicsList(List<Topic> topicList)
{
    foreach (var topic in topicList)
    {
        Console.WriteLine($"{topic.TopicArn}");
    }
}
}
```

- APIの詳細については、AWS SDK for .NET API リファレンスの「[ListTopics](#)」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。

```
//! Retrieve a list of Amazon Simple Notification Service (Amazon SNS) topics.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SNS::listTopics(const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::String nextToken; // Next token is used to handle a paginated response.
    bool result = true;
    do {
        Aws::SNS::Model::ListTopicsRequest request;

        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        const Aws::SNS::Model::ListTopicsOutcome outcome = snsClient.ListTopics(
            request);

        if (outcome.IsSuccess()) {
            std::cout << "Topics list:" << std::endl;
            for (auto const &topic: outcome.GetResult().GetTopics()) {
                std::cout << "  * " << topic.GetTopicArn() << std::endl;
            }
        }
        else {
            std::cerr << "Error listing topics " <<
outcome.GetError().GetMessage() <<
                std::endl;
        }
    }
}
```

```
        result = false;
        break;
    }

    nextToken = outcome.GetResult().GetNextToken();
} while (!nextToken.empty());

return result;
}
```

- API の詳細については、AWS SDK for C++ API リファレンスの「[ListTopics](#)」を参照してください。

CLI

AWS CLI

SNS トピックを一覧表示するには

次の `list-topics` 例では、AWS アカウント内のすべての SNS トピックを一覧表示します。

```
aws sns list-topics
```

出力:

```
{
  "Topics": [
    {
      "TopicArn": "arn:aws:sns:us-west-2:123456789012:my-topic"
    }
  ]
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[ListTopics](#)」を参照してください。

Go

SDK for Go V2

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。

```
package main

import (
    "context"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/sns"
    "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
// Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
    ctx := context.Background()
    sdkConfig, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        fmt.Println("Couldn't load default configuration. Have you set up your AWS
account?")
        fmt.Println(err)
        return
    }
    snsClient := sns.NewFromConfig(sdkConfig)
    fmt.Println("Let's list the topics for your account.")
    var topics []types.Topic
    paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
    for paginator.HasMorePages() {
```

```
output, err := paginator.NextPage(ctx)
if err != nil {
    log.Printf("Couldn't get topics. Here's why: %v\n", err)
    break
} else {
    topics = append(topics, output.Topics...)
}
}
if len(topics) == 0 {
    fmt.Println("You don't have any topics!")
} else {
    for _, topic := range topics {
        fmt.Printf("\t%v\n", *topic.TopicArn)
    }
}
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[ListTopics](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.ListTopicsRequest;
import software.amazon.awssdk.services.sns.model.ListTopicsResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
*/
public class ListTopics {
    public static void main(String[] args) {
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listSNSTopics(snsClient);
        snsClient.close();
    }

    public static void listSNSTopics(SnsClient snsClient) {
        try {
            ListTopicsRequest request = ListTopicsRequest.builder()
                .build();

            ListTopicsResponse result = snsClient.listTopics(request);
            System.out.println(
                "Status was " + result.sdkHttpResponse().statusCode() + "\n
\nTopics\n\n" + result.topics());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[ListTopics](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[ListTopics](#)」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 listSNSTopics() {  
    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->  
        val response = snsClient.listTopics(ListTopicsRequest { })  
        response.topics?.forEach { topic ->  
            println("The topic ARN is ${topic.topicArn}")  
        }  
    }  
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[ListTopics](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';
```

```
use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Returns a list of the requester's topics from your AWS SNS account in the
 * region specified.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSclient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

try {
    $result = $SnSclient->listTopics();
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- APIの詳細については、AWS SDK for PHP API リファレンスの「[ListTopics](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def list_topics(self):
        """
        Lists topics for the current account.

        :return: An iterator that yields the topics.
        """
        try:
            topics_iter = self.sns_resource.topics.all()
            logger.info("Got topics.")
        except ClientError:
            logger.exception("Couldn't get topics.")
            raise
        else:
            return topics_iter
```

- APIの詳細については、AWS SDK for Python (Boto3) API リファレンスの「[ListTopics](#)」を参照してください。

Ruby

SDK for Ruby

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。

```
require 'aws-sdk-sns' # v2: require 'aws-sdk'

def list_topics?(sns_client)
  sns_client.topics.each do |topic|
    puts topic.arn
  rescue StandardError => e
    puts "Error while listing the topics: #{e.message}"
  end
end

def run_me
  region = 'REGION'
  sns_client = Aws::SNS::Resource.new(region: region)

  puts 'Listing the topics.'

  return if list_topics?(sns_client)

  puts 'The bucket was not created. Stopping program.'
  exit 1
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for Ruby API リファレンスの「[ListTopics](#)」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_topics(client: &Client) -> Result<(), Error> {
    let resp = client.list_topics().send().await?;

    println!("Topic ARNs:");

    for topic in resp.topics() {
        println!("{}", topic.topic_arn().unwrap_or_default());
    }

    Ok(())
}

```

- API の詳細については、AWS SDK for Rust API リファレンスの「[ListTopics](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```

TRY.
    oo_result = lo_sns->listtopics( ).
    " oo_result is returned for
    testing purposes. "
    DATA(lt_topics) = oo_result->get_topics( ).
    MESSAGE 'Retrieved list of topics.' TYPE 'I'.
CATCH /aws1/cx_rt_generic.
    MESSAGE 'Unable to list topics.' TYPE 'E'.
ENDTRY.

```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[ListTopics](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

var topics: [String] = []
let outputPages = snsClient.listTopicsPaginated(
    input: ListTopicsInput()
)

// Each time a page of results arrives, process its contents.

for try await output in outputPages {
    guard let topicList = output.topics else {
        print("Unable to get a page of Amazon SNS topics.")
        return
    }

    // Iterate over the topics listed on this page, adding their ARNs
    // to the `topics` array.

    for topic in topicList {
        guard let arn = topic.topicArn else {
            print("Topic has no ARN.")
            return
        }
        topics.append(arn)
    }
}
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[ListTopics](#)」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 での Amazon SNS 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 **Publish**で を使用する

次のサンプルコードは、Publish を使用する方法を説明しています。

アクション例は、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。次のコード例で、このアクションのコンテキストを確認できます。

- [FIFO トピックを作成して発行する](#)
- [SMS テキストメッセージを発行する](#)
- [メッセージをキューに発行する](#)

.NET

SDK for .NET

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。

トピックへのメッセージの発行

```
using System;
using System.Threading.Tasks;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

/// <summary>
/// This example publishes a message to an Amazon Simple Notification
/// Service (Amazon SNS) topic.
/// </summary>
```



```
public class PublishToSNSTopic
{
    public static async Task Main()
    {
        string topicArn = "arn:aws:sns:us-
east-2:000000000000:ExampleSNSTopic";
        string messageText = "This is an example message to publish to the
ExampleSNSTopic.";

        IAmazonSimpleNotificationService client = new
AmazonSimpleNotificationServiceClient();

        await PublishToTopicAsync(client, topicArn, messageText);
    }

    /// <summary>
    /// Publishes a message to an Amazon SNS topic.
    /// </summary>
    /// <param name="client">The initialized client object used to publish
    /// to the Amazon SNS topic.</param>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="messageText">The text of the message.</param>
    public static async Task PublishToTopicAsync(
        IAmazonSimpleNotificationService client,
        string topicArn,
        string messageText)
    {
        var request = new PublishRequest
        {
            TopicArn = topicArn,
            Message = messageText,
        };

        var response = await client.PublishAsync(request);

        Console.WriteLine($"Successfully published message ID:
{response.MessageId}");
    }
}
```

グループ、重複、属性のオプションを指定してトピックにメッセージを発行します。

```
/// <summary>
/// Publish messages using user settings.
/// </summary>
/// <returns>Async task.</returns>
public static async Task PublishMessages()
{
    Console.WriteLine("Now we can publish messages.");

    var keepSendingMessages = true;
    string? deduplicationId = null;
    string? toneAttribute = null;
    while (keepSendingMessages)
    {
        Console.WriteLine();
        var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

        if (_useFifoTopic)
        {
            Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                               "\r\nAll messages within the same group will be
received in the order " +
                               "they were published.");

            Console.WriteLine();
            var messageId = GetUserResponse("Enter a message group ID
for this message:", "1");

            if (!_useContentBasedDeduplication)
            {
                Console.WriteLine("Because you are not using content-based
deduplication, " +
                                   "you must enter a deduplication ID.");

                Console.WriteLine("Enter a deduplication ID for this
message.");
                deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
            }

            if (GetYesNoResponse("Add an attribute to this message?"))
            {
```

```

        Console.WriteLine("Enter a number for an attribute.");
        for (int i = 0; i < _tones.Length; i++)
        {
            Console.WriteLine($"{i + 1}. {_tones[i]}");
        }

        var selection = GetUserResponse("", "1");
        int.TryParse(selection, out var selectionNumber);

        if (selectionNumber > 0 && selectionNumber < _tones.Length)
        {
            toneAttribute = _tones[selectionNumber - 1];
        }
    }

    var messageID = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
        messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

keepSendingMessages = GetYesNoResponse("Send another message?",
false);
}
}

```

ユーザーの選択を発行アクションに適用します。

```

    /// <summary>
    /// Publish a message to a topic with an attribute and optional deduplication
    and group IDs.
    /// </summary>
    /// <param name="topicArn">The ARN of the topic.</param>
    /// <param name="message">The message to publish.</param>
    /// <param name="attributeName">The optional attribute for the message.</
param>
    /// <param name="attributeValue">The optional attribute value for the
message.</param>
    /// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
    /// <param name="groupId">The optional group ID for the message.</param>

```

```
/// <returns>The ID of the message published.</returns>
public async Task<string> PublishToTopicWithAttribute(
    string topicArn,
    string message,
    string? attributeName = null,
    string? attributeValue = null,
    string? deduplicationId = null,
    string? groupId = null)
{
    var publishRequest = new PublishRequest()
    {
        TopicArn = topicArn,
        Message = message,
        MessageDeduplicationId = deduplicationId,
        MessageGroupId = groupId
    };

    if (attributeValue != null)
    {
        // Add the string attribute if it exists.
        publishRequest.MessageAttributes =
            new Dictionary<string, MessageAttributeValue>
            {
                { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
            };
    }

    var publishResponse = await
        _amazonSNSClient.PublishAsync(publishRequest);
    return publishResponse.MessageId;
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[発行](#)」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。

```
//! Send a message to an Amazon Simple Notification Service (Amazon SNS) topic.
/*!
    \param message: The message to publish.
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::publishToTopic(const Aws::String &message,
                                const Aws::String &topicARN,
                                const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::PublishRequest request;
    request.SetMessage(message);
    request.SetTopicArn(topicARN);

    const Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Message published successfully with id '"
                    << outcome.GetResult().GetMessageId() << "'." << std::endl;
    }
    else {
        std::cerr << "Error while publishing message "
                    << outcome.GetError().GetMessage()
                    << std::endl;
    }

    return outcome.IsSuccess();
}
```

属性を指定してメッセージを発行します。

```
static const Aws::String TONE_ATTRIBUTE("tone");
static const Aws::Vector<Aws::String> TONES = {"cheerful", "funny",
"serious",
"sincere"};

Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::SNS::SNSClient snsClient(clientConfiguration);

Aws::SNS::Model::PublishRequest request;
request.SetTopicArn(topicARN);
Aws::String message = askQuestion("Enter a message text to publish. ");
request.SetMessage(message);

if (filteringMessages && askYesNoQuestion(
    "Add an attribute to this message? (y/n) ")) {
    for (size_t i = 0; i < TONES.size(); ++i) {
        std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
    }
    int selection = askQuestionForIntRange(
        "Enter a number for an attribute. ",
        1, static_cast<int>(TONES.size()));
    Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
    messageAttributeValue.SetDataType("String");
    messageAttributeValue.SetStringValue(TONES[selection - 1]);
    request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
}

Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

if (outcome.IsSuccess()) {
    std::cout << "Your message was successfully published." << std::endl;
}
else {
    std::cerr << "Error with TopicsAndQueues::Publish. "
        << outcome.GetError().GetMessage()
        << std::endl;

    cleanUp(topicARN,
        queueURLS,
```

```
        subscriptionARNS,  
        snsClient,  
        sqsClient);  
  
    return false;  
}
```

- API の詳細については、AWS SDK for C++ API リファレンスの「[Publish](#)」を参照してください。

CLI

AWS CLI

例 1: トピックにメッセージを発行するには

次の publish の例では、指定した Amazon SNS トピックに指定した通知を公開します。メッセージはテキストファイルから取得されたもので、改行を含めることができます。

```
aws sns publish \  
  --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic" \  
  --message file://message.txt
```

message.txt の内容:

```
Hello World  
Second Line
```

出力:

```
{  
  "MessageId": "123a45b6-7890-12c3-45d6-111122223333"  
}
```

例 2: 電話番号に SMS メッセージを公開するには

次の publish の例では、Hello world! メッセージを電話番号 +1-555-555-0100 に公開します。

```
aws sns publish \  
  --message "Hello world!" \  
  --phone-number +1-555-555-0100
```

出力:

```
{  
  "MessageId": "123a45b6-7890-12c3-45d6-333322221111"  
}
```

- APIの詳細については、「AWS CLI コマンドリファレンス」の「[Publish](#)」を参照してください。

Go

SDK for Go V2

 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 (  
  "context"  
  "encoding/json"  
  "log"  
  
  "github.com/aws/aws-sdk-go-v2/aws"  
  "github.com/aws/aws-sdk-go-v2/service/sns"  
  "github.com/aws/aws-sdk-go-v2/service/sns/types"  
)  
  
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)  
// actions  
// used in the examples.  
type SnsActions struct {  
  SnsClient *sns.Client  
}
```



```
// Publish publishes a message to an Amazon SNS topic. The message is then sent
// to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
// group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
// value
// filter attribute can be specified so that the message can be filtered
// according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
string, groupId string, dedupId string, filterKey string, filterValue string)
error {
    publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
aws.String(message)}
    if groupId != "" {
        publishInput.MessageGroupId = aws.String(groupId)
    }
    if dedupId != "" {
        publishInput.MessageDeduplicationId = aws.String(dedupId)
    }
    if filterKey != "" && filterValue != "" {
        publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
            filterKey: {DataType: aws.String("String"), StringValue:
aws.String(filterValue)},
        }
    }
    _, err := actor.SnsClient.Publish(ctx, &publishInput)
    if err != nil {
        log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
err)
    }
    return err
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「[Publish](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class PublishTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <message> <topicArn>

            Where:
                message - The message text to send.
                topicArn - The ARN of the topic to publish.
            "";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String message = args[0];
```

```
String topicArn = args[1];
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();
pubTopic(snsClient, message, topicArn);
snsClient.close();
}

public static void pubTopic(SnsClient snsClient, String message, String
topicArn) {
    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .topicArn(topicArn)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out
            .println(result.messageId() + " Message sent. Status is " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[Publish](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
plain string or an object
 *
 * if you are using the `json`
`MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  // }
```

```
// MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'  
// }  
return response;  
};
```

グループ、重複、属性のオプションを指定してトピックにメッセージを発行します。

```
async publishMessages() {  
  const message = await this.prompter.input({  
    message: MESSAGES.publishMessagePrompt,  
  });  
  
  let groupId;  
  let deduplicationId;  
  let choices;  
  
  if (this.isFifo) {  
    await this.logger.log(MESSAGES.groupIdNotice);  
    groupId = await this.prompter.input({  
      message: MESSAGES.groupIdPrompt,  
    });  
  
    if (this.autoDedup === false) {  
      await this.logger.log(MESSAGES.deduplicationIdNotice);  
      deduplicationId = await this.prompter.input({  
        message: MESSAGES.deduplicationIdPrompt,  
      });  
    }  
  
    choices = await this.prompter.checkbox({  
      message: MESSAGES.messageAttributesPrompt,  
      choices: toneChoices,  
    });  
  }  
  
  await this.snsClient.send(  
    new PublishCommand({  
      TopicArn: this.topicArn,  
      Message: message,  
      ...(groupId  
        ? {  
          MessageGroupId: groupId,  

```

```
        }
        : {}),
...(deduplicationId
  ? {
    MessageDeduplicationId: deduplicationId,
    }
  : {}),
...(choices
  ? {
    MessageAttributes: {
      tone: {
        DataType: "String.Array",
        StringValue: JSON.stringify(choices),
      },
    },
  }
  : {}),
)),
);

const publishAnother = await this.prompter.confirm({
  message: MESSAGES.publishAnother,
});

if (publishAnother) {
  await this.publishMessages();
}
}
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[Publish](#)」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 pubTopic(
    topicArnVal: String,
    messageVal: String,
) {
    val request =
        PublishRequest {
            message = messageVal
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[Publish](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Sends a message to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->publish([
        'Message' => $message,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[Publish](#)」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: この例では、インラインで宣言された単一の MessageAttribute を使用してメッセージを発行しています。

```
Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute
@{'City'=[Amazon.SimpleNotificationService.Model.MessageAttributeValue]@{DataType='String';
StringValue='AnyCity'}}}
```

例 2: この例では、事前に宣言された複数の MessageAttribute を使用してメッセージを発行しています。

```
$cityAttributeValue = New-Object
    Amazon.SimpleNotificationService.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object
    Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute $messageAttributes
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V4)」の「[Publish](#)」を参照してください。

Tools for PowerShell V5

例 1: この例では、インラインで宣言された単一の MessageAttribute を使用してメッセージを発行しています。

```
Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
Message "Hello" -MessageAttribute
```

```
@{'City'=[Amazon.SimpleNotificationService.Model.MessageAttributeValue]@{DataType='String'
StringValue = 'AnyCity'}}
```

例 2: この例では、事前に宣言された複数の MessageAttribute を使用してメッセージを発行しています。

```
$cityAttributeValue = New-Object
    Amazon.SimpleNotificationService.Model.MessageAttributeValue
$cityAttributeValue.DataType = "String"
$cityAttributeValue.StringValue = "AnyCity"

$populationAttributeValue = New-Object
    Amazon.SimpleNotificationService.Model.MessageAttributeValue
$populationAttributeValue.DataType = "Number"
$populationAttributeValue.StringValue = "1250800"

$messageAttributes = New-Object System.Collections.Hashtable
$messageAttributes.Add("City", $cityAttributeValue)
$messageAttributes.Add("Population", $populationAttributeValue)

Publish-SNSMessage -TopicArn "arn:aws:sns:us-west-2:123456789012:my-topic" -
    Message "Hello" -MessageAttribute $messageAttributes
```

- API の詳細については、「AWS Tools for PowerShell コマンドレットリファレンス (V5)」の「[Publish](#)」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。

サブスクリプションが属性に基づいてフィルター処理できるように、属性を含むメッセージを発行します。

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""
```

```

def __init__(self, sns_resource):
    """
    :param sns_resource: A Boto3 Amazon SNS resource.
    """
    self.sns_resource = sns_resource

    @staticmethod
    def publish_message(topic, message, attributes):
        """
        Publishes a message, with attributes, to a topic. Subscriptions can be
        filtered
        based on message attributes so that a subscription receives messages only
        when specified attributes are present.

        :param topic: The topic to publish to.
        :param message: The message to publish.
        :param attributes: The key-value attributes to attach to the message.
        Values
                           must be either `str` or `bytes`.
        :return: The ID of the message.
        """
        try:
            att_dict = {}
            for key, value in attributes.items():
                if isinstance(value, str):
                    att_dict[key] = {"DataType": "String", "StringValue": value}
                elif isinstance(value, bytes):
                    att_dict[key] = {"DataType": "Binary", "BinaryValue": value}
            response = topic.publish(Message=message, MessageAttributes=att_dict)
            message_id = response["MessageId"]
            logger.info(
                "Published message with attributes %s to topic %s.",
                attributes,
                topic.arn,
            )
        except ClientError:
            logger.exception("Couldn't publish message to topic %s.", topic.arn)
            raise
        else:
            return message_id

```

サブスクライバーのプロトコルに基づいて異なる形式のメッセージを発行します。

```
class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def publish_multi_message(
        topic, subject, default_message, sms_message, email_message
    ):
        """
        Publishes a multi-format message to a topic. A multi-format message takes
        different forms based on the protocol of the subscriber. For example,
        an SMS subscriber might receive a short version of the message
        while an email subscriber could receive a longer version.

        :param topic: The topic to publish to.
        :param subject: The subject of the message.
        :param default_message: The default version of the message. This version
        is
                                sent to subscribers that have protocols that are
        not
                                otherwise specified in the structured message.
        :param sms_message: The version of the message sent to SMS subscribers.
        :param email_message: The version of the message sent to email
        subscribers.
        :return: The ID of the message.
        """
        try:
            message = {
                "default": default_message,
                "sms": sms_message,
                "email": email_message,
            }
            response = topic.publish(
```

```

        Message=json.dumps(message), Subject=subject,
        MessageStructure="json"
    )
    message_id = response["MessageId"]
    logger.info("Published multi-format message to topic %s.", topic.arn)
except ClientError:
    logger.exception("Couldn't publish message to topic %s.", topic.arn)
    raise
else:
    return message_id

```

```

class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def publish_message(
        self,
        topic_arn: str,
        message: str,
        tone_attribute: Optional[str] = None,
        deduplication_id: Optional[str] = None,
        message_group_id: Optional[str] = None

```

```
) -> str:
    """
    Publish a message to an SNS topic.

    :param topic_arn: The ARN of the SNS topic.
    :param message: The message content to publish.
    :param tone_attribute: Optional tone attribute for message filtering.
    :param deduplication_id: Optional deduplication ID for FIFO topics.
    :param message_group_id: Optional message group ID for FIFO topics.
    :return: The message ID of the published message.
    :raises ClientError: If the message publication fails.
    """
    try:
        publish_args = {
            'TopicArn': topic_arn,
            'Message': message
        }

        # Add message attributes if tone is specified
        if tone_attribute:
            publish_args['MessageAttributes'] = {
                'tone': {
                    'DataType': 'String',
                    'StringValue': tone_attribute
                }
            }

        # Add FIFO-specific parameters
        if message_group_id:
            publish_args['MessageGroupId'] = message_group_id

        if deduplication_id:
            publish_args['MessageDeduplicationId'] = deduplication_id

        response = self.sns_client.publish(**publish_args)

        message_id = response['MessageId']
        logger.info(f"Published message to topic {topic_arn} with ID: {message_id}")
        return message_id

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
```

```
        logger.error(f"Error publishing message to topic: {error_code} - {e}")
        raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[Publish](#)」を参照してください。

Ruby

SDK for Ruby

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。

```
# Service class for sending messages using Amazon Simple Notification Service (SNS)
class SnsMessageSender
  # Initializes the SnsMessageSender with an SNS client
  #
  # @param sns_client [Aws::SNS::Client] The SNS client
  def initialize(sns_client)
    @sns_client = sns_client
    @logger = Logger.new($stdout)
  end

  # Sends a message to a specified SNS topic
  #
  # @param topic_arn [String] The ARN of the SNS topic
  # @param message [String] The message to send
  # @return [Boolean] true if message was successfully sent, false otherwise
  def send_message(topic_arn, message)
    @sns_client.publish(topic_arn: topic_arn, message: message)
    @logger.info("Message sent successfully to #{topic_arn}.")
    true
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Error while sending the message: #{e.message}")
  end
end
```

```

    false
  end
end

# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_arn = 'SNS_TOPIC_ARN' # Should be replaced with a real topic ARN
  message = 'MESSAGE'         # Should be replaced with the actual message
  content

  sns_client = Aws::SNS::Client.new
  message_sender = SnsMessageSender.new(sns_client)

  @logger.info('Sending message.')
  unless message_sender.send_message(topic_arn, message)
    @logger.error('Message sending failed. Stopping program.')
    exit 1
  end
end
end

```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for Ruby API リファレンスの「[発行](#)」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 subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`", topic_arn);
}

```



```
let rsp = client
    .subscribe()
    .topic_arn(topic_arn)
    .protocol("email")
    .endpoint(email_address)
    .send()
    .await?;

println!("Added a subscription: {:?}", rsp);

let rsp = client
    .publish()
    .topic_arn(topic_arn)
    .message("hello sns!")
    .send()
    .await?;

println!("Published message: {:?}", rsp);

Ok(())
}
```

- APIの詳細については、AWS SDK for Rust API リファレンスの「[発行](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.
    oo_result = lo_sns->publish(
testing purposes. " " oo_result is returned for
    iv_topicarn = iv_topic_arn
```

```

        iv_message = iv_message ).
    MESSAGE 'Message published to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.

```

属性を含むメッセージをトピックに発行します。

```

TRY.
    oo_result = lo_sns->publish(
testing purposes. " oo_result is returned for
        iv_topicarn = iv_topic_arn
        iv_message = iv_message
        it_messageattributes = it_msg_attrs ).
    MESSAGE 'Message with attributes published to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.

```

マルチフォーマットメッセージをトピックに発行します。

```

" Build JSON message structure for multi-format message
DATA(lv_json_message) = |\{ "default": "{ iv_default_message }", "sms":
"{ iv_sms_message }", "email": "{ iv_email_message }" \}|.

TRY.
    oo_result = lo_sns->publish(
testing purposes. " oo_result is returned for
        iv_topicarn = iv_topic_arn
        iv_message = lv_json_message
        iv_subject = iv_subject
        iv_messagestructure = 'json' ).
    MESSAGE 'Multi-format message published to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.

```

- APIの詳細については、AWS SDK for SAP ABAP API リファレンスの「[Publish](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

let output = try await snsClient.publish(
    input: PublishInput(
        message: message,
        topicArn: arn
    )
)

guard let messageId = output.messageId else {
    print("No message ID received from Amazon SNS.")
    return
}

print("Published message with ID \(messageId)")
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[Publish](#)」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 での Amazon SNS 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 **SetSMSAttributes**で を使用する

次のサンプルコードは、SetSMSAttributes 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。

Amazon SNS を使用して DefaultSMSType 属性を設定する方法。

```
#!/ Set the default settings for sending SMS messages.
/*!
  \param smsType: The type of SMS message that you will send by default.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SNS::setSMSType(const Aws::String &smsType,
                             const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SetSMSAttributesRequest request;
    request.AddAttributes("DefaultSMSType", smsType);

    const Aws::SNS::Model::SetSMSAttributesOutcome outcome =
snsClient.SetSMSAttributes(
    request);

    if (outcome.IsSuccess()) {
        std::cout << "SMS Type set successfully " << std::endl;
    }
    else {
        std::cerr << "Error while setting SMS Type: '"
                    << 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「[SetSMSAttributes](#)」を参照してください。

CLI

AWS CLI

SMS メッセージ属性を設定するには

次の `set-sms-attributes` の例では、SMS メッセージのデフォルトの送信者 ID を `MyName` に設定します。

```
aws sns set-sms-attributes \  
    --attributes DefaultSenderId=MyName
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[SetSMSAttributes](#)」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。

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sns.SnsClient;  
import software.amazon.awssdk.services.sns.model.SetSmsAttributesRequest;  
import software.amazon.awssdk.services.sns.model.SetSmsAttributesResponse;  
import software.amazon.awssdk.services.sns.model.SnsException;  
import java.util.HashMap;  
  
/**  
 * 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
*/
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }

    public static void setSNSAttributes(SnsClient snsClient, HashMap<String,
String> attributes) {
        try {
            SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
                .attributes(attributes)
                .build();

            SetSmsAttributesResponse result =
snsClient.setSMSAttributes(request);
            System.out.println("Set default Attributes to " + attributes + ".
Status was "
                + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[SetSMSAttributes](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional - (Default) Noncritical messages, such as marketing
        // messages.
        // Transactional - Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
```

```
// '$metadata': {  
//   httpStatusCode: 200,  
//   requestId: '1885b977-2d7e-535e-8214-e44be727e265',  
//   extendedRequestId: undefined,  
//   cfId: undefined,  
//   attempts: 1,  
//   totalRetryDelay: 0  
// }  
// }  
return response;  
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[SetSMSAttributes](#)」を参照してください。

PHP

SDK for PHP

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。

```
$SnSClient = new SnsClient([  
    'profile' => 'default',  
    'region' => 'us-east-1',  
    'version' => '2010-03-31'  
]);  
  
try {  
    $result = $SnSClient->SetSMSAttributes([  
        'attributes' => [  
            'DefaultSMSType' => 'Transactional',  
        ],  
    ]);  
    var_dump($result);  
} catch (AwsException $e) {
```



```
// output error message if fails
error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[SetSMSAttributes](#)」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 での Amazon SNS 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 **SetSubscriptionAttributes**で を使用する

次のサンプルコードは、SetSubscriptionAttributes を使用する方法を説明しています。

CLI

AWS CLI

サブスクリプション属性を設定するには

次の set-subscription-attributes の例では、RawMessageDelivery 属性を SQS サブスクリプションに設定します。

```
aws sns set-subscription-attributes \
  --subscription-arn arn:aws:sns:us-east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \
  --attribute-name RawMessageDelivery \
  --attribute-value true
```

このコマンドでは何も出力されません。

次の set-subscription-attributes の例では、FilterPolicy 属性を SQS サブスクリプションに設定します。

```
aws sns set-subscription-attributes \
  --subscription-arn arn:aws:sns:us-east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \
```

```
--attribute-name FilterPolicy \  
--attribute-value "{ \"anyMandatoryKey\": [\"any\", \"of\", \"these\"] }"
```

このコマンドでは何も出力されません。

次の `set-subscription-attributes` の例では、`FilterPolicy` 属性を SQS サブスクリプションから削除します。

```
aws sns set-subscription-attributes \  
  --subscription-arn arn:aws:sns:us-east-1:123456789012:mytopic:f248de18-2cf6-578c-8592-b6f1eaa877dc \  
  --attribute-name FilterPolicy \  
  --attribute-value "{}"
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[SetSubscriptionAttributes](#)」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。

```
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.sns.SnsClient;  
import software.amazon.awssdk.services.sns.model.SnsException;  
import java.util.ArrayList;  
  
/**  
 * 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
*/
public class UseMessageFilterPolicy {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <subscriptionArn>

            Where:
                subscriptionArn - The ARN of a subscription.

            "";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String subscriptionArn = args[0];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        usePolicy(snsClient, subscriptionArn);
        snsClient.close();
    }

    public static void usePolicy(SnsClient snsClient, String subscriptionArn) {
        try {
            SNSMessageFilterPolicy fp = new SNSMessageFilterPolicy();
            // Add a filter policy attribute with a single value
            fp.addAttribute("store", "example_corp");
            fp.addAttribute("event", "order_placed");

            // Add a prefix attribute
            fp.addAttributePrefix("customer_interests", "bas");

            // Add an anything-but attribute
            fp.addAttributeAnythingBut("customer_interests", "baseball");

            // Add a filter policy attribute with a list of values
            ArrayList<String> attributeValues = new ArrayList<>();
            attributeValues.add("rugby");
```

```

        attributeValues.add("soccer");
        attributeValues.add("hockey");
        fp.addAttribute("customer_interests", attributeValues);

        // Add a numeric attribute
        fp.addAttribute("price_usd", "=", 0);

        // Add a numeric attribute with a range
        fp.addAttributeRange("price_usd", ">", 0, "<=", 100);

        // Apply the filter policy attributes to an Amazon SNS subscription
        fp.apply(snsClient, subscriptionArn);

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[SetSubscriptionAttributes](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """

```

```
self.sns_resource = sns_resource

    @staticmethod
    def add_subscription_filter(subscription, attributes):
        """
        Adds a filter policy to a subscription. A filter policy is a key and a
        list of values that are allowed. When a message is published, it must
        have an
            attribute that passes the filter or it will not be sent to the
        subscription.

        :param subscription: The subscription the filter policy is attached to.
        :param attributes: A dictionary of key-value pairs that define the
        filter.
        """
        try:
            att_policy = {key: [value] for key, value in attributes.items()}
            subscription.set_attributes(
                AttributeName="FilterPolicy",
                AttributeValue=json.dumps(att_policy)
            )
            logger.info("Added filter to subscription %s.", subscription.arn)
        except ClientError:
            logger.exception(
                "Couldn't add filter to subscription %s.", subscription.arn
            )
            raise
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[SetSubscriptionAttributes](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.  
    lo_sns->setsubscriptionattributes(  
        iv_subscriptionarn = iv_subscription_arn  
        iv_attributename   = 'FilterPolicy'  
        iv_attributevalue  = iv_filter_policy ).  
    MESSAGE 'Added filter policy to subscription.' TYPE 'I'.  
CATCH /aws1/cx_snsnotfoundexception.  
    MESSAGE 'Subscription does not exist.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[SetSubscriptionAttributes](#)」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 での Amazon SNS 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 **SetSubscriptionAttributesRedrivePolicy**で を使用する

次の例は、SetSubscriptionAttributesRedrivePolicy を使用する方法を説明しています。

Java

SDK for Java 1.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。

```
// Specify the ARN of the Amazon SNS subscription.
String subscriptionArn =
    "arn:aws:sns:us-east-2:123456789012:MyEndpoint:1234a567-
bc89-012d-3e45-6fg7h890123i";

// Specify the ARN of the Amazon SQS queue to use as a dead-letter queue.
String redrivePolicy =
    "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-2:123456789012:MyDeadLetterQueue\"}";

// Set the specified Amazon SQS queue as a dead-letter queue
// of the specified Amazon SNS subscription by setting the RedrivePolicy
// attribute.
SetSubscriptionAttributesRequest request = new SetSubscriptionAttributesRequest()
    .withSubscriptionArn(subscriptionArn)
    .withAttributeName("RedrivePolicy")
    .withAttributeValue(redrivePolicy);
sns.setSubscriptionAttributes(request);
```

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 での Amazon SNS 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 `SetTopicAttributes`で を使用する

次のサンプルコードは、`SetTopicAttributes` を使用する方法を説明しています。

CLI

AWS CLI

トピックの属性を設定するには

次の `set-topic-attributes` の例では、指定したトピックの `DisplayName` 属性を設定します。

```
aws sns set-topic-attributes \
    --topic-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
    --attribute-name DisplayName \
    --attribute-value MyTopicDisplayName
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[SetTopicAttributes](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetTopicAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetTopicAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class SetTopicAttributes {

    public static void main(String[] args) {
        final String usage = ""

            Usage:    <attribute> <topicArn> <value>

            Where:
                attribute - The attribute action to use. Valid parameters are:
                Policy | DisplayName | DeliveryPolicy .
                topicArn - The ARN of the topic.\s
                value - The value for the attribute.
```



```
        """;

        if (args.length < 3) {
            System.out.println(usage);
            System.exit(1);
        }

        String attribute = args[0];
        String topicArn = args[1];
        String value = args[2];

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        setTopAttr(snsClient, attribute, topicArn, value);
        snsClient.close();
    }

    public static void setTopAttr(SnsClient snsClient, String attribute, String
topicArn, String value) {
        try {
            SetTopicAttributesRequest request =
SetTopicAttributesRequest.builder()
                .attributeName(attribute)
                .attributeValue(value)
                .topicArn(topicArn)
                .build();

            SetTopicAttributesResponse result =
snsClient.setTopicAttributes(request);
            System.out.println(
                "\n\nStatus was " + result.sdkHttpResponse().statusCode() +
"\n\nTopic " + request.topicArn()
                + " updated " + request.attributeName() + " to " +
request.attributeValue());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[SetTopicAttributes](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  ),
```

```
);  
console.log(response);  
// {  
//   '$metadata': {  
//     httpStatusCode: 200,  
//     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',  
//     extendedRequestId: undefined,  
//     cfId: undefined,  
//     attempts: 1,  
//     totalRetryDelay: 0  
//   }  
// }  
return response;  
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[SetTopicAttributes](#)」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 setTopAttr(  
    attribute: String?,  
    topicArnVal: String?,  
    value: String?,  
) {  
    val request =  
        SetTopicAttributesRequest {  
            attributeName = attribute  
            attributeValue = value  
            topicArn = topicArnVal  
        }  
}
```

```
SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
    snsClient.setTopicAttributes(request)
    println("Topic ${request.topicArn} was updated.")
}
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[SetTopicAttributes](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Configure the message delivery status attributes for an Amazon SNS Topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnsClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$attribute = 'Policy | DisplayName | DeliveryPolicy';
```

```
$value = 'First Topic';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->setTopicAttributes([
        'AttributeName' => $attribute,
        'AttributeValue' => $value,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- API の詳細については、AWS SDK for PHP API リファレンスの「[SetTopicAttributes](#)」を参照してください。

Ruby

SDK for Ruby

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。

```
# Service class to enable an SNS resource with a specified policy
class SnsResourceEnabler
  # Initializes the SnsResourceEnabler with an SNS resource client
  #
  # @param sns_resource [Aws::SNS::Resource] The SNS resource client
  def initialize(sns_resource)
    @sns_resource = sns_resource
    @logger = Logger.new($stdout)
  end

  # Sets a policy on a specified SNS topic
```

```

#
# @param topic_arn [String] The ARN of the SNS topic
# @param resource_arn [String] The ARN of the resource to include in the policy
# @param policy_name [String] The name of the policy attribute to set
def enable_resource(topic_arn, resource_arn, policy_name)
  policy = generate_policy(topic_arn, resource_arn)
  topic = @sns_resource.topic(topic_arn)

  topic.set_attributes({
    attribute_name: policy_name,
    attribute_value: policy
  })

  @logger.info("Policy #{policy_name} set successfully for topic
#{topic_arn}.")
  rescue Aws::SNS::Errors::ServiceError => e
    @logger.error("Failed to set policy: #{e.message}")
  end

private

# Generates a policy string with dynamic resource ARNs
#
# @param topic_arn [String] The ARN of the SNS topic
# @param resource_arn [String] The ARN of the resource
# @return [String] The policy as a JSON string
def generate_policy(topic_arn, resource_arn)
  {
    Version: '2008-10-17',
    Id: '__default_policy_ID',
    Statement: [{
      Sid: '__default_statement_ID',
      Effect: 'Allow',
      Principal: { "AWS": '*' },
      Action: ['SNS:Publish'],
      Resource: topic_arn,
      Condition: {
        ArnEquals: {
          "AWS:SourceArn": resource_arn
        }
      }
    }]
  }.to_json
end
end

```

```
# Example usage:
if $PROGRAM_NAME == __FILE__
  topic_arn = 'MY_TOPIC_ARN' # Should be replaced with a real topic ARN
  resource_arn = 'MY_RESOURCE_ARN' # Should be replaced with a real resource ARN
  policy_name = 'POLICY_NAME' # Typically, this is "Policy"

  sns_resource = Aws::SNS::Resource.new
  enabler = SnsResourceEnabler.new(sns_resource)

  enabler.enable_resource(topic_arn, resource_arn, policy_name)
end
```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for Ruby API リファレンスの「[SetTopicAttributes](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.
  lo_sns->settopicattributes(
    iv_topicarn = iv_topic_arn
    iv_attributename = iv_attribute_name
    iv_attributevalue = iv_attribute_value ).
  MESSAGE 'Set/updated SNS topic attributes.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
  MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[SetTopicAttributes](#)」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 での Amazon SNS 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 **Subscribe**で 使用する

次のサンプルコードは、Subscribe を使用する方法を説明しています。

アクション例は、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。次のコード例で、このアクションのコンテキストを確認できます。

- [FIFO トピックを作成して発行する](#)
- [メッセージをキューに発行する](#)

.NET

SDK for .NET

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。

E メールアドレスをトピックにサブスクライブします。

```
/// <summary>
/// Creates a new subscription to a topic.
/// </summary>
/// <param name="client">The initialized Amazon SNS client object, used
/// to create an Amazon SNS subscription.</param>
/// <param name="topicArn">The ARN of the topic to subscribe to.</param>
/// <returns>A SubscribeResponse object which includes the subscription
/// ARN for the new subscription.</returns>
public static async Task<SubscribeResponse> TopicSubscribeAsync(
    IAmazonSimpleNotificationService client,
```



```
        string topicArn)
    {
        SubscribeRequest request = new SubscribeRequest()
        {
            TopicArn = topicArn,
            ReturnSubscriptionArn = true,
            Protocol = "email",
            Endpoint = "recipient@example.com",
        };

        var response = await client.SubscribeAsync(request);

        return response;
    }
```

オプションのフィルターでトピックにキューをサブスクライブします。

```
/// <summary>
/// Subscribe a queue to a topic with optional filters.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <returns>The ARN of the new subscription.</returns>
public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
{
    var subscribeRequest = new SubscribeRequest()
    {
        TopicArn = topicArn,
        Protocol = "sqs",
        Endpoint = queueArn
    };

    if (!string.IsNullOrEmpty(filterPolicy))
    {
        subscribeRequest.Attributes = new Dictionary<string, string>
        { { "FilterPolicy", filterPolicy } };
    }
}
```

```
var subscribeResponse = await
_amazonSNSClient.SubscribeAsync(subscribeRequest);
return subscribeResponse.SubscriptionArn;
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
delivery to an email address.
/*!
\param topicARN: An SNS topic Amazon Resource Name (ARN).
\param emailAddress: An email address.
\param clientConfiguration: AWS client configuration.
\return bool: Function succeeded.
*/
bool AwsDoc::SNS::subscribeEmail(const Aws::String &topicARN,
                                const Aws::String &emailAddress,
                                const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("email");
    request.SetEndpoint(emailAddress);

    const Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);
```

```

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
outcome.GetResult().GetSubscriptionArn()
        << "'" << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
outcome.GetError().GetMessage()
        << std::endl;
    }

    return outcome.IsSuccess();
}

```

モバイルアプリケーションをトピックにサブスクライブします。

```

//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
delivery to a mobile app.
/*!
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
    \param endpointARN: The ARN for a mobile app or device endpoint.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool
AwsDoc::SNS::subscribeApp(const Aws::String &topicARN,
                        const Aws::String &endpointARN,
                        const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("application");
    request.SetEndpoint(endpointARN);

    const Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {

```

```

        std::cout << "Subscribed successfully." << std::endl;
        std::cout << "Subscription ARN '" <<
outcome.GetResult().GetSubscriptionArn()
        << "'" << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
outcome.GetError().GetMessage()
        << std::endl;
    }

    return outcome.IsSuccess();
}

```

Lambda 関数をトピックにサブスクライブします。

```

//! Subscribe to an Amazon Simple Notification Service (Amazon SNS) topic with
delivery to an AWS Lambda function.
/*!
    \param topicARN: The Amazon Resource Name (ARN) for an Amazon SNS topic.
    \param lambdaFunctionARN: The ARN for an AWS Lambda function.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::subscribeLambda(const Aws::String &topicARN,
                                   const Aws::String &lambdaFunctionARN,
                                   const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("lambda");
    request.SetEndpoint(lambdaFunctionARN);

    const Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        std::cout << "Subscribed successfully." << std::endl;
    }
}

```

```

        std::cout << "Subscription ARN '" <<
outcome.GetResult().GetSubscriptionArn()
        << "'" << std::endl;
    }
    else {
        std::cerr << "Error while subscribing " <<
outcome.GetError().GetMessage()
        << std::endl;
    }

    return outcome.IsSuccess();
}

```

SQS キューをトピックにサブスクライブします。

```

Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("sqs");
    request.SetEndpoint(queueARN);

    Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
        std::cout << "The queue '" << queueName
        << "' has been subscribed to the topic '"
        << "'" << topicName << "'" << std::endl;
        std::cout << "with the subscription ARN '" << subscriptionARN <<
        ".'"
        << std::endl;
        subscriptionARNS.push_back(subscriptionARN);
    }
    else {
        std::cerr << "Error with TopicsAndQueues::Subscribe. "

```

```

        << outcome.GetError().GetMessage()
        << std::endl;

        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }

```

トピックにフィルターでサブスクライブします。

```

static const Aws::String TONE_ATTRIBUTE("tone");
static const Aws::Vector<Aws::String> TONES = {"cheerful", "funny",
"serious",
                                                "sincere"};

Aws::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

Aws::SNS::SNSClient snsClient(clientConfiguration);

Aws::SNS::Model::SubscribeRequest request;
request.SetTopicArn(topicARN);
request.SetProtocol("sqs");
request.SetEndpoint(queueARN);
if (isFifoTopic) {
    if (first) {
        std::cout << "Subscriptions to a FIFO topic can have
filters."
                    << std::endl;
        std::cout
            << "If you add a filter to this subscription, then
only the filtered messages "
            << "will be received in the queue." << std::endl;
        std::cout << "For information about message filtering, "
            << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
            << std::endl;
    }
}

```

```

        std::cout << "For this example, you can filter messages by a
\"\"\"
        << TONE_ATTRIBUTE << "\" attribute." << std::endl;
    }

    std::ostringstream ostringstream;
    ostringstream << "Filter messages for \"" << queueName
        << "\"'s subscription to the topic \""
        << topicName << "\"? (y/n)";

    // Add filter if user answers yes.
    if (askYesNoQuestion(ostringstream.str())) {
        Aws::String jsonPolicy = getFilterPolicyFromUser();
        if (!jsonPolicy.empty()) {
            filteringMessages = true;

            std::cout << "This is the filter policy for this
subscription."
                << std::endl;
            std::cout << jsonPolicy << std::endl;

            request.AddAttributes("FilterPolicy", jsonPolicy);
        }
        else {
            std::cout
                << "Because you did not select any attributes, no
filter "
                << "will be added to this subscription." <<
std::endl;
        }
    } // if (isFifoTopic)
    Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

    if (outcome.IsSuccess()) {
        Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
        std::cout << "The queue '" << queueName
            << "' has been subscribed to the topic '"
            << "'" << topicName << "'" << std::endl;
        std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
            << std::endl;
    }

```

```

        subscriptionARNS.push_back(subscriptionARN);
    }
    else {
        std::cerr << "Error with TopicsAndQueues::Subscribe. "
                    << outcome.GetError().GetMessage()
                    << std::endl;

        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }

    //! Routine that lets the user select attributes for a subscription filter
    policy.
    /*!
    \sa getFilterPolicyFromUser()
    \return Aws::String: The filter policy as JSON.
    */
    Aws::String AwsDoc::TopicsAndQueues::getFilterPolicyFromUser() {
        std::cout
            << "You can filter messages by one or more of the following \""
            << TONE_ATTRIBUTE << "\" attributes." << std::endl;

        std::vector<Aws::String> filterSelections;
        int selection;
        do {
            for (size_t j = 0; j < TONES.size(); ++j) {
                std::cout << "  " << (j + 1) << ". " << TONES[j]
                    << std::endl;
            }
            selection = askQuestionForIntRange(
                "Enter a number (or enter zero to stop adding more). ",
                0, static_cast<int>(TONES.size()));

            if (selection != 0) {
                const Aws::String &selectedTone(TONES[selection - 1]);
                // Add the tone to the selection if it is not already added.
                if (std::find(filterSelections.begin(),
                            filterSelections.end(),
                            selectedTone)

```



```

        == filterSelections.end()) {
            filterSelections.push_back(selectedTone);
        }
    }
} while (selection != 0);

Aws::String result;
if (!filterSelections.empty()) {
    std::ostringstream jsonPolicyStream;
    jsonPolicyStream << "{ \"\" << TONE_ATTRIBUTE << "\": [";

    for (size_t j = 0; j < filterSelections.size(); ++j) {
        jsonPolicyStream << "\"" << filterSelections[j] << "\"";
        if (j < filterSelections.size() - 1) {
            jsonPolicyStream << ",";
        }
    }
    jsonPolicyStream << "] }";

    result = jsonPolicyStream.str();
}

return result;
}

```

- API の詳細については、AWS SDK for C++ API リファレンスの「[Subscribe](#)」を参照してください。

CLI

AWS CLI

トピックにサブスクライブするには

次の subscribe コマンドは、指定したトピックに E メールアドレスをサブスクライブします。

```

aws sns subscribe \
    --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic \
    --protocol email \

```

```
--notification-endpoint my-email@example.com
```

出力:

```
{  
  "SubscriptionArn": "pending confirmation"  
}
```

- API の詳細については、「AWS CLI コマンドリファレンス」の「[Subscribe](#)」を参照してください。

Go

SDK for Go V2

 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。

オプションのフィルターでトピックにキューをサブスクライブします。

```
import (  
    "context"  
    "encoding/json"  
    "log"  
  
    "github.com/aws/aws-sdk-go-v2/aws"  
    "github.com/aws/aws-sdk-go-v2/service/sns"  
    "github.com/aws/aws-sdk-go-v2/service/sns/types"  
)  
  
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)  
// actions  
// used in the examples.  
type SnsActions struct {  
    SnsClient *sns.Client  
}
```

```
// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
// an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
// policy
// so that messages are only sent to the queue when the message has the specified
// attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
queueArn string, filterMap map[string][]string) (string, error) {
    var subscriptionArn string
    var attributes map[string]string
    if filterMap != nil {
        filterBytes, err := json.Marshal(filterMap)
        if err != nil {
            log.Printf("Couldn't create filter policy, here's why: %v\n", err)
            return "", err
        }
        attributes = map[string]string{"FilterPolicy": string(filterBytes)}
    }
    output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
        Protocol:      aws.String("sqs"),
        TopicArn:      aws.String(topicArn),
        Attributes:    attributes,
        Endpoint:      aws.String(queueArn),
        ReturnSubscriptionArn: true,
    })
    if err != nil {
        log.Printf("Couldn't subscribe queue %v to topic %v. Here's why: %v\n",
            queueArn, topicArn, err)
    } else {
        subscriptionArn = *output.SubscriptionArn
    }

    return subscriptionArn, err
}
```

- APIの詳細については、AWS SDK for Go API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * 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
 */
public class SubscribeEmail {
    public static void main(String[] args) {
        final String usage = ""
            Usage:      <topicArn> <email>

            Where:
                topicArn - The ARN of the topic to subscribe.
                email - The email address to use.
            "";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
```

```

        String email = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        subEmail(snsClient, topicArn, email);
        snsClient.close();
    }

    public static void subEmail(SnsClient snsClient, String topicArn, String
email) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                .protocol("email")
                .endpoint(email)
                .returnSubscriptionArn(true)
                .topicArn(topicArn)
                .build();

            SubscribeResponse result = snsClient.subscribe(request);
            System.out.println("Subscription ARN: " + result.subscriptionArn() +
"\n\n Status is "
                + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}

```

HTTP エンドポイントをトピックにサブスクライブします。

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * 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
*/
public class SubscribeHTTPS {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicArn> <url>

            Where:
                topicArn - The ARN of the topic to subscribe.
                url - The HTTPS endpoint that you want to receive
notifications.

            """;

        if (args.length < 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String url = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        subHTTPS(snsClient, topicArn, url);
        snsClient.close();
    }

    public static void subHTTPS(SnsClient snsClient, String topicArn, String url)
    {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                .protocol("https")
                .endpoint(url)
                .returnSubscriptionArn(true)
                .topicArn(topicArn)
                .build();

            SubscribeResponse result = snsClient.subscribe(request);
        }
    }
}
```

```

        System.out.println("Subscription ARN is " + result.subscriptionArn()
+ "\n\n Status is "
        + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

Lambda 関数をトピックにサブスクライブします。

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * 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
 */
public class SubscribeLambda {

    public static void main(String[] args) {

        final String usage = ""

            Usage:    <topicArn> <lambdaArn>

            Where:
                topicArn - The ARN of the topic to subscribe.
                lambdaArn - The ARN of an AWS Lambda function.
            "";

        if (args.length != 2) {

```

```
        System.out.println(usage);
        System.exit(1);
    }

    String topicArn = args[0];
    String lambdaArn = args[1];
    SnsClient snsClient = SnsClient.builder()
        .region(Region.US_EAST_1)
        .build();

    String arnValue = subLambda(snsClient, topicArn, lambdaArn);
    System.out.println("Subscription ARN: " + arnValue);
    snsClient.close();
}

public static String subLambda(SnsClient snsClient, String topicArn, String
lambdaArn) {
    try {
        SubscribeRequest request = SubscribeRequest.builder()
            .protocol("lambda")
            .endpoint(lambdaArn)
            .returnSubscriptionArn(true)
            .topicArn(topicArn)
            .build();

        SubscribeResponse result = snsClient.subscribe(request);
        return result.subscriptionArn();

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
}
```

- APIの詳細については、AWS SDK for Java 2.x API リファレンスの「[Subscribe](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm
 * a subscription.
 * @param {string} emailAddress - The email address that is subscribed to the
 * topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
```

```
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'pending confirmation'
// }
};
```

モバイルアプリケーションをトピックにサブスクライブします。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing
 * to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint
 * is created
 *
 * when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
//    requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
//    extendedRequestId: undefined,
//    cfId: undefined,
//    attempts: 1,
//    totalRetryDelay: 0
//  },
//  SubscriptionArn: 'pending confirmation'
// }
return response;
};
```

Lambda 関数をトピックにサブスクライブします。

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing
 * to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  // }
```

```
// SubscriptionArn: 'pending confirmation'
// }
return response;
};
```

SQS キューをトピックにサブスクライブします。

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
  test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

トピックにフィルターでサブスクライブします。

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";
```

```
const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute
      // set to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
  // test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
suspend fun subEmail(
    topicArnVal: String,
    email: String,
): String {
    val request =
        SubscribeRequest {
            protocol = "email"
            endpoint = email
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        return result.subscriptionArn.toString()
    }
}
```

Lambda 関数をトピックにサブスクライブします。

```
suspend fun subLambda(
    topicArnVal: String?,
    lambdaArn: String?,
) {
    val request =
        SubscribeRequest {
            protocol = "lambda"
            endpoint = lambdaArn
            returnSubscriptionArn = true
        }
```

```
        topicArn = topicArnVal
    }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.subscribe(request)
        println(" The subscription Arn is ${result.subscriptionArn}")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[Subscribe](#)」を参照してください。

PHP

SDK for PHP

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。

E メールアドレスをトピックにサブスクライブします。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 * message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnsClient = new SnsClient([
    'profile' => 'default',
```

```
'region' => 'us-east-1',
'version' => '2010-03-31'
]);

$protocol = 'email';
$endpoint = 'sample@example.com';
$topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

try {
    $result = $SnSClient->subscribe([
        'Protocol' => $protocol,
        'Endpoint' => $endpoint,
        'ReturnSubscriptionArn' => true,
        'TopicArn' => $topic,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

HTTP エンドポイントをトピックにサブスクライブします。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Prepares to subscribe an endpoint by sending the endpoint a confirmation
 * message.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
```



```

        'version' => '2010-03-31'
    ]);

    $protocol = 'https';
    $endpoint = 'https://';
    $topic = 'arn:aws:sns:us-east-1:111122223333:MyTopic';

    try {
        $result = $SnSClient->subscribe([
            'Protocol' => $protocol,
            'Endpoint' => $endpoint,
            'ReturnSubscriptionArn' => true,
            'TopicArn' => $topic,
        ]);
        var_dump($result);
    } catch (AwsException $e) {
        // output error message if fails
        error_log($e->getMessage());
    }
}

```

- API の詳細については、AWS SDK for PHP API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```

class SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """

```

```

        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def subscribe(topic, protocol, endpoint):
        """
        Subscribes an endpoint to the topic. Some endpoint types, such as email,
        must be confirmed before their subscriptions are active. When a
        subscription
        is not confirmed, its Amazon Resource Number (ARN) is set to
        'PendingConfirmation'.

        :param topic: The topic to subscribe to.
        :param protocol: The protocol of the endpoint, such as 'sms' or 'email'.
        :param endpoint: The endpoint that receives messages, such as a phone
        number
                        (in E.164 format) for SMS messages, or an email address
        for
                        email messages.
        :return: The newly added subscription.
        """
        try:
            subscription = topic.subscribe(
                Protocol=protocol, Endpoint=endpoint, ReturnSubscriptionArn=True
            )
            logger.info("Subscribed %s %s to topic %s.", protocol, endpoint,
            topic.arn)
        except ClientError:
            logger.exception(
                "Couldn't subscribe %s %s to topic %s.", protocol, endpoint,
            topic.arn
            )
            raise
        else:
            return subscription

```

オプションのフィルターでトピックにキューをサブスクライブします。

```
class SnsWrapper:
```

```
"""Wrapper class for managing Amazon SNS operations."""

def __init__(self, sns_client: Any) -> None:
    """
    Initialize the SnsWrapper.

    :param sns_client: A Boto3 Amazon SNS client.
    """
    self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def subscribe_queue_to_topic(
        self,
        topic_arn: str,
        queue_arn: str,
        filter_policy: Optional[str] = None
    ) -> str:
        """
        Subscribe an SQS queue to an SNS topic.

        :param topic_arn: The ARN of the SNS topic.
        :param queue_arn: The ARN of the SQS queue.
        :param filter_policy: Optional JSON filter policy for message filtering.
        :return: The ARN of the subscription.
        :raises ClientError: If the subscription fails.
        """
        try:
            attributes = {}
            if filter_policy:
                attributes['FilterPolicy'] = filter_policy

            response = self.sns_client.subscribe(
                TopicArn=topic_arn,
                Protocol='sqs',
```

```
        Endpoint=queue_arn,
        Attributes=attributes
    )

    subscription_arn = response['SubscriptionArn']
    logger.info(f"Subscribed queue {queue_arn} to topic {topic_arn}")
    return subscription_arn

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error subscribing queue to topic: {error_code} - {e}")
    raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「[Subscribe](#)」を参照してください。

Ruby

SDK for Ruby

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。

E メールアドレスをトピックにサブスクライブします。

```
require 'aws-sdk-sns'
require 'logger'

# Represents a service for creating subscriptions in Amazon Simple Notification
# Service (SNS)
class SubscriptionService
  # Initializes the SubscriptionService with an SNS client
  #
  # @param sns_client [Aws::SNS::Client] The SNS client
  def initialize(sns_client)
    @sns_client = sns_client
  end
end
```

```
@logger = Logger.new($stdout)
end

# Attempts to create a subscription to a topic
#
# @param topic_arn [String] The ARN of the SNS topic
# @param protocol [String] The subscription protocol (e.g., email)
# @param endpoint [String] The endpoint that receives the notifications (email
address)
# @return [Boolean] true if subscription was successfully created, false
otherwise
def create_subscription(topic_arn, protocol, endpoint)
  @sns_client.subscribe(topic_arn: topic_arn, protocol: protocol, endpoint:
endpoint)
  @logger.info('Subscription created successfully.')
  true
rescue Aws::SNS::Errors::ServiceError => e
  @logger.error("Error while creating the subscription: #{e.message}")
  false
end
end

# Main execution if the script is run directly
if $PROGRAM_NAME == __FILE__
  protocol = 'email'
  endpoint = 'EMAIL_ADDRESS' # Should be replaced with a real email address
  topic_arn = 'TOPIC_ARN'    # Should be replaced with a real topic ARN

  sns_client = Aws::SNS::Client.new
  subscription_service = SubscriptionService.new(sns_client)

  @logger.info('Creating the subscription.')
  unless subscription_service.create_subscription(topic_arn, protocol, endpoint)
    @logger.error('Subscription creation failed. Stopping program.')
    exit 1
  end
end
end
```

- 詳細については、「[AWS SDK for Ruby デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for Ruby API リファレンスの「[Subscribe](#)」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。

E メールアドレスをトピックにサブスクライブします。

```
async fn subscribe_and_publish(
    client: &Client,
    topic_arn: &str,
    email_address: &str,
) -> Result<(), Error> {
    println!("Receiving on topic with ARN: `{}`", topic_arn);

    let rsp = client
        .subscribe()
        .topic_arn(topic_arn)
        .protocol("email")
        .endpoint(email_address)
        .send()
        .await?;

    println!("Added a subscription: {:?}", rsp);

    let rsp = client
        .publish()
        .topic_arn(topic_arn)
        .message("hello sns!")
        .send()
        .await?;

    println!("Published message: {:?}", rsp);

    Ok(())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「[Subscribe](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

E メールアドレスをトピックにサブスクライブします。

```
TRY.  
    oo_result = lo_sns->subscribe(                                "oo_result is  
returned for testing purposes."  
        iv_topicarn = iv_topic_arn  
        iv_protocol = 'email'  
        iv_endpoint = iv_email_address  
        iv_returnsubscriptionarn = abap_true ).  
    MESSAGE 'Email address subscribed to SNS topic.' TYPE 'I'.  
    CATCH /aws1/cx_snsnotfoundexception.  
        MESSAGE 'Topic does not exist.' TYPE 'E'.  
    CATCH /aws1/cx_snssubscriptionlmt00.  
        MESSAGE 'Unable to create subscriptions. You have reached the maximum  
number of subscriptions allowed.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[Subscribe](#)」を参照してください。

Swift

SDK for Swift

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。

E メールアドレスをトピックにサブスクライブします。

```
import AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

let output = try await snsClient.subscribe(
    input: SubscribeInput(
        endpoint: email,
        protocol: "email",
        returnSubscriptionArn: true,
        topicArn: arn
    )
)

guard let subscriptionArn = output.subscriptionArn else {
    print("No subscription ARN received from Amazon SNS.")
    return
}

print("Subscription \(subscriptionArn) created.")
```

SMS で通知を受け取るには、電話番号をトピックに登録します。

```
import AWSSNS

let config = try await SNSClient.SNSClientConfiguration(region: region)
let snsClient = SNSClient(config: config)

let output = try await snsClient.subscribe(
```



```
        input: SubscribeInput(
            endpoint: phone,
            protocol: "sms",
            returnSubscriptionArn: true,
            topicArn: arn
        )
    )

    guard let subscriptionArn = output.subscriptionArn else {
        print("No subscription ARN received from Amazon SNS.")
        return
    }

    print("Subscription \(subscriptionArn) created.")
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[Subscribe](#)」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 での Amazon SNS 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 **TagResource**で を使用する

次のサンプルコードは、TagResource を使用する方法を説明しています。

CLI

AWS CLI

トピックにタグを追加するには

次の tag-resource の例では、指定した Amazon SNS トピックにメタデータタグを追加します。

```
aws sns tag-resource \
  --resource-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
  --tags Key=Team,Value=Alpha
```

このコマンドでは何も出力されません。

- API の詳細については、AWS CLI コマンドリファレンスの「[TagResource](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.Tag;
import software.amazon.awssdk.services.sns.model.TagResourceRequest;
import java.util.ArrayList;
import java.util.List;

/**
 * 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
 */
public class AddTags {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicArn>

            Where:
                topicArn - The ARN of the topic to which tags are added.

        """;
```

```
        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        addTopicTags(snsClient, topicArn);
        snsClient.close();
    }

    public static void addTopicTags(SnsClient snsClient, String topicArn) {
        try {
            Tag tag = Tag.builder()
                .key("Team")
                .value("Development")
                .build();

            Tag tag2 = Tag.builder()
                .key("Environment")
                .value("Gamma")
                .build();

            List<Tag> tagList = new ArrayList<>();
            tagList.add(tag);
            tagList.add(tag2);

            TagResourceRequest tagResourceRequest = TagResourceRequest.builder()
                .resourceArn(topicArn)
                .tags(tagList)
                .build();

            snsClient.tagResource(tagResourceRequest);
            System.out.println("Tags have been added to " + topicArn);

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[TagResource](#)」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 addTopicTags(topicArn: String) {
    val tag =
        Tag {
            key = "Team"
            value = "Development"
        }

    val tag2 =
        Tag {
            key = "Environment"
            value = "Gamma"
        }

    val tagList = mutableListOf<Tag>()
    tagList.add(tag)
    tagList.add(tag2)

    val request =
        TagResourceRequest {
            resourceArn = topicArn
            tags = tagList
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.tagResource(request)
        println("Tags have been added to $topicArn")
    }
}
```

```
}  
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[TagResource](#)」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 での Amazon SNS 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 **Unsubscribe**で 使用する

次のサンプルコードは、Unsubscribe を使用する方法を説明しています。

アクション例は、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。次のコード例で、このアクションのコンテキストを確認できます。

- [メッセージをキューに発行する](#)

.NET

SDK for .NET

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。

サブスクリプション ARN でトピックからサブスクライブを解除します。

```
/// <summary>  
/// Unsubscribe from a topic by a subscription ARN.  
/// </summary>  
/// <param name="subscriptionArn">The ARN of the subscription.</param>  
/// <returns>True if successful.</returns>  
public async Task<bool> UnsubscribeByArn(string subscriptionArn)  
{  
    var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(  

```

```
        new UnsubscribeRequest()
        {
            SubscriptionArn = subscriptionArn
        });
    return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
}
```

- APIの詳細については、「AWS SDK for .NET API リファレンス」の「[Unsubscribe](#)」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 subscription to an Amazon Simple Notification Service (Amazon SNS)
topic.
/*!
    \param subscriptionARN: The Amazon Resource Name (ARN) for an Amazon SNS topic
    subscription.
    \param clientConfiguration: AWS client configuration.
    \return bool: Function succeeded.
*/
bool AwsDoc::SNS::unsubscribe(const Aws::String &subscriptionARN,
                              const Aws::Client::ClientConfiguration
                              &clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::UnsubscribeRequest request;
    request.SetSubscriptionArn(subscriptionARN);

    const Aws::SNS::Model::UnsubscribeOutcome outcome =
    snsClient.Unsubscribe(request);

    if (outcome.IsSuccess()) {
```

```
        std::cout << "Unsubscribed successfully " << std::endl;
    }
    else {
        std::cerr << "Error while unsubscribing " <<
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「[Unsubscribe](#)」を参照してください。

CLI

AWS CLI

トピックからサブスクライブを解除するには

次の unsubscribe の例では、指定したサブスクリプションをトピックから削除します。

```
aws sns unsubscribe \
    --subscription-arn arn:aws:sns:us-west-2:0123456789012:my-
topic:8a21d249-4329-4871-acc6-7be709c6ea7f
```

このコマンドでは何も出力されません。

- API の詳細については、「AWS CLI コマンドリファレンス」の「[Unsubscribe](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;

/**
 * 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
 */
public class Unsubscribe {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <subscriptionArn>

            Where:
                subscriptionArn - The ARN of the subscription to delete.
            "";

        if (args.length < 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String subscriptionArn = args[0];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        unSub(snsClient, subscriptionArn);
        snsClient.close();
    }

    public static void unSub(SnsClient snsClient, String subscriptionArn) {
        try {
            UnsubscribeRequest request = UnsubscribeRequest.builder()
                .subscriptionArn(subscriptionArn)
```



```
        .build();

        UnsubscribeResponse result = snsClient.unsubscribe(request);
        System.out.println("\n\nStatus was " +
            result.sdkHttpResponse().statusCode()
                + "\n\nSubscription was removed for " +
            request.subscriptionArn());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[Unsubscribe](#)」を参照してください。

JavaScript

SDK for JavaScript (v3)

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。

別のモジュールでクライアントを作成し、エクスポートします。

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave
// it blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

SDK モジュールとクライアントモジュールをインポートし、API を呼び出します。

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-
  xxxx-xxxx-xxxxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- 詳細については、「[AWS SDK for JavaScript デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for JavaScript API リファレンスの「[Unsubscribe](#)」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 unSub(subscriptionArnVal: String) {
    val request =
        UnsubscribeRequest {
            subscriptionArn = subscriptionArnVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        snsClient.unsubscribe(request)
        println("Subscription was removed for ${request.subscriptionArn}")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[Unsubscribe](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
```

```
use Aws\Sns\SnsClient;

/**
 * Deletes a subscription to an Amazon SNS topic.
 *
 * This code expects that you have AWS credentials set up per:
 * https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
 * guide_credentials.html
 */

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$subscription = 'arn:aws:sns:us-east-1:111122223333:MySubscription';

try {
    $result = $SnSClient->unsubscribe([
        'SubscriptionArn' => $subscription,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[Unsubscribe](#)」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 SnsWrapper:
    """Encapsulates Amazon SNS topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    @staticmethod
    def delete_subscription(subscription):
        """
        Unsubscribes and deletes a subscription.
        """
        try:
            subscription.delete()
            logger.info("Deleted subscription %s.", subscription.arn)
        except ClientError:
            logger.exception("Couldn't delete subscription %s.",
                              subscription.arn)
            raise
```

```
class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.
```

```
        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')
        return cls(sns_client)

    def unsubscribe(self, subscription_arn: str) -> bool:
        """
        Unsubscribe from an SNS topic.

        :param subscription_arn: The ARN of the subscription to remove.
        :return: True if successful.
        :raises ClientError: If the unsubscribe operation fails.
        """
        try:
            self.sns_client.unsubscribe(SubscriptionArn=subscription_arn)

            logger.info(f"Unsubscribed: {subscription_arn}")
            return True

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')

            if error_code == 'NotFound':
                logger.warning(f"Subscription not found: {subscription_arn}")
                return True # Already unsubscribed
            else:
                logger.error(f"Error unsubscribing: {error_code} - {e}")
                raise
```

- APIの詳細については、AWS SDK for Python (Boto3) API リファレンスの「[Unsubscribe](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
TRY.  
    lo_sns->unsubscribe( iv_subscriptionarn = iv_subscription_arn ).  
    MESSAGE 'Subscription deleted.' TYPE 'I'.  
CATCH /aws1/cx_snsnotfoundexception.  
    MESSAGE 'Subscription does not exist.' TYPE 'E'.  
CATCH /aws1/cx_snsinvalidparameterex.  
    MESSAGE 'Subscription with "PendingConfirmation" status cannot be  
deleted/unsubscribed. Confirm subscription before performing unsubscribe  
operation.' TYPE 'E'.  
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「[Unsubscribe](#)」を参照してください。

Swift

SDK for Swift

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 AWSSNS  
  
let config = try await SNSClient.SNSClientConfiguration(region: region)  
let snsClient = SNSClient(config: config)
```

```
_ = try await snsClient.unsubscribe(
    input: UnsubscribeInput(
        subscriptionArn: arn
    )
)

print("Unsubscribed.")
```

- API の詳細については、「AWS SDK for Swift API リファレンス」の「[Unsubscribe](#)」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 での Amazon SNS 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。

SDK を使用した Amazon SNS のシナリオ AWS SDKs

次のコード例は、AWS SDKs を使用して Amazon SNS で一般的なシナリオを実装する方法を示しています。これらのシナリオでは、Amazon SNS 内で複数の関数を呼び出すか、その他の AWS のサービスと組み合わせることで、特定のタスクを実行する方法を説明しています。各シナリオには、完全なソースコードへのリンクが含まれており、そこからコードの設定方法と実行方法に関する手順を確認できます。

シナリオは、サービスアクションをコンテキストで理解するのに役立つ中級レベルの経験を対象としています。

例

- [DynamoDB テーブルにデータを送信するアプリケーションを構築する](#)
- [メッセージを翻訳する公開およびサブスクリプションアプリケーションを構築する](#)
- [AWS SDK を使用して Amazon SNS プッシュ通知用のプラットフォームエンドポイントを作成する](#)
- [ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成](#)
- [Amazon Textract エクスプローラーアプリケーションを作成する](#)
- [AWS SDK を使用して FIFO Amazon SNS トピックを作成して公開する](#)
- [AWS SDK を使用して Amazon Rekognition でビデオ内の人物とオブジェクトを検出する](#)

- [AWS SDK を使用して Amazon SNS トピックに SMS メッセージを発行する](#)
- [AWS SDK を使用して Amazon S3 で Amazon SNS Amazon S3に大規模なメッセージを発行する](#)
- [AWS SDK を使用して Amazon SNS SMS テキストメッセージを発行する](#)
- [AWS SDK を使用して Amazon SNS メッセージを Amazon SQS キューに発行する](#)
- [API Gateway を使用して Lambda 関数を呼び出す](#)
- [スケジュールされたイベントを使用した Lambda 関数の呼び出し](#)

DynamoDB テーブルにデータを送信するアプリケーションを構築する

次のコード例は、Amazon DynamoDB テーブルにデータを送信し、ユーザーがテーブルを更新したときに通知するアプリケーションを構築する方法を示しています。

Java

SDK for Java 2.x

Amazon DynamoDB Java API を使用してデータを送信し、Amazon Simple Notification Service Java API を使用してテキストメッセージを送信する動的ウェブアプリケーションを作成する方法について説明します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- DynamoDB
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

この例では、ユーザーが Amazon DynamoDB テーブルにデータを送信し、Amazon Simple Notification Service (Amazon SNS) を使用して管理者にテキストメッセージを送信できるようにするアプリを構築する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例は、[AWS SDK for JavaScript v3 デベロッパーガイド](#)でも使用できます。

この例で使用されているサービス

- DynamoDB
- Amazon SNS

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 での Amazon SNS 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。

メッセージを翻訳する公開およびサブスクリプションアプリケーションを構築する

次のコード例は、サブスクリプションと発行機能を持ち、メッセージを翻訳するアプリケーションを作成する方法を示しています。

.NET

SDK for .NET

Amazon Simple Notification Service .NET API を使用して、サブスクリプションおよびパブリッシュ機能を持つウェブアプリケーションを作成する方法を説明します。さらに、このサンプルアプリケーションではメッセージを翻訳します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon SNS
- Amazon Translate

Java

SDK for Java 2.x

Amazon Simple Notification Service Java API を使用して、サブスクリプションおよびパブリッシュ機能を持つウェブアプリケーションを作成する方法を説明します。さらに、このサンプルアプリケーションではメッセージを翻訳します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

Java Async API を使用するサンプルの完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon SNS
- Amazon Translate

Kotlin

SDK for Kotlin

Amazon SNS Kotlin API を使用して、サブスクリプションおよび発行機能を持つアプリケーションを作成する方法を示しています。さらに、このサンプルアプリケーションではメッセージを翻訳します。

詳細なソースコードとウェブアプリケーションを作成する手順については、「[GitHub](#)」の詳しい例を参照してください。

詳細なソースコードと Android ネイティブアプリケーションを作成する手順については、「[GitHub](#)」の詳しい例を参照してください。

この例で使用されているサービス

- Amazon SNS
- Amazon Translate

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 での Amazon SNS 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 を使用して Amazon SNS プッシュ通知用のプラットフォームエンドポイントを作成する

次のコード例は、Amazon SNS プッシュ通知のプラットフォームエンドポイントを作成する方法を示しています。

CLI

AWS CLI

プラットフォームアプリケーションのエンドポイントを作成するには

次の `create-platform-endpoint` の例では、指定したトークンを使用して、指定したプラットフォームアプリケーションのエンドポイントを作成します。

```
aws sns create-platform-endpoint \
  --platform-application-arn arn:aws:sns:us-west-2:123456789012:app/GCM/  
MyApplication \
  --token EXAMPLE12345...
```

出力:

```
{
  "EndpointArn": "arn:aws:sns:us-west-2:1234567890:endpoint/GCM/MyApplication/12345678-abcd-9012-efgh-345678901234"
}
```

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.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreatePlatformEndpointRequest;
import software.amazon.awssdk.services.sns.model.CreatePlatformEndpointResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
*
* In addition, create a platform application using the AWS Management Console.
* See this doc topic:
*
* https://docs.aws.amazon.com/sns/latest/dg/mobile-push-send-register.html
*
* Without the values created by following the previous link, this code examples
* does not work.
*/

public class RegistrationExample {
    public static void main(String[] args) {
        final String usage = ""

            Usage:      <token> <platformApplicationArn>

            Where:
                token - The device token or registration ID of the mobile device.
This is a unique
                identifier provided by the device platform (e.g., Apple Push
Notification Service (APNS) for iOS devices, Firebase Cloud Messaging (FCM)
                for Android devices) when the mobile app is registered to receive
push notifications.

                platformApplicationArn - The ARN value of platform application.
You can get this value from the AWS Management Console.\s

            "";

        if (args.length != 2) {
            System.out.println(usage);
            return;
        }

        String token = args[0];
        String platformApplicationArn = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

```

```
        createEndpoint(snsClient, token, platformApplicationArn);
    }

    public static void createEndpoint(SnsClient snsClient, String token, String
platformApplicationArn) {
        System.out.println("Creating platform endpoint with token " + token);
        try {
            CreatePlatformEndpointRequest endpointRequest =
CreatePlatformEndpointRequest.builder()
                .token(token)
                .platformApplicationArn(platformApplicationArn)
                .build();

            CreatePlatformEndpointResponse response =
snsClient.createPlatformEndpoint(endpointRequest);
            System.out.println("The ARN of the endpoint is " +
response.endpointArn());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }
}
```

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 での Amazon SNS 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。

ユーザーがラベルを使用して写真を管理できる写真アセット管理アプリケーションの作成

以下のコード例は、ユーザーがラベルを使用して写真を管理できるサーバーレスアプリケーションの作成方法を示しています。

.NET

SDK for .NET

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、[GitHub](#) で完全な例を参照してください。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

C++

SDK for C++

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、[GitHub](#) で完全な例を参照してください。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Java

SDK for Java 2.x

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、[GitHub](#) で完全な例を参照してください。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、[GitHub](#) で完全な例を参照してください。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition

- Amazon S3
- Amazon SNS

Kotlin

SDK for Kotlin

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、[GitHub](#) で完全な例を参照してください。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

PHP

SDK for PHP

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、[GitHub](#) で完全な例を参照してください。

この例のソースについて詳しくは、[AWS コミュニティ](#)でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ

- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Rust

SDK for Rust

Amazon Rekognition を使用して画像内のラベルを検出し、保存して後で取得できるようにする写真アセット管理アプリケーションの開発方法を示します。

完全なソースコードと設定および実行の手順については、[GitHub](#) で完全な例を参照してください。

この例のソースについて詳しくは、[AWS コミュニティ](#) でブログ投稿を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

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 での Amazon SNS 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。

Amazon Textract エクスプローラーアプリケーションを作成する

次のコード例は、インタラクティブアプリケーションで Amazon Textract の出力を調べる方法を示しています。

JavaScript

SDK for JavaScript (v3)

を使用して、Amazon Textract を使用してドキュメントイメージからデータを抽出し、インタラクティブなウェブページに表示する React アプリケーション AWS SDK for JavaScript を構築する方法を示します。この例はウェブブラウザで実行され、認証情報に認証された Amazon Cognito ID が必要です。Amazon Simple Storage Service (Amazon S3) をストレージに使用し、通知のために、Amazon Simple Notification Service (Amazon SNS) トピックにサブスクライブした Amazon Simple Queue Service (Amazon SQS) キューをポーリングします。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK for Python (Boto3)

Amazon Textract AWS SDK for Python (Boto3) で を使用して、ドキュメントイメージ内のテキスト、フォーム、テーブル要素を検出する方法を示します。入力イメージと Amazon Textract 出力は、検出された要素を探索できる Tkinter アプリケーションに表示されます。

- Amazon Textract にドキュメントイメージを送信し、検出された要素の出力を調べます。
- Amazon Textract に直接イメージを送信するか、Amazon Simple Storage Service (Amazon S3) バケットを通じてイメージを送信します。
- 非同期 API を使用して、ジョブの完了時に Amazon Simple Notification Service (Amazon SNS) トピックに通知を発行するジョブを開始します。
- Amazon Simple Queue Service (Amazon SQS) キューにジョブ完了メッセージについてポーリングし、結果を表示します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon Cognito ID
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

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 での Amazon SNS 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 を使用して FIFO Amazon SNS トピックを作成して公開する

次のコード例は、FIFO Amazon SNS トピックを作成、発行する方法を示しています。

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。

この例では

- 1 つの Amazon SNS FIFO トピック、2 つの Amazon SQS FIFO キュー、および 1 つの標準キューを作成します。
- キューをトピックにサブスクライブし、メッセージをトピックに発行します。

[テスト](#)では、各キューへのメッセージの受信を検証します。[完全な例](#)では、アクセスポリシーの追加と、最後にリソースの削除も示しています。

```
public class PriceUpdateExample {  
    public final static SnsClient snsClient = SnsClient.create();
```

```

public final static SqsClient sqsClient = SqsClient.create();

public static void main(String[] args) {

    final String usage = "\n" +
        "Usage: " +
        "    <topicName> <wholesaleQueueFifoName> <retailQueueFifoName>
<analyticsQueueName>\n\n" +
        "Where:\n" +
        "    fifoTopicName - The name of the FIFO topic that you want to
create. \n\n" +
        "    wholesaleQueueARN - The name of a SQS FIFO queue that will be
created for the wholesale consumer. \n\n"
        +
        "    retailQueueARN - The name of a SQS FIFO queue that will created
for the retail consumer. \n\n" +
        "    analyticsQueueARN - The name of a SQS standard queue that will be
created for the analytics consumer. \n\n";
    if (args.length != 4) {
        System.out.println(usage);
        System.exit(1);
    }

    final String fifoTopicName = args[0];
    final String wholeSaleQueueName = args[1];
    final String retailQueueName = args[2];
    final String analyticsQueueName = args[3];

    // For convenience, the QueueData class holds metadata about a queue:
    ARN, URL,
    // name and type.
    List<QueueData> queues = List.of(
        new QueueData(wholeSaleQueueName, QueueType.FIFO),
        new QueueData(retailQueueName, QueueType.FIFO),
        new QueueData(analyticsQueueName, QueueType.Standard));

    // Create queues.
    createQueues(queues);

    // Create a topic.
    String topicARN = createFIFOTopic(fifoTopicName);

    // Subscribe each queue to the topic.
    subscribeQueues(queues, topicARN);

```

```
// Allow the newly created topic to send messages to the queues.
addAccessPolicyToQueuesFINAL(queues, topicARN);

// Publish a sample price update message with payload.
publishPriceUpdate(topicARN, "{\"product\": 214, \"price\": 79.99}\",
"Consumables");

// Clean up resources.
deleteSubscriptions(queues);
deleteQueues(queues);
deleteTopic(topicARN);
}

public static String createFIFOTopic(String topicName) {
    try {
        // Create a FIFO topic by using the SNS service client.
        Map<String, String> topicAttributes = Map.of(
            "FifoTopic", "true",
            "ContentBasedDeduplication", "false",
            "FifoThroughputScope", "MessageGroup");

        CreateTopicRequest topicRequest = CreateTopicRequest.builder()
            .name(topicName)
            .attributes(topicAttributes)
            .build();

        CreateTopicResponse response = snsClient.createTopic(topicRequest);
        String topicArn = response.topicArn();
        System.out.println("The topic ARN is" + topicArn);

        return topicArn;

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

public static void subscribeQueues(List<QueueData> queues, String topicARN) {
    queues.forEach(queue -> {
        SubscribeRequest subscribeRequest = SubscribeRequest.builder()
            .topicArn(topicARN)
```

```
        .endpoint(queue.queueARN)
        .protocol("sqs")
        .build();

    // Subscribe to the endpoint by using the SNS service client.
    // Only Amazon SQS queues can receive notifications from an Amazon
SNS FIFO
    // topic.
    SubscribeResponse subscribeResponse =
snsClient.subscribe(subscribeRequest);
    System.out.println("The queue [" + queue.queueARN + "] subscribed to
the topic [" + topicARN + "]);
    queue.subscriptionARN = subscribeResponse.subscriptionArn();
    });
}

    public static void publishPriceUpdate(String topicArn, String payload, String
groupId) {

        try {
            // Create and publish a message that updates the wholesale price.
            String subject = "Price Update";
            String dedupId = UUID.randomUUID().toString();
            String attributeName = "business";
            String attributeValue = "wholesale";

            MessageAttributeValue msgAttValue = MessageAttributeValue.builder()
                .dataType("String")
                .stringValue(attributeValue)
                .build();

            Map<String, MessageAttributeValue> attributes = new HashMap<>();
            attributes.put(attributeName, msgAttValue);
            PublishRequest pubRequest = PublishRequest.builder()
                .topicArn(topicArn)
                .subject(subject)
                .message(payload)
                .messageGroupId(groupId)
                .messageDeduplicationId(dedupId)
                .messageAttributes(attributes)
                .build();

            final PublishResponse response = snsClient.publish(pubRequest);
            System.out.println(response.messageId());
```

```
        System.out.println(response.sequenceNumber());
        System.out.println("Message was published to " + topicArn);

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- APIの詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
 - [CreateTopic](#)
 - [Publish](#)
 - [Subscribe](#)

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。

FIFO トピックを作成し、そのトピックに Amazon SQS FIFO キューと標準キューをサブスクライブして、メッセージを Amazon SNS トピックに発行します。

```
def usage_demo():
    """Shows how to subscribe queues to a FIFO topic."""
    print("-" * 88)
    print("Welcome to the `Subscribe queues to a FIFO topic` demo!")
    print("-" * 88)

    sns = boto3.resource("sns")
    sqs = boto3.resource("sqs")
    fifo_topic_wrapper = FifoTopicWrapper(sns)
    sns_wrapper = SnsWrapper(sns)
```



```
prefix = "sqs-subscribe-demo-"
queues = set()
subscriptions = set()

wholesale_queue = sqs.create_queue(
    QueueName=prefix + "wholesale.fifo",
    Attributes={
        "MaximumMessageSize": str(4096),
        "ReceiveMessageWaitTimeSeconds": str(10),
        "VisibilityTimeout": str(300),
        "FifoQueue": str(True),
        "ContentBasedDeduplication": str(True),
    },
)
queues.add(wholesale_queue)
print(f"Created FIFO queue with URL: {wholesale_queue.url}.")

retail_queue = sqs.create_queue(
    QueueName=prefix + "retail.fifo",
    Attributes={
        "MaximumMessageSize": str(4096),
        "ReceiveMessageWaitTimeSeconds": str(10),
        "VisibilityTimeout": str(300),
        "FifoQueue": str(True),
        "ContentBasedDeduplication": str(True),
    },
)
queues.add(retail_queue)
print(f"Created FIFO queue with URL: {retail_queue.url}.")

analytics_queue = sqs.create_queue(QueueName=prefix + "analytics",
Attributes={})
queues.add(analytics_queue)
print(f"Created standard queue with URL: {analytics_queue.url}.")

topic = fifo_topic_wrapper.create_fifo_topic("price-updates-topic.fifo")
print(f"Created FIFO topic: {topic.attributes['TopicArn']}.")

for q in queues:
    fifo_topic_wrapper.add_access_policy(q, topic.attributes["TopicArn"])

print(f"Added access policies for topic: {topic.attributes['TopicArn']}.")

for q in queues:
```

```

        sub = fifo_topic_wrapper.subscribe_queue_to_topic(
            topic, q.attributes["QueueArn"]
        )
        subscriptions.add(sub)

    print(f"Subscribed queues to topic: {topic.attributes['TopicArn']}.")

    input("Press Enter to publish a message to the topic.")

    message_id = fifo_topic_wrapper.publish_price_update(
        topic, '{"product": 214, "price": 79.99}', "Consumables"
    )

    print(f"Published price update with message ID: {message_id}.")

    # Clean up the subscriptions, queues, and topic.
    input("Press Enter to clean up resources.")
    for s in subscriptions:
        sns_wrapper.delete_subscription(s)

    sns_wrapper.delete_topic(topic)

    for q in queues:
        fifo_topic_wrapper.delete_queue(q)

    print(f"Deleted subscriptions, queues, and topic.")

    print("Thanks for watching!")
    print("-" * 88)

class FifoTopicWrapper:
    """Encapsulates Amazon SNS FIFO topic and subscription functions."""

    def __init__(self, sns_resource):
        """
        :param sns_resource: A Boto3 Amazon SNS resource.
        """
        self.sns_resource = sns_resource

    def create_fifo_topic(self, topic_name):
        """
        Create a FIFO topic.

```

Topic names must be made up of only uppercase and lowercase ASCII letters, numbers, underscores, and hyphens, and must be between 1 and 256 characters long.

For a FIFO topic, the name must end with the `.fifo` suffix.

:param topic_name: The name for the topic.

:return: The new topic.

"""

try:

 topic = self.sns_resource.create_topic(

 Name=topic_name,

 Attributes={

 "FifoTopic": str(True),

 "ContentBasedDeduplication": str(False),

 "FifoThroughputScope": "MessageGroup",

 },

)

 logger.info("Created FIFO topic with name=%s.", topic_name)

 return topic

except ClientError as error:

 logger.exception("Couldn't create topic with name=%s!", topic_name)

 raise error

@staticmethod

def add_access_policy(queue, topic_arn):

 """

 Add the necessary access policy to a queue, so
it can receive messages from a topic.

:param queue: The queue resource.

:param topic_arn: The ARN of the topic.

:return: None.

"""

try:

 queue.set_attributes(

 Attributes={

 "Policy": json.dumps(

 {

 "Version": "2012-10-17",

 "Statement": [

 {

 "Sid": "test-sid",

```

        "Effect": "Allow",
        "Principal": {"AWS": "*"},
        "Action": "SQS:SendMessage",
        "Resource": queue.attributes["QueueArn"],
        "Condition": {
            "ArnLike": {"aws:SourceArn": topic_arn}
        },
    ],
}

)

logger.info("Added trust policy to the queue.")
except ClientError as error:
    logger.exception("Couldn't add trust policy to the queue!")
    raise error


@staticmethod
def subscribe_queue_to_topic(topic, queue_arn):
    """
    Subscribe a queue to a topic.

    :param topic: The topic resource.
    :param queue_arn: The ARN of the queue.
    :return: The subscription resource.
    """
    try:
        subscription = topic.subscribe(
            Protocol="sqs",
            Endpoint=queue_arn,
        )
        logger.info("The queue is subscribed to the topic.")
        return subscription
    except ClientError as error:
        logger.exception("Couldn't subscribe queue to topic!")
        raise error


@staticmethod
def publish_price_update(topic, payload, group_id):
    """
    Compose and publish a message that updates the wholesale price.

```

```

:param topic: The topic to publish to.
:param payload: The message to publish.
:param group_id: The group ID for the message.
:return: The ID of the message.
"""
try:
    att_dict = {"business": {"DataType": "String", "StringValue":
"wholesale"}}
    dedup_id = uuid.uuid4()
    response = topic.publish(
        Subject="Price Update",
        Message=payload,
        MessageAttributes=att_dict,
        MessageGroupId=group_id,
        MessageDeduplicationId=str(dedup_id),
    )
    message_id = response["MessageId"]
    logger.info("Published message to topic %s.", topic.arn)
except ClientError as error:
    logger.exception("Couldn't publish message to topic %s.", topic.arn)
    raise error
return message_id

@staticmethod
def delete_queue(queue):
    """
    Removes an SQS queue. When run against an AWS account, it can take up to
    60 seconds before the queue is actually deleted.

    :param queue: The queue to delete.
    :return: None
    """
    try:
        queue.delete()
        logger.info("Deleted queue with URL=%s.", queue.url)
    except ClientError as error:
        logger.exception("Couldn't delete queue with URL=%s!", queue.url)
        raise error

```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
 - [CreateTopic](#)
 - [Publish](#)
 - [Subscribe](#)

SAP ABAP

SDK for SAP ABAP

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。

FIFO トピックを作成し、そのトピックに Amazon SQS FIFO キューをサブスクライブして、Amazon SNS トピックにメッセージを発行します。

```
" Creates a FIFO topic. "
DATA lt_tpc_attributes TYPE /aws1/
cl_snsstopicattrsm_w=>tt_topicattributesmap.
DATA ls_tpc_attributes TYPE /aws1/
cl_snsstopicattrsm_w=>ts_topicattributesmap_maprow.
  ls_tpc_attributes-key = 'FifoTopic'.
  ls_tpc_attributes-value = NEW /aws1/cl_snsstopicattrsm_w( iv_value =
'true' ).
  INSERT ls_tpc_attributes INTO TABLE lt_tpc_attributes.

TRY.
  DATA(lo_create_result) = lo_sns->createtopic(
    iv_name = iv_topic_name
    it_attributes = lt_tpc_attributes ).
  DATA(lv_topic_arn) = lo_create_result->get_topicarn( ).
  ov_topic_arn = lv_topic_arn.
  "
  ov_topic_arn is returned for testing purposes. "
  MESSAGE 'FIFO topic created' TYPE 'I'.
```

```

    CATCH /aws1/cx_snsopiclimitexcdex.
        MESSAGE 'Unable to create more topics. You have reached the maximum
number of topics allowed.' TYPE 'E'.
    ENDTRY.

    " Subscribes an endpoint to an Amazon Simple Notification Service (Amazon
SNS) topic. "
    " Only Amazon Simple Queue Service (Amazon SQS) FIFO queues can be subscribed
to an SNS FIFO topic. "
    TRY.
        DATA(lo_subscribe_result) = lo_sns->subscribe(
            iv_topicarn = lv_topic_arn
            iv_protocol = 'sqs'
            iv_endpoint = iv_queue_arn ).
        DATA(lv_subscription_arn) = lo_subscribe_result->get_subscriptionarn( ).
        ov_subscription_arn = lv_subscription_arn.
        "
        ov_subscription_arn is returned for testing purposes. "
        MESSAGE 'SQS queue was subscribed to SNS topic.' TYPE 'I'.
    CATCH /aws1/cx_snsnotfoundexception.
        MESSAGE 'Topic does not exist.' TYPE 'E'.
    CATCH /aws1/cx_snssubscriptionlmt00.
        MESSAGE 'Unable to create subscriptions. You have reached the maximum
number of subscriptions allowed.' TYPE 'E'.
    ENDTRY.

    " Publish message to SNS topic. "
    TRY.
        DATA lt_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>tt_messageattributemap.
        DATA ls_msg_attributes TYPE /aws1/
cl_snsmessageattrvalue=>ts_messageattributemap_maprow.
        ls_msg_attributes-key = 'Importance'.
        ls_msg_attributes-value = NEW /aws1/cl_snsmessageattrvalue( iv_datatype =
'String'

iv_stringvalue = 'High' ).
        INSERT ls_msg_attributes INTO TABLE lt_msg_attributes.

        DATA(lo_result) = lo_sns->publish(
            iv_topicarn = lv_topic_arn
            iv_message = 'The price of your mobile plan has been increased from
$19 to $23'
            iv_subject = 'Changes to mobile plan'
            iv_messagegroupid = 'Update-2'

```

```
        iv_messagededuplicationid = 'Update-2.1'
        it_messageattributes = lt_msg_attributes ).
    ov_message_id = lo_result->get_messageid( ).
    ov_message_id is returned for testing purposes. "
    MESSAGE 'Message was published to SNS topic.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Topic does not exist.' TYPE 'E'.
ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の以下のトピックを参照してください。
 - [CreateTopic](#)
 - [Publish](#)
 - [Subscribe](#)

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 での Amazon SNS 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 を使用して Amazon Rekognition でビデオ内の人物とオブジェクトを検出する

次のコード例は、Amazon Rekognition で動画内の人やオブジェクトを検出する方法を示します。

Java

SDK for Java 2.x

Amazon Rekognition Java API を使用し、Amazon Simple Storage Service (Amazon S3) バケットにあるビデオ内で顔やオブジェクトを検出するアプリケーションを作成する方法について説明します。アプリケーションは Amazon Simple Email Service (Amazon SES) を使用して、結果を含む E メール通知を管理者に送信します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

Python

SDK for Python (Boto3)

Amazon Rekognition を使用して、非同期検出ジョブを開始して、動画内の顔、オブジェクト、人を検出します。この例では、ジョブが完了し、Amazon Simple Queue Service (Amazon SQS) キューをトピックにサブスクライブしたときに、Amazon Simple Notification Service (Amazon SNS) トピックに通知するように Amazon Rekognition を設定します。キューがジョブに関するメッセージを受信すると、ジョブが取得され、結果が出力されます。

この例は GitHub で最もよく見られます。完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- Amazon Rekognition
- Amazon S3
- Amazon SES
- Amazon SNS
- Amazon SQS

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 での Amazon SNS 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 を使用して Amazon SNS トピックに SMS メッセージを発行する

次のコードの例は以下の方法を示しています。

- Amazon SNS トピックを作成します。
- 携帯電話番号をトピックにサブスクライブする。
- SMS メッセージをトピックに発行して、登録されているすべての電話番号がメッセージを一度に受信できるようにします。

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。

トピックを作成し、その ARN を返します。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class CreateTopic {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicName>

            Where:
                topicName - The name of the topic to create (for example,
                mytopic).
```

```
        """;

    if (args.length != 1) {
        System.out.println(usage);
        System.exit(1);
    }

    String topicName = args[0];
    System.out.println("Creating a topic with name: " + topicName);
    SnsClient snsClient = SnsClient.builder()
        .region(Region.US_EAST_1)
        .build();

    String arnVal = createSNSTopic(snsClient, topicName);
    System.out.println("The topic ARN is" + arnVal);
    snsClient.close();
}

public static String createSNSTopic(SnsClient snsClient, String topicName) {
    CreateTopicResponse result;
    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
            .name(topicName)
            .build();

        result = snsClient.createTopic(request);
        return result.topicArn();

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
}
```

トピックへのエンドポイントのサブスクライブ。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SnsException;
```

```
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;

/**
 * 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
 */
public class SubscribeTextSMS {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <topicArn> <phoneNumber>

            Where:
                topicArn - The ARN of the topic to subscribe.
                phoneNumber - A mobile phone number that receives
notifications (for example, +1XXX5550100).
            "";

        if (args.length < 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String topicArn = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();

        subTextSNS(snsClient, topicArn, phoneNumber);
        snsClient.close();
    }

    public static void subTextSNS(SnsClient snsClient, String topicArn, String
phoneNumber) {
        try {
            SubscribeRequest request = SubscribeRequest.builder()
                .protocol("sms")
```

```

        .endpoint(phoneNumber)
        .returnSubscriptionArn(true)
        .topicArn(topicArn)
        .build();

        SubscribeResponse result = snsClient.subscribe(request);
        System.out.println("Subscription ARN: " + result.subscriptionArn() +
"\n\n Status is "
        + result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}

```

送信者の ID、上限価格、タイプなど、メッセージ上の属性を設定します。メッセージ属性はオプションです。

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesRequest;
import software.amazon.awssdk.services.sns.model.SetSmsAttributesResponse;
import software.amazon.awssdk.services.sns.model.SnsException;
import java.util.HashMap;

/**
 * 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
 */
public class SetSMSAttributes {
    public static void main(String[] args) {
        HashMap<String, String> attributes = new HashMap<>(1);
        attributes.put("DefaultSMSType", "Transactional");
        attributes.put("UsageReportS3Bucket", "janbucket");
    }
}

```

```

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();
        setSNSAttributes(snsClient, attributes);
        snsClient.close();
    }

    public static void setSNSAttributes(SnsClient snsClient, HashMap<String,
String> attributes) {
        try {
            SetSmsAttributesRequest request = SetSmsAttributesRequest.builder()
                .attributes(attributes)
                .build();

            SetSmsAttributesResponse result =
snsClient.setSMSAttributes(request);
            System.out.println("Set default Attributes to " + attributes + ".
Status was "
                + result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}

```

トピックへのメッセージの発行 メッセージは、すべての受信者に送信されます。

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
*/
public class PublishTextSMS {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <message> <phoneNumber>

            Where:
                message - The message text to send.
                phoneNumber - The mobile phone number to which a message is
sent (for example, +1XXX5550100).\s
            "";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String message = args[0];
        String phoneNumber = args[1];
        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .build();
        pubTextSMS(snsClient, message, phoneNumber);
        snsClient.close();
    }

    public static void pubTextSMS(SnsClient snsClient, String message, String
phoneNumber) {
        try {
            PublishRequest request = PublishRequest.builder()
                .message(message)
                .phoneNumber(phoneNumber)
                .build();

            PublishResponse result = snsClient.publish(request);
            System.out
                .println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
        }
    }
}
```

```
        System.exit(1);
    }
}
}
```

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 での Amazon SNS 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 を使用して Amazon S3 で Amazon SNS Amazon S3に大規模なメッセージを発行する

次のコード例は、Amazon S3 を使用してメッセージペイロードを保存する大きなメッセージを Amazon SNS に発行する方法を示しています。

Java

SDK for Java 1.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。

容量の大きなメッセージを発行するには、Amazon SNS Extended Client Library for Java を使います。送信するメッセージは、実際のメッセージコンテンツを含む Amazon S3 オブジェクトをリファレンスします。

```
import com.amazonaws.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazonaws.sqs.javamessaging.ExtendedClientConfiguration;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.sns.AmazonSNS;
import com.amazonaws.services.sns.AmazonSNSClientBuilder;
import com.amazonaws.services.sns.model.CreateTopicRequest;
import com.amazonaws.services.sns.model.PublishRequest;
```



```
import com.amazonaws.services.sns.model.SetSubscriptionAttributesRequest;
import com.amazonaws.services.sns.util.Topics;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.ReceiveMessageResult;
import software.amazon.sns.AmazonSNSExtendedClient;
import software.amazon.sns.SNSExtendedClientConfiguration;

public class Example {

    public static void main(String[] args) {
        final String BUCKET_NAME = "extended-client-bucket";
        final String TOPIC_NAME = "extended-client-topic";
        final String QUEUE_NAME = "extended-client-queue";
        final Regions region = Regions.DEFAULT_REGION;

        // Message threshold controls the maximum message size that will
        // be allowed to
        // be published
        // through SNS using the extended client. Payload of messages
        // exceeding this
        // value will be stored in
        // S3. The default value of this parameter is 256 KB which is the
        // maximum
        // message size in SNS (and SQS).
        final int EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD = 32;

        // Initialize SNS, SQS and S3 clients
        final AmazonSNS snsClient =
            AmazonSNSClientBuilder.standard().withRegion(region).build();
        final AmazonSQS sqsClient =
            AmazonSQSClientBuilder.standard().withRegion(region).build();
        final AmazonS3 s3Client =
            AmazonS3ClientBuilder.standard().withRegion(region).build();

        // Create bucket, topic, queue and subscription
        s3Client.createBucket(BUCKET_NAME);
        final String topicArn = snsClient.createTopic(
            new
            CreateTopicRequest().withName(TOPIC_NAME)).getTopicArn();
        final String queueUrl = sqsClient.createQueue(
            new
            CreateQueueRequest().withQueueName(QUEUE_NAME)).getQueueUrl();
```

```
        final String subscriptionArn = Topics.subscribeQueue(
            snsClient, sqsClient, topicArn, queueUrl);

        // To read message content stored in S3 transparently through SQS
extended
        // client,
        // set the RawMessageDelivery subscription attribute to TRUE
        final SetSubscriptionAttributesRequest
subscriptionAttributesRequest = new SetSubscriptionAttributesRequest();

subscriptionAttributesRequest.setSubscriptionArn(subscriptionArn);

subscriptionAttributesRequest.setAttributeName("RawMessageDelivery");
        subscriptionAttributesRequest.setAttributeValue("TRUE");

snsClient.setSubscriptionAttributes(subscriptionAttributesRequest);

        // Initialize SNS extended client
        // PayloadSizeThreshold triggers message content storage in S3
when the
        // threshold is exceeded
        // To store all messages content in S3, use AlwaysThroughS3 flag
        final SNSExtendedClientConfiguration
snsExtendedClientConfiguration = new SNSExtendedClientConfiguration()
            .withPayloadSupportEnabled(s3Client, BUCKET_NAME)

        .withPayloadSizeThreshold(EXTENDED_STORAGE_MESSAGE_SIZE_THRESHOLD);
        final AmazonSNSExtendedClient snsExtendedClient = new
AmazonSNSExtendedClient(snsClient,
            snsExtendedClientConfiguration);

        // Publish message via SNS with storage in S3
        final String message = "This message is stored in S3 as it
exceeds the threshold of 32 bytes set above.";
        snsExtendedClient.publish(topicArn, message);

        // Initialize SQS extended client
        final ExtendedClientConfiguration sqsExtendedClientConfiguration
= new ExtendedClientConfiguration()
            .withPayloadSupportEnabled(s3Client,
BUCKET_NAME);
        final AmazonSQSExtendedClient sqsExtendedClient = new
AmazonSQSExtendedClient(sqsClient,
            sqsExtendedClientConfiguration);
```

```
        // Read the message from the queue
        final ReceiveMessageResult result =
sqsExtendedClient.receiveMessage(queueUrl);
        System.out.println("Received message is " +
result.getMessages().get(0).getBody());
    }
}
```

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 での Amazon SNS 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 を使用して Amazon SNS SMS テキストメッセージを発行する

次のコード例は、Amazon SNS を使用して SMS メッセージを発行する方法を示しています。

.NET

SDK for .NET

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。

```
namespace SNSMessageExample
{
    using System;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SimpleNotificationService;
    using Amazon.SimpleNotificationService.Model;

    public class SNSMessage
    {
        private AmazonSimpleNotificationServiceClient snsClient;

        /// <summary>
```

```
/// Initializes a new instance of the <see cref="SNSMessage"/> class.
/// Constructs a new SNSMessage object initializing the Amazon Simple
/// Notification Service (Amazon SNS) client using the supplied
/// Region endpoint.
/// </summary>
/// <param name="regionEndpoint">The Amazon Region endpoint to use in
/// sending test messages with this object.</param>
public SNSMessage(RegionEndpoint regionEndpoint)
{
    snsClient = new
AmazonSimpleNotificationServiceClient(regionEndpoint);
}

/// <summary>
/// Sends the SMS message passed in the text parameter to the phone
number
/// in phoneNum.
/// </summary>
/// <param name="phoneNum">The ten-digit phone number to which the text
/// message will be sent.</param>
/// <param name="text">The text of the message to send.</param>
/// <returns>Async task.</returns>
public async Task SendTextMessageAsync(string phoneNum, string text)
{
    if (string.IsNullOrEmpty(phoneNum) || string.IsNullOrEmpty(text))
    {
        return;
    }

    // Now actually send the message.
    var request = new PublishRequest
    {
        Message = text,
        PhoneNumber = phoneNum,
    };

    try
    {
        var response = await snsClient.PublishAsync(request);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error sending message: {ex}");
    }
}
```

```
    }
  }
}
```

- APIの詳細については、AWS SDK for .NET API リファレンスの「[Publish](#)」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。

```
/**
 * Publish SMS: use Amazon Simple Notification Service (Amazon SNS) to send an
 * SMS text message to a phone number.
 * Note: This requires additional AWS configuration prior to running example.
 *
 * NOTE: When you start using Amazon SNS to send SMS messages, your AWS account
 * is in the SMS sandbox and you can only
 * use verified destination phone numbers. See https://docs.aws.amazon.com/sns/
 * latest/dg/sns-sms-sandbox.html.
 * NOTE: If destination is in the US, you also have an additional restriction
 * that you have use a dedicated
 * origination ID (phone number). You can request an origination number using
 * Amazon Pinpoint for a fee.
 * See https://aws.amazon.com/blogs/compute/provisioning-and-using-10dlc-
 * origination-numbers-with-amazon-sns/
 * for more information.
 *
 * <phone_number_value> input parameter uses E.164 format.
 * For example, in United States, this input value should be of the form:
 * +12223334444
 */
```

```
//! Send an SMS text message to a phone number.
/*!
  \param message: The message to publish.
  \param phoneNumber: The phone number of the recipient in E.164 format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
*/
bool AwsDoc::SNS::publishSms(const Aws::String &message,
                             const Aws::String &phoneNumber,
                             const Aws::Client::ClientConfiguration
&clientConfiguration) {
    Aws::SNS::SNSClient snsClient(clientConfiguration);

    Aws::SNS::Model::PublishRequest request;
    request.SetMessage(message);
    request.SetPhoneNumber(phoneNumber);

    const Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

    if (outcome.IsSuccess()) {
        std::cout << "Message published successfully with message id, '"
                    << outcome.GetResult().GetMessageId() << "'."
                    << std::endl;
    }
    else {
        std::cerr << "Error while publishing message "
                    << 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「[Publish](#)」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。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import software.amazon.awssdk.services.sns.model.SnsException;

/**
 * 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
 */
public class PublishTextSMS {
    public static void main(String[] args) {
        final String usage = ""

            Usage:    <message> <phoneNumber>

            Where:
                message - The message text to send.
                phoneNumber - The mobile phone number to which a message is
sent (for example, +1XXX5550100).\s
            "";

        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }
    }
}
```

```
String message = args[0];
String phoneNumber = args[1];
SnsClient snsClient = SnsClient.builder()
    .region(Region.US_EAST_1)
    .build();
pubTextSMS(snsClient, message, phoneNumber);
snsClient.close();
}

public static void pubTextSMS(SnsClient snsClient, String message, String
phoneNumber) {
    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .phoneNumber(phoneNumber)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out
            .println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
}
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「[Publish](#)」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 pubTextSMS(
    messageVal: String?,
    phoneNumberVal: String?,
) {
    val request =
        PublishRequest {
            message = messageVal
            phoneNumber = phoneNumberVal
        }

    SnsClient.fromEnvironment { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println("${result.messageId} message sent.")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「[Publish](#)」を参照してください。

PHP

SDK for PHP

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。

```
require 'vendor/autoload.php';

use Aws\Exception\AwsException;
use Aws\Sns\SnsClient;

/**
 * Sends a text message (SMS message) directly to a phone number using Amazon
 * SNS.
 *
 * This code expects that you have AWS credentials set up per:
```

```
* https://docs.aws.amazon.com/sdk-for-php/v3/developer-guide/
guide_credentials.html
*/

$SnSClient = new SnsClient([
    'profile' => 'default',
    'region' => 'us-east-1',
    'version' => '2010-03-31'
]);

$message = 'This message is sent from a Amazon SNS code sample.';
$phone = '+1XXX5550100';

try {
    $result = $SnSClient->publish([
        'Message' => $message,
        'PhoneNumber' => $phone,
    ]);
    var_dump($result);
} catch (AwsException $e) {
    // output error message if fails
    error_log($e->getMessage());
}
```

- 詳細については、「[AWS SDK for PHP デベロッパーガイド](#)」を参照してください。
- API の詳細については、AWS SDK for PHP API リファレンスの「[発行](#)」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 SnsWrapper:
```

```
"""Encapsulates Amazon SNS topic and subscription functions."""

def __init__(self, sns_resource):
    """
    :param sns_resource: A Boto3 Amazon SNS resource.
    """
    self.sns_resource = sns_resource

def publish_text_message(self, phone_number, message):
    """
    Publishes a text message directly to a phone number without need for a
    subscription.

    :param phone_number: The phone number that receives the message. This
    must be
                           in E.164 format. For example, a United States phone
                           number might be +12065550101.
    :param message: The message to send.
    :return: The ID of the message.
    """
    try:
        response = self.sns_resource.meta.client.publish(
            PhoneNumber=phone_number, Message=message
        )
        message_id = response["MessageId"]
        logger.info("Published message to %s.", phone_number)
    except ClientError:
        logger.exception("Couldn't publish message to %s.", phone_number)
        raise
    else:
        return message_id
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の「[Publish](#)」を参照してください。

SAP ABAP

SDK for SAP ABAP

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。

```
" iv_phone_number = '+12065550101' - Phone number in E.164 format
TRY.
    oo_result = lo_sns->publish(                                " oo_result is returned for
testing purposes. "
        iv_phonenumber = iv_phone_number
        iv_message = iv_message ).
    MESSAGE 'Message published to phone number.' TYPE 'I'.
CATCH /aws1/cx_snsnotfoundexception.
    MESSAGE 'Phone number does not exist.' TYPE 'E'.
ENDTRY.
```

- API の詳細については、AWS SDK for SAP ABAP API リファレンスの「[Publish](#)」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 での Amazon SNS 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 を使用して Amazon SNS メッセージを Amazon SQS キューに発行する

次のコード例は、以下を実行する方法を示しています。

- トピック (FIFO または非 FIFO) を作成します。
- フィルターを適用するオプションを使用して、複数のキューをトピックにサブスクライブします。
- メッセージをトピックに発行します。
- キューをポーリングして受信メッセージを確認します。

.NET

SDK for .NET

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。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```
/// <summary>
/// Console application to run a feature scenario for topics and queues.
/// </summary>
public static class TopicsAndQueues
{
    private static bool _useFifoTopic = false;
    private static bool _useContentBasedDeduplication = false;
    private static string _topicName = null!;
    private static string _topicArn = null!;

    private static readonly int _queueCount = 2;
    private static readonly string[] _queueUrls = new string[_queueCount];
    private static readonly string[] _subscriptionArns = new string[_queueCount];
    private static readonly string[] _tones = { "cheerful", "funny", "serious",
"sincere" };
    public static SNSWrapper SnsWrapper { get; set; } = null!;
    public static SQSWrapper SqsWrapper { get; set; } = null!;
    public static bool UseConsole { get; set; } = true;
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EventBridge.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonSQS>()
                    .AddAWSService<IAmazonSimpleNotificationService>()
```

```
        .AddTransient<SNSWrapper>()
        .AddTransient<SQSWrapper>()
    )
    .Build();

    ServicesSetup(host);
    PrintDescription();

    await RunScenario();
}

/// <summary>
/// Populate the services for use within the console application.
/// </summary>
/// <param name="host">The services host.</param>
private static void ServicesSetup(IHost host)
{
    SnsWrapper = host.Services.GetRequiredService<SNSWrapper>();
    SqsWrapper = host.Services.GetRequiredService<SQSWrapper>();
}

/// <summary>
/// Run the scenario for working with topics and queues.
/// </summary>
/// <returns>True if successful.</returns>
public static async Task<bool> RunScenario()
{
    try
    {
        await SetupTopic();

        await SetupQueues();

        await PublishMessages();

        foreach (var queueUrl in _queueUrls)
        {
            var messages = await PollForMessages(queueUrl);
            if (messages.Any())
            {
                await DeleteMessages(queueUrl, messages);
            }
        }
    }
}
```

```

        await CleanupResources();

        Console.WriteLine("Messaging with topics and queues scenario is
complete.");
        return true;
    }
    catch (Exception ex)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"There was a problem running the scenario:
{ex.Message}");
        await CleanupResources();
        Console.WriteLine(new string('-', 80));
        return false;
    }
}

/// <summary>
/// Print a description for the tasks in the scenario.
/// </summary>
/// <returns>Async task.</returns>
private static void PrintDescription()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Welcome to messaging with topics and queues.");

    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"In this scenario, you will create an SNS topic and
subscribe {_queueCount} SQS queues to the topic." +
        $"{r\n}You can select from several options for
configuring the topic and the subscriptions for the 2 queues." +
        $"{r\n}You can then post to the topic and see the
results in the queues.\r\n");

    Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up the SNS topic to be used with the queues.
/// </summary>
/// <returns>Async task.</returns>
private static async Task<string> SetupTopic()
{
    Console.WriteLine(new string('-', 80));

```

```

        Console.WriteLine($"SNS topics can be configured as FIFO (First-In-First-Out)." +
            $"\r\nFIFO topics deliver messages in order and support deduplication and message filtering." +
            $"\r\nYou can then post to the topic and see the results in the queues.\r\n");

        _useFifoTopic = GetYesNoResponse("Would you like to work with FIFO topics?");

        if (_useFifoTopic)
        {
            Console.WriteLine(new string('-', 80));
            _topicName = GetUserResponse("Enter a name for your SNS topic: ", "example-topic");
            Console.WriteLine(
                "Because you have selected a FIFO topic, '.fifo' must be appended to the topic name.\r\n");

            Console.WriteLine(new string('-', 80));
            Console.WriteLine($"Because you have chosen a FIFO topic, deduplication is supported." +
                $"\r\nDeduplication IDs are either set in the message or automatically generated " +
                $"\r\nfrom content using a hash function.\r\n" +
                $"\r\nIf a message is successfully published to an SNS FIFO topic, any message " +
                $"\r\npublished and determined to have the same deduplication ID, " +
                $"\r\nwithin the five-minute deduplication interval, is accepted but not delivered.\r\n" +
                $"\r\nFor more information about deduplication, " +
                $"\r\nsee https://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.");

            _useContentBasedDeduplication = GetYesNoResponse("Use content-based deduplication instead of entering a deduplication ID?");
            Console.WriteLine(new string('-', 80));
        }

        _topicArn = await SnsWrapper.CreateTopicWithName(_topicName, _useFifoTopic, _useContentBasedDeduplication);

        Console.WriteLine($"Your new topic with the name {_topicName}" +

```



```

        $"\\r\\nand Amazon Resource Name (ARN) {_topicArn}" +
        $"\\r\\nhas been created.\\r\\n");

    Console.WriteLine(new string('-', 80));
    return _topicArn;
}

/// <summary>
/// Set up the queues.
/// </summary>
/// <returns>Async task.</returns>
private static async Task SetupQueues()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now you will create {_queueCount} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.");

    // Repeat this section for each queue.
    for (int i = 0; i < _queueCount; i++)
    {
        var queueName = GetUserResponse("Enter a name for an Amazon SQS
queue: ", $"example-queue-{i}");
        if (_useFifoTopic)
        {
            // Only explain this once.
            if (i == 0)
            {
                Console.WriteLine(
                    "Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.");
            }

            var queueUrl = await SqsWrapper.CreateQueueWithName(queueName,
_useFifoTopic);

            _queueUrls[i] = queueUrl;

            Console.WriteLine($"Your new queue with the name {queueName}" +
                $"\\r\\nand queue URL {queueUrl}" +
                $"\\r\\nhas been created.\\r\\n");

            if (i == 0)
            {
                Console.WriteLine(

```

```

        $"The queue URL is used to retrieve the queue ARN,\r\n" +
        $"which is used to create a subscription.");
        Console.WriteLine(new string('-', 80));
    }

    var queueArn = await SqsWrapper.GetQueueArnByUrl(queueUrl);

    if (i == 0)
    {
        Console.WriteLine(
            $"An AWS Identity and Access Management (IAM) policy must
be attached to an SQS queue, enabling it to receive\r\n" +
            $"messages from an SNS topic");
    }

    await SqsWrapper.SetQueuePolicyForTopic(queueArn, _topicArn,
queueUrl);

    await SetupFilters(i, queueArn, queueName);
}

Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Set up filters with user options for a queue.
/// </summary>
/// <param name="queueCount">The number of this queue.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="queueName">The name of the queue.</param>
/// <returns>Async Task.</returns>
public static async Task SetupFilters(int queueCount, string queueArn, string
queueName)
{
    if (_useFifoTopic)
    {
        Console.WriteLine(new string('-', 80));
        // Only explain this once.
        if (queueCount == 0)
        {
            Console.WriteLine(
                "Subscriptions to a FIFO topic can have filters." +

```

```

        "If you add a filter to this subscription, then only the
        filtered messages " +
        "will be received in the queue.");

        Console.WriteLine(
            "For information about message filtering, " +
            "see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html");

        Console.WriteLine(
            "For this example, you can filter messages by a" +
            "TONE attribute.");
    }

    var useFilter = GetYesNoResponse($"Filter messages for {queueName}'s
subscription to the topic?");

    string? filterPolicy = null;
    if (useFilter)
    {
        filterPolicy = CreateFilterPolicy();
    }
    var subscriptionArn = await
SnsWrapper.SubscribeTopicWithFilter(_topicArn, filterPolicy,
    queueArn);
    _subscriptionArns[queueCount] = subscriptionArn;

    Console.WriteLine(
        $"The queue {queueName} has been subscribed to the topic
{_topicName} " +
        $"with the subscription ARN {subscriptionArn}");
    Console.WriteLine(new string('-', 80));
}
}

/// <summary>
/// Use user input to create a filter policy for a subscription.
/// </summary>
/// <returns>The serialized filter policy.</returns>
public static string CreateFilterPolicy()
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine(
        $"You can filter messages by one or more of the following" +

```

```

        $"TONE attributes.");

        List<string> filterSelections = new List<string>();

        var selectionNumber = 0;
        do
        {
            Console.WriteLine(
                $"Enter a number to add a TONE filter, or enter 0 to stop adding
filters.");
            for (int i = 0; i < _tones.Length; i++)
            {
                Console.WriteLine($"\\t{i + 1}. {_tones[i]}");
            }

            var selection = GetUserResponse("", filterSelections.Any() ? "0" :
"1");

            int.TryParse(selection, out selectionNumber);
            if (selectionNumber > 0 && !
filterSelections.Contains(_tones[selectionNumber - 1]))
            {
                filterSelections.Add(_tones[selectionNumber - 1]);
            }
        } while (selectionNumber != 0);

        var filters = new Dictionary<string, List<string>>
        {
            { "tone", filterSelections }
        };
        string filterPolicy = JsonSerializer.Serialize(filters);
        return filterPolicy;
    }

    /// <summary>
    /// Publish messages using user settings.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task PublishMessages()
    {
        Console.WriteLine("Now we can publish messages.");

        var keepSendingMessages = true;
        string? deduplicationId = null;
        string? toneAttribute = null;
    }

```

```

        while (keepSendingMessages)
        {
            Console.WriteLine();
            var message = GetUserResponse("Enter a message to publish.", "This is
a sample message");

            if (_useFifoTopic)
            {
                Console.WriteLine("Because you are using a FIFO topic, you must
set a message group ID." +
                                "\r\nAll messages within the same group will be
received in the order " +
                                "they were published.");

                Console.WriteLine();
                var messageId = GetUserResponse("Enter a message group ID
for this message:", "1");

                if (!_useContentBasedDeduplication)
                {
                    Console.WriteLine("Because you are not using content-based
deduplication, " +
                                    "you must enter a deduplication ID.");

                    Console.WriteLine("Enter a deduplication ID for this
message.");
                    deduplicationId = GetUserResponse("Enter a deduplication ID
for this message.", "1");
                }

                if (GetYesNoResponse("Add an attribute to this message?"))
                {
                    Console.WriteLine("Enter a number for an attribute.");
                    for (int i = 0; i < _tones.Length; i++)
                    {
                        Console.WriteLine($"{i + 1}. {_tones[i]}");
                    }

                    var selection = GetUserResponse("", "1");
                    int.TryParse(selection, out var selectionNumber);

                    if (selectionNumber > 0 && selectionNumber < _tones.Length)
                    {
                        toneAttribute = _tones[selectionNumber - 1];
                    }
                }
            }
        }
    }
}

```

```

        }
    }

    var messageID = await SnsWrapper.PublishToTopicWithAttribute(
        _topicArn, message, "tone", toneAttribute, deduplicationId,
messageGroupId);

    Console.WriteLine($"Message published with id {messageID}.");
}

keepSendingMessages = GetYesNoResponse("Send another message?",
false);
}
}

/// <summary>
/// Poll for the published messages to see the results of the user's choices.
/// </summary>
/// <returns>Async task.</returns>
public static async Task<List<Message>> PollForMessages(string queueUrl)
{
    Console.WriteLine(new string('-', 80));
    Console.WriteLine($"Now the SQS queue at {queueUrl} will be polled to
retrieve the messages." +
        "\r\nPress any key to continue.");
    if (UseConsole)
    {
        Console.ReadLine();
    }

    var moreMessages = true;
    var messages = new List<Message>();
    while (moreMessages)
    {
        var newMessages = await SqsWrapper.ReceiveMessagesByUrl(queueUrl,
10);

        moreMessages = newMessages.Any();
        if (moreMessages)
        {
            messages.AddRange(newMessages);
        }
    }
}

```

```
        Console.WriteLine($"{messages.Count} message(s) were received by the
queue at {queueUrl}.");

        foreach (var message in messages)
        {
            Console.WriteLine("\tMessage:" +
                              $"{message.Body}");
        }

        Console.WriteLine(new string('-', 80));
        return messages;
    }

    /// <summary>
    /// Delete the message using handles in a batch.
    /// </summary>
    /// <returns>Async task.</returns>
    public static async Task DeleteMessages(string queueUrl, List<Message>
messages)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Now we can delete the messages in this queue in a
batch.");
        await SqsWrapper.DeleteMessageBatchByUrl(queueUrl, messages);
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up the resources from the scenario.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        try
        {
            foreach (var queueUrl in _queueUrls)
            {
                if (!string.IsNullOrEmpty(queueUrl))
                {
                    var deleteQueue =
                        GetYesNoResponse($"Delete queue with url {queueUrl}?");
```

```

        if (deleteQueue)
        {
            await SqsWrapper.DeleteQueueByUrl(queueUrl);
        }
    }

    foreach (var subscriptionArn in _subscriptionArns)
    {
        if (!string.IsNullOrEmpty(subscriptionArn))
        {
            await SnsWrapper.UnsubscribeByArn(subscriptionArn);
        }
    }

    var deleteTopic = GetYesNoResponse($"Delete topic {_topicName}?");
    if (deleteTopic)
    {
        await SnsWrapper.DeleteTopicByArn(_topicArn);
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Unable to clean up resources. Here's why:
{ex.Message}.");
}

Console.WriteLine(new string('-', 80));
}

/// <summary>
/// Helper method to get a yes or no response from the user.
/// </summary>
/// <param name="question">The question string to print on the console.</
param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static bool GetYesNoResponse(string question, bool defaultAnswer =
true)
{
    if (UseConsole)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
    }
}

```



```

        var response = ynResponse != null &&
            ynResponse.Equals("y",
                StringComparison.InvariantCultureIgnoreCase);

        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}

/// <summary>
/// Helper method to get a string response from the user through the console.
/// </summary>
/// <param name="question">The question string to print on the console.</
param>
/// <param name="defaultAnswer">Optional default answer to use.</param>
/// <returns>True if the user responds with a yes.</returns>
private static string GetUserResponse(string question, string defaultAnswer)
{
    if (UseConsole)
    {
        var response = "";
        while (string.IsNullOrEmpty(response))
        {
            Console.WriteLine(question);
            response = Console.ReadLine();
        }
        return response;
    }
    // If not using the console, use the default.
    return defaultAnswer;
}
}

```

Amazon SQS オペレーションをラップするクラスを作成します。

```

/// <summary>
/// Wrapper for Amazon Simple Queue Service (SQS) operations.
/// </summary>
public class SQSWrapper
{
    private readonly IAmazonSQS _amazonSQSClient;

```

```
/// <summary>
/// Constructor for the Amazon SQS wrapper.
/// </summary>
/// <param name="amazonSQS">The injected Amazon SQS client.</param>
public SQSWrapper(IAmazonSQS amazonSQS)
{
    _amazonSQSClient = amazonSQS;
}

/// <summary>
/// Create a queue with a specific name.
/// </summary>
/// <param name="queueName">The name for the queue.</param>
/// <param name="useFifoQueue">True to use a FIFO queue.</param>
/// <returns>The url for the queue.</returns>
public async Task<string> CreateQueueWithName(string queueName, bool
useFifoQueue)
{
    int maxMessage = 256 * 1024;
    var queueAttributes = new Dictionary<string, string>
    {
        {
            QueueAttributeName.MaximumMessageSize,
            maxMessage.ToString()
        }
    };

    var createQueueRequest = new CreateQueueRequest()
    {
        QueueName = queueName,
        Attributes = queueAttributes
    };

    if (useFifoQueue)
    {
        // Update the name if it is not correct for a FIFO queue.
        if (!queueName.EndsWith(".fifo"))
        {
            createQueueRequest.QueueName = queueName + ".fifo";
        }

        // Add an attribute for a FIFO queue.
        createQueueRequest.Attributes.Add(
```

```

        QueueAttributeName.FifoQueue, "true");
    }

    var createResponse = await _amazonSQSClient.CreateQueueAsync(
        new CreateQueueRequest()
        {
            QueueName = queueName
        });
    return createResponse.QueueUrl;
}

/// <summary>
/// Get the ARN for a queue from its URL.
/// </summary>
/// <param name="queueUrl">The URL of the queue.</param>
/// <returns>The ARN of the queue.</returns>
public async Task<string> GetQueueArnByUrl(string queueUrl)
{
    var getAttributesRequest = new GetQueueAttributesRequest()
    {
        QueueUrl = queueUrl,
        AttributeNames = new List<string>() { QueueAttributeName.QueueArn }
    };

    var getAttributesResponse = await
        _amazonSQSClient.GetQueueAttributesAsync(
            getAttributesRequest);

    return getAttributesResponse.QueueARN;
}

/// <summary>
/// Set the policy attribute of a queue for a topic.
/// </summary>
/// <param name="queueArn">The ARN of the queue.</param>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="queueUrl">The url for the queue.</param>
/// <returns>True if successful.</returns>
public async Task<bool> SetQueuePolicyForTopic(string queueArn, string
topicArn, string queueUrl)
{
    var queuePolicy = "{" +
        "\"Version\": \"2012-10-17\", " +
        "\"Statement\": [{" +

```

```

        "\"Effect\": \"Allow\", \" +
        "\"Principal\": {\" +
            $\"\"Service\": \" +
                \"sns.amazonaws.com\" +
            \"}, \" +
        "\"Action\": \"sqs:SendMessage\", \" +
        $\"\"Resource\": \"{queueArn}\", \" +
        "\"Condition\": {\" +
            \"ArnEquals\": {\" +
                $\"\"aws:SourceArn\":
    \"{topicArn}\" +
        \"}\" +
        \"}\" +
    \"}]]\" +
    \"\";

    var attributesResponse = await _amazonSQSClient.SetQueueAttributesAsync(
        new SetQueueAttributesRequest()
        {
            QueueUrl = queueUrl,
            Attributes = new Dictionary<string, string>() { { \"Policy\",
queuePolicy } }
        });
    return attributesResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Receive messages from a queue by its URL.
/// </summary>
/// <param name=\"queueUrl\">The url of the queue.</param>
/// <returns>The list of messages.</returns>
public async Task<List<Message>> ReceiveMessagesByUrl(string queueUrl, int
maxMessages)
{
    // Setting WaitTimeSeconds to non-zero enables long polling.
    // For information about long polling, see
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
    var messageResponse = await _amazonSQSClient.ReceiveMessageAsync(
        new ReceiveMessageRequest()
        {
            QueueUrl = queueUrl,
            MaxNumberOfMessages = maxMessages,
            WaitTimeSeconds = 1
        });
}

```

```
        return messageResponse.Messages;
    }

    /// <summary>
    /// Delete a batch of messages from a queue by its url.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteMessageBatchByUrl(string queueUrl,
List<Message> messages)
    {
        var deleteRequest = new DeleteMessageBatchRequest()
        {
            QueueUrl = queueUrl,
            Entries = new List<DeleteMessageBatchRequestEntry>()
        };
        foreach (var message in messages)
        {
            deleteRequest.Entries.Add(new DeleteMessageBatchRequestEntry()
            {
                ReceiptHandle = message.ReceiptHandle,
                Id = message.MessageId
            });
        }

        var deleteResponse = await
        _amazonSQSClient.DeleteMessageBatchAsync(deleteRequest);

        return deleteResponse.Failed.Any();
    }

    /// <summary>
    /// Delete a queue by its URL.
    /// </summary>
    /// <param name="queueUrl">The url of the queue.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteQueueByUrl(string queueUrl)
    {
        var deleteResponse = await _amazonSQSClient.DeleteQueueAsync(
            new DeleteQueueRequest()
            {
                QueueUrl = queueUrl
            });
        return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

```
}  
}
```

Amazon SNS オペレーションをラップするクラスを作成します。

```
/// <summary>  
/// Wrapper for Amazon Simple Notification Service (SNS) operations.  
/// </summary>  
public class SNSWrapper  
{  
    private readonly IAmazonSimpleNotificationService _amazonSNSClient;  
  
    /// <summary>  
    /// Constructor for the Amazon SNS wrapper.  
    /// </summary>  
    /// <param name="amazonSNS">The injected Amazon SNS client.</param>  
    public SNSWrapper(IAmazonSimpleNotificationService amazonSNS)  
    {  
        _amazonSNSClient = amazonSNS;  
    }  
  
    /// <summary>  
    /// Create a new topic with a name and specific FIFO and de-duplication  
    attributes.  
    /// </summary>  
    /// <param name="topicName">The name for the topic.</param>  
    /// <param name="useFifoTopic">True to use a FIFO topic.</param>  
    /// <param name="useContentBasedDeduplication">True to use content-based de-  
    duplication.</param>  
    /// <returns>The ARN of the new topic.</returns>  
    public async Task<string> CreateTopicWithName(string topicName, bool  
    useFifoTopic, bool useContentBasedDeduplication)  
    {  
        var createTopicRequest = new CreateTopicRequest()  
        {  
            Name = topicName,  
        };  
  
        if (useFifoTopic)  
        {  
            // Update the name if it is not correct for a FIFO topic.
```

```

        if (!topicName.EndsWith(".fifo"))
        {
            createTopicRequest.Name = topicName + ".fifo";
        }

        // Add the attributes from the method parameters.
        createTopicRequest.Attributes = new Dictionary<string, string>
        {
            { "FifoTopic", "true" }
        };
        if (useContentBasedDeduplication)
        {
            createTopicRequest.Attributes.Add("ContentBasedDeduplication",
"true");
        }
    }

    var createResponse = await
_amazonSNSClient.CreateTopicAsync(createTopicRequest);
    return createResponse.TopicArn;
}

/// <summary>
/// Subscribe a queue to a topic with optional filters.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="useFifoTopic">The optional filtering policy for the
subscription.</param>
/// <param name="queueArn">The ARN of the queue.</param>
/// <returns>The ARN of the new subscription.</returns>
public async Task<string> SubscribeTopicWithFilter(string topicArn, string?
filterPolicy, string queueArn)
{
    var subscribeRequest = new SubscribeRequest()
    {
        TopicArn = topicArn,
        Protocol = "sqs",
        Endpoint = queueArn
    };

    if (!string.IsNullOrEmpty(filterPolicy))
    {
        subscribeRequest.Attributes = new Dictionary<string, string>
{ { "FilterPolicy", filterPolicy } };
    }
}

```

```

    }

    var subscribeResponse = await
        _amazonSNSClient.SubscribeAsync(subscribeRequest);
    return subscribeResponse.SubscriptionArn;
}

/// <summary>
/// Publish a message to a topic with an attribute and optional deduplication
and group IDs.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <param name="message">The message to publish.</param>
/// <param name="attributeName">The optional attribute for the message.</
param>
/// <param name="attributeValue">The optional attribute value for the
message.</param>
/// <param name="deduplicationId">The optional deduplication ID for the
message.</param>
/// <param name="groupId">The optional group ID for the message.</param>
/// <returns>The ID of the message published.</returns>
public async Task<string> PublishToTopicWithAttribute(
    string topicArn,
    string message,
    string? attributeName = null,
    string? attributeValue = null,
    string? deduplicationId = null,
    string? groupId = null)
{
    var publishRequest = new PublishRequest()
    {
        TopicArn = topicArn,
        Message = message,
        MessageDeduplicationId = deduplicationId,
        MessageGroupId = groupId
    };

    if (attributeValue != null)
    {
        // Add the string attribute if it exists.
        publishRequest.MessageAttributes =
            new Dictionary<string, MessageAttributeValue>
            {

```



```
        { attributeName!, new MessageAttributeValue() { StringValue =
attributeValue, DataType = "String"} }
        };
    }

    var publishResponse = await
_amazonSNSClient.PublishAsync(publishRequest);
    return publishResponse.MessageId;
}

/// <summary>
/// Unsubscribe from a topic by a subscription ARN.
/// </summary>
/// <param name="subscriptionArn">The ARN of the subscription.</param>
/// <returns>True if successful.</returns>
public async Task<bool> UnsubscribeByArn(string subscriptionArn)
{
    var unsubscribeResponse = await _amazonSNSClient.UnsubscribeAsync(
        new UnsubscribeRequest()
        {
            SubscriptionArn = subscriptionArn
        });
    return unsubscribeResponse.HttpStatusCode == HttpStatusCode.OK;
}

/// <summary>
/// Delete a topic by its topic ARN.
/// </summary>
/// <param name="topicArn">The ARN of the topic.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteTopicByArn(string topicArn)
{
    var deleteResponse = await _amazonSNSClient.DeleteTopicAsync(
        new DeleteTopicRequest()
        {
            TopicArn = topicArn
        });
    return deleteResponse.HttpStatusCode == HttpStatusCode.OK;
}
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

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::Client::ClientConfiguration clientConfig;
// Optional: Set to the AWS Region (overrides config file).
// clientConfig.region = "us-east-1";

//! Workflow for messaging with topics and queues using Amazon SNS and Amazon
SQS.
/*!
\param clientConfig Aws client configuration.
\return bool: Successful completion.
*/
bool AwsDoc::TopicsAndQueues::messagingWithTopicsAndQueues(
    const Aws::Client::ClientConfiguration &clientConfiguration) {
    std::cout << "Welcome to messaging with topics and queues." << std::endl;
```

```
printAsterisksLine();
std::cout << "In this workflow, you will create an SNS topic and subscribe "
          << NUMBER_OF_QUEUES <<
          << " SQS queues to the topic." << std::endl;
std::cout
    << "You can select from several options for configuring the topic and
the subscriptions for the "
    << NUMBER_OF_QUEUES << " queues." << std::endl;
std::cout << "You can then post to the topic and see the results in the
queues."
    << std::endl;

Aws::SNS::SNSClient snsClient(clientConfiguration);

printAsterisksLine();

std::cout << "SNS topics can be configured as FIFO (First-In-First-Out)."
    << std::endl;
std::cout
    << "FIFO topics deliver messages in order and support deduplication
and message filtering."
    << std::endl;
bool isFifoTopic = askYesNoQuestion(
    "Would you like to work with FIFO topics? (y/n) ");

bool contentBasedDeduplication = false;
Aws::String topicName;
if (isFifoTopic) {
    printAsterisksLine();
    std::cout << "Because you have chosen a FIFO topic, deduplication is
supported."
    << std::endl;
    std::cout
        << "Deduplication IDs are either set in the message or
automatically generated "
        << "from content using a hash function." << std::endl;
    std::cout
        << "If a message is successfully published to an SNS FIFO topic,
any message "
        << "published and determined to have the same deduplication ID, "
        << std::endl;
    std::cout
        << "within the five-minute deduplication interval, is accepted
but not delivered."
```

```
        << std::endl;
    std::cout
        << "For more information about deduplication, "
        << "see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html."
        << std::endl;
    contentBasedDeduplication = askYesNoQuestion(
        "Use content-based deduplication instead of entering a
deduplication ID? (y/n) ");
    }

    printAsterisksLine();

    Aws::SQS::SQSClient sqsClient(clientConfiguration);
    Aws::Vector<Aws::String> queueURLS;
    Aws::Vector<Aws::String> subscriptionARNS;

    Aws::String topicARN;
    {
        topicName = askQuestion("Enter a name for your SNS topic. ");

        // 1. Create an Amazon SNS topic, either FIFO or non-FIFO.
        Aws::SNS::Model::CreateTopicRequest request;

        if (isFifoTopic) {
            request.AddAttributes("FifoTopic", "true");
            if (contentBasedDeduplication) {
                request.AddAttributes("ContentBasedDeduplication", "true");
            }
            topicName = topicName + FIFO_SUFFIX;

            std::cout
                << "Because you have selected a FIFO topic, '.fifo' must be
appended to the topic name."
                << std::endl;
        }

        request.SetName(topicName);

        Aws::SNS::Model::CreateTopicOutcome outcome =
snsClient.CreateTopic(request);

        if (outcome.IsSuccess()) {
            topicARN = outcome.GetResult().GetTopicArn();
        }
    }
}
```

```

        std::cout << "Your new topic with the name '" << topicName
                    << "' and the topic Amazon Resource Name (ARN) " <<
std::endl;
        std::cout << "'" << topicARN << "' has been created." << std::endl;

    }
    else {
        std::cerr << "Error with TopicsAndQueues::CreateTopic. "
                    << outcome.GetError().GetMessage()
                    << std::endl;

        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }
}

printAsterisksLine();

std::cout << "Now you will create " << NUMBER_OF_QUEUES
            << " SQS queues to subscribe to the topic." << std::endl;
Aws::Vector<Aws::String> queueNames;
bool filteringMessages = false;
bool first = true;
for (int i = 1; i <= NUMBER_OF_QUEUES; ++i) {
    Aws::String queueURL;
    Aws::String queueName;
    {
        printAsterisksLine();
        std::ostringstream ostringstream;
        ostringstream << "Enter a name for " << (first ? "an" : "the next")
                        << " SQS queue. ";
        queueName = askQuestion(ostringstream.str());

        // 2. Create an SQS queue.
        Aws::SQS::Model::CreateQueueRequest request;
        if (isFifoTopic) {
            request.AddAttributes(Aws::SQS::Model::QueueAttributeName::FifoQueue,
                                "true");

```

```

        queueName = queueName + FIFO_SUFFIX;

        if (first) // Only explain this once.
        {
            std::cout
                << "Because you are creating a FIFO SQS queue,
'.fifo' must "
                << "be appended to the queue name." << std::endl;
        }
    }

    request.SetQueueName(queueName);
    queueNames.push_back(queueName);

    Aws::SQS::Model::CreateQueueOutcome outcome =
        sqsClient.CreateQueue(request);

    if (outcome.IsSuccess()) {
        queueURL = outcome.GetResult().GetQueueUrl();
        std::cout << "Your new SQS queue with the name '" << queueName
            << "' and the queue URL " << std::endl;
        std::cout << "'" << queueURL << "' has been created." <<
std::endl;
    }
    else {
        std::cerr << "Error with SQS::CreateQueue. "
            << outcome.GetError().GetMessage()
            << std::endl;

        cleanUp(topicARN,
            queueURLS,
            subscriptionARNS,
            snsClient,
            sqsClient);

        return false;
    }
}
queueURLS.push_back(queueURL);

if (first) // Only explain this once.
{
    std::cout

```

```

        << "The queue URL is used to retrieve the queue ARN, which is
"
        << "used to create a subscription." << std::endl;
    }

    Aws::String queueARN;
    {
        // 3. Get the SQS queue ARN attribute.
        Aws::SQS::Model::GetQueueAttributesRequest request;
        request.SetQueueUrl(queueURL);

        request.AddAttributeNames(Aws::SQS::Model::QueueAttributeName::QueueArn);

        Aws::SQS::Model::GetQueueAttributesOutcome outcome =
            sqsClient.GetQueueAttributes(request);

        if (outcome.IsSuccess()) {
            const Aws::Map<Aws::SQS::Model::QueueAttributeName, Aws::String>
&attributes =
                outcome.GetResult().GetAttributes();
            const auto &iter = attributes.find(
                Aws::SQS::Model::QueueAttributeName::QueueArn);
            if (iter != attributes.end()) {
                queueARN = iter->second;
                std::cout << "The queue ARN '" << queueARN
                    << "' has been retrieved."
                    << std::endl;
            }
            else {
                std::cerr
                    << "Error ARN attribute not returned by
GetQueueAttribute."
                    << std::endl;

                cleanUp(topicARN,
                    queueURLS,
                    subscriptionARNS,
                    snsClient,
                    sqsClient);

                return false;
            }
        }
        else {

```

```

        std::cerr << "Error with SQS::GetQueueAttributes. "
        << outcome.GetError().GetMessage()
        << std::endl;

        cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

        return false;
    }
}

if (first) {
    std::cout
        << "An IAM policy must be attached to an SQS queue, enabling
it to receive "
        << "messages from an SNS topic." << std::endl;
}

{
    // 4. Set the SQS queue policy attribute with a policy enabling the
receipt of SNS messages.
    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(queueURL);
    Aws::String policy = createPolicyForQueue(queueARN, topicARN);
    request.AddAttributes(Aws::SQS::Model::QueueAttributeName::Policy,
                        policy);

    Aws::SQS::Model::SetQueueAttributesOutcome outcome =
        sqsClient.SetQueueAttributes(request);

    if (outcome.IsSuccess()) {
        std::cout << "The attributes for the queue '" << queueName
        << "' were successfully updated." << std::endl;
    }
    else {
        std::cerr << "Error with SQS::SetQueueAttributes. "
        << outcome.GetError().GetMessage()
        << std::endl;

        cleanUp(topicARN,
                queueURLS,

```



```

        subscriptionARNs,
        snsClient,
        sqsClient);

    return false;
}
}

printAsterisksLine();

{
    // 5. Subscribe the SQS queue to the SNS topic.
    Aws::SNS::Model::SubscribeRequest request;
    request.SetTopicArn(topicARN);
    request.SetProtocol("sqs");
    request.SetEndpoint(queueARN);
    if (isFifoTopic) {
        if (first) {
            std::cout << "Subscriptions to a FIFO topic can have
filters."
                        << std::endl;
            std::cout
                << "If you add a filter to this subscription, then
only the filtered messages "
                << "will be received in the queue." << std::endl;
            std::cout << "For information about message filtering, "
                << "see https://docs.aws.amazon.com/sns/latest/dg/
sns-message-filtering.html"
                << std::endl;
            std::cout << "For this example, you can filter messages by a
\"\"\"
                        << TONE_ATTRIBUTE << "\" attribute.\" << std::endl;
        }

        std::ostringstream ostream;
        ostream << "Filter messages for \"" << queueName
            << "\"'s subscription to the topic \""
            << topicName << "\"? (y/n)";

        // Add filter if user answers yes.
        if (askYesNoQuestion(ostream.str())) {
            Aws::String jsonPolicy = getFilterPolicyFromUser();
            if (!jsonPolicy.empty()) {
                filteringMessages = true;
            }
        }
    }
}

```

```

        std::cout << "This is the filter policy for this
subscription."
                << std::endl;
        std::cout << jsonPolicy << std::endl;

        request.AddAttributes("FilterPolicy", jsonPolicy);
    }
    else {
        std::cout
            << "Because you did not select any attributes, no
filter "
            << "will be added to this subscription." <<
std::endl;
    }
} // if (isFifoTopic)
Aws::SNS::Model::SubscribeOutcome outcome =
snsClient.Subscribe(request);

if (outcome.IsSuccess()) {
    Aws::String subscriptionARN =
outcome.GetResult().GetSubscriptionArn();
    std::cout << "The queue '" << queueName
                << "' has been subscribed to the topic '"
                << "'" << topicName << "'" << std::endl;
    std::cout << "with the subscription ARN '" << subscriptionARN <<
"."
            << std::endl;
    subscriptionARNS.push_back(subscriptionARN);
}
else {
    std::cerr << "Error with TopicsAndQueues::Subscribe. "
                << outcome.GetError().GetMessage()
                << std::endl;

    cleanUp(topicARN,
            queueURLS,
            subscriptionARNS,
            snsClient,
            sqsClient);

    return false;
}

```

```

    }

    first = false;
}

first = true;
do {
    printAsterisksLine();

    // 6. Publish a message to the SNS topic.
    Aws::SNS::Model::PublishRequest request;
    request.SetTopicArn(topicARN);
    Aws::String message = askQuestion("Enter a message text to publish. ");
    request.SetMessage(message);
    if (isFifoTopic) {
        if (first) {
            std::cout
                << "Because you are using a FIFO topic, you must set a
message group ID."
                << std::endl;
            std::cout
                << "All messages within the same group will be received
in the "
                << "order they were published." << std::endl;
        }
        Aws::String messageGroupID = askQuestion(
            "Enter a message group ID for this message. ");
        request.SetMessageGroupId(messageGroupID);
        if (!contentBasedDeduplication) {
            if (first) {
                std::cout
                    << "Because you are not using content-based
deduplication, "
                    << "you must enter a deduplication ID." << std::endl;
            }
            Aws::String deduplicationID = askQuestion(
                "Enter a deduplication ID for this message. ");
            request.SetMessageDeduplicationId(deduplicationID);
        }
    }

    if (filteringMessages && askYesNoQuestion(
        "Add an attribute to this message? (y/n) ")) {
        for (size_t i = 0; i < TONES.size(); ++i) {

```

```

        std::cout << "  " << (i + 1) << ". " << TONES[i] << std::endl;
    }
    int selection = askQuestionForIntRange(
        "Enter a number for an attribute. ",
        1, static_cast<int>(TONES.size()));
    Aws::SNS::Model::MessageAttributeValue messageAttributeValue;
    messageAttributeValue.SetDataType("String");
    messageAttributeValue.SetStringValue(TONES[selection - 1]);
    request.AddMessageAttributes(TONE_ATTRIBUTE, messageAttributeValue);
}

Aws::SNS::Model::PublishOutcome outcome = snsClient.Publish(request);

if (outcome.IsSuccess()) {
    std::cout << "Your message was successfully published." << std::endl;
}
else {
    std::cerr << "Error with TopicsAndQueues::Publish. "
                << outcome.GetError().GetMessage()
                << std::endl;

    cleanUp(topicARN,
            queueURLS,
            subscriptionARNS,
            snsClient,
            sqsClient);

    return false;
}

first = false;
} while (askYesNoQuestion("Post another message? (y/n) "));

printAsterisksLine();

std::cout << "Now the SQS queue will be polled to retrieve the messages."
          << std::endl;
askQuestion("Press any key to continue...", alwaysTrueTest);

for (size_t i = 0; i < queueURLS.size(); ++i) {
    // 7. Poll an SQS queue for its messages.
    std::vector<Aws::String> messages;
    std::vector<Aws::String> receiptHandles;
    while (true) {

```

```

        Aws::SQS::Model::ReceiveMessageRequest request;
        request.SetMaxNumberOfMessages(10);
        request.SetQueueUrl(queueURLS[i]);

        // Setting WaitTimeSeconds to non-zero enables long polling.
        // For information about long polling, see
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        request.SetWaitTimeSeconds(1);
        Aws::SQS::Model::ReceiveMessageOutcome outcome =
            sqsClient.ReceiveMessage(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::SQS::Model::Message> &newMessages =
outcome.GetResult().GetMessages();
            if (newMessages.empty()) {
                break;
            }
            else {
                for (const Aws::SQS::Model::Message &message: newMessages) {
                    messages.push_back(message.GetBody());
                    receiptHandles.push_back(message.GetReceiptHandle());
                }
            }
        }
        else {
            std::cerr << "Error with SQS::ReceiveMessage. "
                << outcome.GetError().GetMessage()
                << std::endl;

            cleanUp(topicARN,
                queueURLS,
                subscriptionARNS,
                snsClient,
                sqsClient);

            return false;
        }
    }

    printAsterisksLine();

    if (messages.empty()) {
        std::cout << "No messages were ";
    }

```

```

    }
    else if (messages.size() == 1) {
        std::cout << "One message was ";
    }
    else {
        std::cout << messages.size() << " messages were ";
    }
    std::cout << "received by the queue '" << queueNames[i]
        << "'." << std::endl;
    for (const Aws::String &message: messages) {
        std::cout << "  Message : '" << message << "'."
            << std::endl;
    }

    // 8. Delete a batch of messages from an SQS queue.
    if (!receiptHandles.empty()) {
        Aws::SQS::Model::DeleteMessageBatchRequest request;
        request.SetQueueUrl(queueURLS[i]);
        int id = 1; // Ids must be unique within a batch delete request.
        for (const Aws::String &receiptHandle: receiptHandles) {
            Aws::SQS::Model::DeleteMessageBatchRequestEntry entry;
            entry.SetId(std::to_string(id));
            ++id;
            entry.SetReceiptHandle(receiptHandle);
            request.AddEntries(entry);
        }

        Aws::SQS::Model::DeleteMessageBatchOutcome outcome =
            sqsClient.DeleteMessageBatch(request);

        if (outcome.IsSuccess()) {
            std::cout << "The batch deletion of messages was successful."
                << std::endl;
        }
        else {
            std::cerr << "Error with SQS::DeleteMessageBatch. "
                << outcome.GetError().GetMessage()
                << std::endl;
            cleanUp(topicARN,
                queueURLS,
                subscriptionARNs,
                snsClient,
                sqsClient);
        }
    }
}

```

```

        return false;
    }
}

return cleanUp(topicARN,
               queueURLS,
               subscriptionARNS,
               snsClient,
               sqsClient,
               true); // askUser
}

bool AwsDoc::TopicsAndQueues::cleanUp(const Aws::String &topicARN,
                                       const Aws::Vector<Aws::String> &queueURLS,
                                       const Aws::Vector<Aws::String>
                                       &subscriptionARNS,
                                       const Aws::SNS::SNSClient &snsClient,
                                       const Aws::SQS::SQSClient &sqsClient,
                                       bool askUser) {

    bool result = true;
    printAsterisksLine();
    if (!queueURLS.empty() && askUser &&
        askYesNoQuestion("Delete the SQS queues? (y/n) ")) {

        for (const auto &queueURL: queueURLS) {
            // 9. Delete an SQS queue.
            Aws::SQS::Model::DeleteQueueRequest request;
            request.SetQueueUrl(queueURL);

            Aws::SQS::Model::DeleteQueueOutcome outcome =
                sqsClient.DeleteQueue(request);

            if (outcome.IsSuccess()) {
                std::cout << "The queue with URL '" << queueURL
                          << "' was successfully deleted." << std::endl;
            }
            else {
                std::cerr << "Error with SQS::DeleteQueue. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }
}

```

```

    }

    for (const auto &subscriptionARN: subscriptionARNS) {
        // 10. Unsubscribe an SNS subscription.
        Aws::SNS::Model::UnsubscribeRequest request;
        request.SetSubscriptionArn(subscriptionARN);

        Aws::SNS::Model::UnsubscribeOutcome outcome =
            snsClient.Unsubscribe(request);

        if (outcome.IsSuccess()) {
            std::cout << "Unsubscribe of subscription ARN '" <<
subscriptionARN
                        << "' was successful." << std::endl;
        }
        else {
            std::cerr << "Error with TopicsAndQueues::Unsubscribe. "
                        << outcome.GetError().GetMessage()
                        << std::endl;
            result = false;
        }
    }
}

printAsterisksLine();
if (!topicARN.empty() && askUser &&
    askYesNoQuestion("Delete the SNS topic? (y/n) ")) {

    // 11. Delete an SNS topic.
    Aws::SNS::Model::DeleteTopicRequest request;
    request.SetTopicArn(topicARN);

    Aws::SNS::Model::DeleteTopicOutcome outcome =
snsClient.DeleteTopic(request);

    if (outcome.IsSuccess()) {
        std::cout << "The topic with ARN '" << topicARN
                    << "' was successfully deleted." << std::endl;
    }
    else {
        std::cerr << "Error with TopicsAndQueues::DeleteTopicRequest. "
                    << outcome.GetError().GetMessage()
                    << std::endl;
        result = false;
    }
}

```



```

    }
}

return result;
}

//! Create an IAM policy that gives an SQS queue permission to receive messages
from an SNS topic.
/*!
\sa createPolicyForQueue()
\param queueARN: The SQS queue Amazon Resource Name (ARN).
\param topicARN: The SNS topic ARN.
\return Aws::String: The policy as JSON.
*/
Aws::String AwsDoc::TopicsAndQueues::createPolicyForQueue(const Aws::String
&queueARN,
                                                             const Aws::String
&topicARN) {
    std::ostringstream policyStream;
    policyStream << R"({
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "sns.amazonaws.com"
                },
                "Action": "sqs:SendMessage",
                "Resource": ")" << queueARN << R"(",
                "Condition": {
                    "ArnEquals": {
                        "aws:SourceArn": ")" << topicARN << R"("
                    }
                }
            }
        ]
    })";

    return policyStream.str();
}

```

- APIの詳細については、「AWS SDK for C++ API リファレンス」の以下のトピックを参照してください。

- [CreateQueue](#)
- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Publish](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

Go

SDK for Go V2

 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。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```
import (  
    "context"  
    "encoding/json"  
    "fmt"  
    "log"  
    "strings"  
    "topics_and_queues/actions"  
  
    "github.com/aws/aws-sdk-go-v2/aws"  
    "github.com/aws/aws-sdk-go-v2/service/sns"  
    "github.com/aws/aws-sdk-go-v2/service/sqs"  
    "github.com/aws/aws-sdk-go-v2/service/sqs/types"
```

```
"github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
    Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions
// so that
// they are simpler to read and understand.
type ScenarioRunner struct {
    questioner demotools.IQuestioner
    snsActor    *actions.SnsActions
    sqsActor    *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string,
bool, bool) {
    log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or
standard.\n" +
        "FIFO topics deliver messages in order and support deduplication and message
filtering.")
    isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
topics? (y/n) ", "y")

    contentBasedDeduplication := false
    if isFifoTopic {
        log.Println(strings.Repeat("-", 88))
        log.Println("Because you have chosen a FIFO topic, deduplication is supported.
\n" +
            "Deduplication IDs are either set in the message or are automatically
generated\n" +
            "from content using a hash function. If a message is successfully published to
\n" +
            "an SNS FIFO topic, any message published and determined to have the same\n" +
            "deduplication ID, within the five-minute deduplication interval, is accepted
\n" +
            "but not delivered. For more information about deduplication, see:\n" +
```

```

    "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
    contentBasedDeduplication = runner.questioner.AskBool(
        "\nDo you want to use content-based deduplication instead of entering a
deduplication ID? (y/n) ", "y")
}
log.Println(strings.Repeat("-", 88))

topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
if isFifoTopic {
    topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
    log.Printf("Because you have selected a FIFO topic, '%v' must be appended to
\n"+
        "the topic name.", FIFO_SUFFIX)
}

topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
contentBasedDeduplication)
if err != nil {
    panic(err)
}
log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN)
\n"+
    "'%v' has been created.", topicName, topicArn)

return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
isFifoTopic bool) (string, string) {
    queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS
queue. ", ordinal))
    if isFifoTopic {
        queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
        if ordinal == "first" {
            log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
                "be appended to the queue name.\n", FIFO_SUFFIX)
        }
    }
    queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
    if err != nil {
        panic(err)
    }
    log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
        "'%v' has been created.", queueName, queueUrl)
}

```

```

    return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
    ctx context.Context, queueName string, queueUrl string, topicName string,
    topicArn string, ordinal string,
    isFifoTopic bool) (string, bool) {

    queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
    if err != nil {
        panic(err)
    }
    log.Printf("The ARN of your queue is: %v.\n", queueArn)

    err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
    if err != nil {
        panic(err)
    }
    log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
        "messages to it.")
    log.Println(strings.Repeat("-", 88))

    var filterPolicy map[string][]string
    if isFifoTopic {
        if ordinal == "first" {
            log.Println("Subscriptions to a FIFO topic can have filters.\n" +
                "If you add a filter to this subscription, then only the filtered messages\n" +
                +
                "will be received in the queue.\n" +
                "For information about message filtering, see\n" +
                "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
                "For this example, you can filter messages by a \"tone\" attribute.")
        }
    }

    wantFiltering := runner.questioner.AskBool(
        fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
            "from the %v topic? (y/n) ", queueName, topicName), "y")
    if wantFiltering {
        log.Println("You can filter messages by one or more of the following \"tone\" attributes.")

        var toneSelections []string
        askAboutTones := true

```

```

    for askAboutTones {
        toneIndex := runner.questioner.AskChoice(
            "Enter the number of the tone you want to filter by:\n", ToneChoices)
        toneSelections = append(toneSelections, ToneChoices[toneIndex])
        askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
the filter? (y/n) ", "y")
    }
    log.Printf("Your subscription will be filtered to only pass the following
tones: %v\n", toneSelections)
    filterPolicy = map[string][]string{TONE_KEY: toneSelections}
}
}

subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
filterPolicy)
if err != nil {
    panic(err)
}
log.Printf("The queue %v is now subscribed to the topic %v with the subscription
ARN %v.\n",
    queueName, topicName, subscriptionArn)

return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn
string, isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
    var message string
    var groupId string
    var dedupId string
    var toneSelection string
    publishMore := true
    for publishMore {
        groupId = ""
        dedupId = ""
        toneSelection = ""
        message = runner.questioner.Ask("Enter a message to publish: ")
        if isFifoTopic {
            log.Println("Because you are using a FIFO topic, you must set a message group
ID.\n" +
                "All messages within the same group will be received in the order they were
published.")
            groupId = runner.questioner.Ask("Enter a message group ID: ")
        }
        if !contentBasedDeduplication {

```

```

    log.Println("Because you are not using content-based deduplication,\n" +
        "you must enter a deduplication ID.")
    dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
}
}
if usingFilters {
    if runner.questioner.AskBool("Add a tone attribute so this message can be
filtered? (y/n) ", "y") {
        toneIndex := runner.questioner.AskChoice(
            "Enter the number of the tone you want to filter by:\n", ToneChoices)
        toneSelection = ToneChoices[toneIndex]
    }
}

err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId,
TONE_KEY, toneSelection)
if err != nil {
    panic(err)
}
log.Println(("Your message was published.))

publishMore = runner.questioner.AskBool("Do you want to publish another
message? (y/n) ", "y")
}
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
[]string) {
    log.Println("Polling queues for messages...")
    for _, queueUrl := range queueUrls {
        var messages []types.Message
        for {
            currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
            if err != nil {
                panic(err)
            }
            if len(currentMsgs) == 0 {
                break
            }
            messages = append(messages, currentMsgs...)
        }
        if len(messages) == 0 {
            log.Printf("No messages were received by queue %v.\n", queueUrl)
        } else if len(messages) == 1 {

```

```

    log.Printf("One message was received by queue %v:\n", queueUrl)

    } else {
        log.Printf("%v messages were received by queue %v:\n", len(messages),
            queueUrl)
    }
    for msgIndex, message := range messages {
        messageBody := MessageBody{}
        err := json.Unmarshal([]byte(*message.Body), &messageBody)
        if err != nil {
            panic(err)
        }
        log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
    }

    if len(messages) > 0 {
        log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
        err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
        if err != nil {
            panic(err)
        }
    }
}
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
// the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
// example.
// This package can be found in the ../../demotools folder of this repo.
func RunTopicsAndQueuesScenario(
    ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
    resources := Resources{}

```



```

defer func() {
    if r := recover(); r != nil {
        log.Println("Something went wrong with the demo.\n" +
            "Cleaning up any resources that were created...")
        resources.Cleanup(ctx)
    }
}()
queueCount := 2

log.Println(strings.Repeat("-", 88))
log.Printf("Welcome to messaging with topics and queues.\n\n"+
    "In this scenario, you will create an SNS topic and subscribe %v SQS queues to
the\n"+
    "topic. You can select from several options for configuring the topic and the
\n"+
    "subscriptions for the queues. You can then post to the topic and see the
results\n"+
    "in the queues.\n", queueCount)

log.Println(strings.Repeat("-", 88))

runner := ScenarioRunner{
    questioner: questioner,
    snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
    sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
}
resources.snsActor = runner.snsActor
resources.sqsActor = runner.sqsActor

topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
runner.CreateTopic(ctx)
resources.topicArn = topicArn
log.Println(strings.Repeat("-", 88))

log.Printf("Now you will create %v SQS queues and subscribe them to the topic.
\n", queueCount)
ordinals := []string{"first", "next"}
usingFilters := false
for _, ordinal := range ordinals {
    queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
    resources.queueUrls = append(resources.queueUrls, queueUrl)

    _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl,
topicName, topicArn, ordinal, isFifoTopic)

```

```

    usingFilters = usingFilters || filtering
}

log.Println(strings.Repeat("-", 88))
runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
usingFilters)
log.Println(strings.Repeat("-", 88))
runner.PollForMessages(ctx, resources.queueUrls)

log.Println(strings.Repeat("-", 88))

wantCleanup := questioner.AskBool("Do you want to remove all AWS resources
created for this scenario? (y/n) ", "y")
if wantCleanup {
    log.Println("Cleaning up resources...")
    resources.Cleanup(ctx)
}

log.Println(strings.Repeat("-", 88))
log.Println("Thanks for watching!")
log.Println(strings.Repeat("-", 88))
}

```

この例で使用されている Amazon SNS アクションをラップする構造体を定義します。

```

import (
    "context"
    "encoding/json"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/sns"
    "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
// actions
// used in the examples.
type SnsActions struct {
    SnsClient *sns.Client

```

```
}

// CreateTopic creates an Amazon SNS topic with the specified name. You can
// optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
// based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
    isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
    var topicArn string
    topicAttributes := map[string]string{}
    if isFifoTopic {
        topicAttributes["FifoTopic"] = "true"
    }
    if contentBasedDeduplication {
        topicAttributes["ContentBasedDeduplication"] = "true"
    }
    topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
        Name:      aws.String(topicName),
        Attributes: topicAttributes,
    })
    if err != nil {
        log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
    } else {
        topicArn = *topic.TopicArn
    }

    return topicArn, err
}

// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
    _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
        TopicArn: aws.String(topicArn)})
    if err != nil {
        log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
    }
    return err
}
```

```
// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to
// an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
// policy
// so that messages are only sent to the queue when the message has the specified
// attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
queueArn string, filterMap map[string][]string) (string, error) {
    var subscriptionArn string
    var attributes map[string]string
    if filterMap != nil {
        filterBytes, err := json.Marshal(filterMap)
        if err != nil {
            log.Printf("Couldn't create filter policy, here's why: %v\n", err)
            return "", err
        }
        attributes = map[string]string{"FilterPolicy": string(filterBytes)}
    }
    output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
        Protocol:          aws.String("sqs"),
        TopicArn:          aws.String(topicArn),
        Attributes:        attributes,
        Endpoint:          aws.String(queueArn),
        ReturnSubscriptionArn: true,
    })
    if err != nil {
        log.Printf("Couldn't subscribe queue %v to topic %v. Here's why: %v\n",
            queueArn, topicArn, err)
    } else {
        subscriptionArn = *output.SubscriptionArn
    }

    return subscriptionArn, err
}

// Publish publishes a message to an Amazon SNS topic. The message is then sent
// to all
// subscribers. When the topic is a FIFO topic, the message must also contain a
// group ID
```

```
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
// value
// filter attribute can be specified so that the message can be filtered
// according to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
string, groupId string, dedupId string, filterKey string, filterValue string)
error {
    publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
aws.String(message)}
    if groupId != "" {
        publishInput.MessageGroupId = aws.String(groupId)
    }
    if dedupId != "" {
        publishInput.MessageDeduplicationId = aws.String(dedupId)
    }
    if filterKey != "" && filterValue != "" {
        publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
            filterKey: {DataType: aws.String("String"), StringValue:
aws.String(filterValue)},
        }
    }
    _, err := actor.SnsClient.Publish(ctx, &publishInput)
    if err != nil {
        log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
err)
    }
    return err
}
```

この例で使用されている Amazon SQS アクションをラップする構造体を定義します。

```
import (
    "context"
    "encoding/json"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/service/sqs"
```

```
"github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
    SqsClient *sqs.Client
}

// CreateQueue creates an Amazon SQS queue with the specified name. You can
// specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
    isFifoQueue bool) (string, error) {
    var queueUrl string
    queueAttributes := map[string]string{}
    if isFifoQueue {
        queueAttributes["FifoQueue"] = "true"
    }
    queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
        QueueName:  aws.String(queueName),
        Attributes: queueAttributes,
    })
    if err != nil {
        log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
    } else {
        queueUrl = *queue.QueueUrl
    }

    return queueUrl, err
}

// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
// (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string)
    (string, error) {
    var queueArn string
    arnAttributeName := types.QueueAttributeNameQueueArn
```

```

attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
&sqs.GetQueueAttributesInput{
    QueueUrl:      aws.String(queueUrl),
    AttributeNames: []types.QueueAttributeName{arnAttributeName},
})
if err != nil {
    log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
} else {
    queueArn = attribute.Attributes[string(arnAttributeName)]
}
return queueArn, err
}

// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy
// to an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages
// to the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
string, queueArn string, topicArn string) error {
    policyDoc := PolicyDocument{
        Version: "2012-10-17",
        Statement: []PolicyStatement{{
            Effect:    "Allow",
            Action:  "sqs:SendMessage",
            Principal: map[string]string{"Service": "sns.amazonaws.com"},
            Resource: aws.String(queueArn),
            Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
topicArn}},
        }},
    }
    policyBytes, err := json.Marshal(policyDoc)
    if err != nil {
        log.Printf("Couldn't create policy document. Here's why: %v\n", err)
        return err
    }
    _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
        Attributes: map[string]string{
            string(types.QueueAttributeNamePolicy): string(policyBytes),
        },
        QueueUrl: aws.String(queueUrl),
    })
}

```

```
    if err != nil {
        log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
            queueUrl, err)
    }
    return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
    Version    string
    Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
    Effect    string
    Action    string
    Principal map[string]string `json:",omitempty"`
    Resource  *string             `json:",omitempty"`
    Condition PolicyCondition     `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string

// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
// queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
    maxMessages int32, waitTime int32) ([]types.Message, error) {
    var messages []types.Message
    result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
        QueueUrl:          aws.String(queueUrl),
        MaxNumberOfMessages: maxMessages,
        WaitTimeSeconds:    waitTime,
    })
    if err != nil {
        log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl,
            err)
    } else {
        messages = result.Messages
    }
}
```



```
    return messages, err
}

// DeleteMessages uses the DeleteMessageBatch action to delete a batch of
// messages from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
    messages []types.Message) error {
    entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
    for msgIndex := range messages {
        entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
        entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
    }
    _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
        Entries:  entries,
        QueueUrl: aws.String(queueUrl),
    })
    if err != nil {
        log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n",
            queueUrl, err)
    }
    return err
}

// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
    _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
        QueueUrl: aws.String(queueUrl)})
    if err != nil {
        log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
    }
    return err
}
```

リソースをクリーンアップします。

```
import (
    "context"
    "fmt"
    "log"
    "topics_and_queues/actions"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
    topicArn  string
    queueUrls []string
    snsActor  *actions.SnsActions
    sqsActor  *actions.SqsActions
}

// Cleanup deletes all AWS resources created during an example.
func (resources Resources) Cleanup(ctx context.Context) {
    defer func() {
        if r := recover(); r != nil {
            fmt.Println("Something went wrong during cleanup. Use the AWS Management
            Console\n" +
                "to remove any remaining resources that were created for this scenario.")
        }
    }()

    var err error
    if resources.topicArn != "" {
        log.Printf("Deleting topic %v.\n", resources.topicArn)
        err = resources.snsActor.DeleteTopic(ctx, resources.topicArn)
        if err != nil {
            panic(err)
        }
    }

    for _, queueUrl := range resources.queueUrls {
        log.Printf("Deleting queue %v.\n", queueUrl)
        err = resources.sqsActor.DeleteQueue(ctx, queueUrl)
        if err != nil {
            panic(err)
        }
    }
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の以下のトピックを参照してください。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

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。

```
package com.example.sns;

import
    software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sns.SnsClient;
import software.amazon.awssdk.services.sns.model.CreateTopicRequest;
import software.amazon.awssdk.services.sns.model.CreateTopicResponse;
import software.amazon.awssdk.services.sns.model.DeleteTopicRequest;
import software.amazon.awssdk.services.sns.model.DeleteTopicResponse;
import software.amazon.awssdk.services.sns.model.MessageAttributeValue;
```

```
import software.amazon.awssdk.services.sns.model.PublishRequest;
import software.amazon.awssdk.services.sns.model.PublishResponse;
import
    software.amazon.awssdk.services.sns.model.SetSubscriptionAttributesRequest;
import software.amazon.awssdk.services.sns.model.SnsException;
import software.amazon.awssdk.services.sns.model.SubscribeRequest;
import software.amazon.awssdk.services.sns.model.SubscribeResponse;
import software.amazon.awssdk.services.sns.model.UnsubscribeRequest;
import software.amazon.awssdk.services.sns.model.UnsubscribeResponse;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueAttributesResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SetQueueAttributesRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import com.google.gson.JsonPrimitive;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 * <p>
 * For more information, see the following documentation topic:
 * <p>
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 * <p>
```

```
* This Java example performs these tasks:
* <p>
* 1. Gives the user three options to choose from.
* 2. Creates an Amazon Simple Notification Service (Amazon SNS) topic.
* 3. Creates an Amazon Simple Queue Service (Amazon SQS) queue.
* 4. Gets the SQS queue Amazon Resource Name (ARN) attribute.
* 5. Attaches an AWS Identity and Access Management (IAM) policy to the queue.
* 6. Subscribes to the SQS queue.
* 7. Publishes a message to the topic.
* 8. Displays the messages.
* 9. Deletes the received message.
* 10. Unsubscribes from the topic.
* 11. Deletes the SNS topic.
*/
public class SNSWorkflow {
    public static final String DASHES = new String(new char[80]).replace("\0",
    "-");

    public static void main(String[] args) {
        final String usage = "\n" +
            "Usage:\n" +
            "    <fifoQueueARN>\n\n" +
            "Where:\n" +
            "    accountId - Your AWS account Id value.";

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        SnsClient snsClient = SnsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .build();

        SqsClient sqsClient = SqsClient.builder()
            .region(Region.US_EAST_1)
            .credentialsProvider(EnvironmentVariableCredentialsProvider.create())
            .build();

        Scanner in = new Scanner(System.in);
        String accountId = args[0];
        String useFIFO;
        String duplication = "n";
```

```
String topicName;
String deduplicationID = null;
String groupId = null;

String topicArn;
String sqsQueueName;
String sqsQueueUrl;
String sqsQueueArn;
String subscriptionArn;
boolean selectFIFO = false;

String message;
List<Message> messageList;
List<String> filterList = new ArrayList<>();
String msgAttValue = "";

System.out.println(DASHES);
System.out.println("Welcome to messaging with topics and queues.");
System.out.println("In this scenario, you will create an SNS topic and
subscribe an SQS queue to the topic.\n" +
    "You can select from several options for configuring the topic and
the subscriptions for the queue.\n" +
    "You can then post to the topic and see the results in the queue.");
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("SNS topics can be configured as FIFO (First-In-First-
Out).\n" +
    "FIFO topics deliver messages in order and support deduplication and
message filtering.\n" +
    "Would you like to work with FIFO topics? (y/n)");
useFIFO = in.nextLine();
if (useFIFO.compareTo("y") == 0) {
    selectFIFO = true;
    System.out.println("You have selected FIFO");
    System.out.println(" Because you have chosen a FIFO topic,
deduplication is supported.\n" +
        "          Deduplication IDs are either set in the message or
automatically generated from content using a hash function.\n"
        +
        "          If a message is successfully published to an SNS FIFO
topic, any message published and determined to have the same deduplication ID,
\n"
        +
```

```
        "        within the five-minute deduplication interval, is  
        accepted but not delivered.\n" +  
        "        For more information about deduplication, see https://  
docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html."));  
  
        System.out.println(  
            "Would you like to use content-based deduplication instead of  
entering a deduplication ID? (y/n)");  
        duplication = in.nextLine();  
        if (duplication.compareTo("y") == 0) {  
            System.out.println("Please enter a group id value");  
            groupId = in.nextLine();  
        } else {  
            System.out.println("Please enter deduplication Id value");  
            deduplicationID = in.nextLine();  
            System.out.println("Please enter a group id value");  
            groupId = in.nextLine();  
        }  
    }  
    System.out.println(DASHES);  
  
    System.out.println(DASHES);  
    System.out.println("2. Create a topic.");  
    System.out.println("Enter a name for your SNS topic.");  
    topicName = in.nextLine();  
    if (selectFIFO) {  
        System.out.println("Because you have selected a FIFO topic, '.fifo'  
must be appended to the topic name.");  
        topicName = topicName + ".fifo";  
        System.out.println("The name of the topic is " + topicName);  
        topicArn = createFIFO(snsClient, topicName, duplication);  
        System.out.println("The ARN of the FIFO topic is " + topicArn);  
    } else {  
        System.out.println("The name of the topic is " + topicName);  
        topicArn = createSNSTopic(snsClient, topicName);  
        System.out.println("The ARN of the non-FIFO topic is " + topicArn);  
    }  
    System.out.println(DASHES);  
  
    System.out.println(DASHES);  
    System.out.println("3. Create an SQS queue.");  
    System.out.println("Enter a name for your SQS queue.");
```

```

        sqsQueueName = in.nextLine();
        if (selectFIFO) {
            sqsQueueName = sqsQueueName + ".fifo";
        }
        sqsQueueUrl = createQueue(sqsClient, sqsQueueName, selectFIFO);
        System.out.println("The queue URL is " + sqsQueueUrl);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the SQS queue ARN attribute.");
        sqsQueueArn = getSqsQueueAttrs(sqsClient, sqsQueueUrl);
        System.out.println("The ARN of the new queue is " + sqsQueueArn);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Attach an IAM policy to the queue.");

        // Define the policy to use. Make sure that you change the REGION if you
are
        // running this code
        // in a different region.
        String policy = ""
        {
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "sns.amazonaws.com"
                    },
                    "Action": "sqs:SendMessage",
                    "Resource": "arn:aws:sqs:us-east-1:%s:%s",
                    "Condition": {
                        "ArnEquals": {
                            "aws:SourceArn": "arn:aws:sns:us-east-1:%s:%s"
                        }
                    }
                }
            ]
        }
        """".formatted(accountId, sqsQueueName, accountId, topicName);

        setQueueAttr(sqsClient, sqsQueueUrl, policy);
        System.out.println(DASHES);

```



```
System.out.println(DASHES);
System.out.println("6. Subscribe to the SQS queue.");
if (selectFIFO) {
    System.out.println(
        "If you add a filter to this subscription, then only the filtered
messages will be received in the queue.\n"
        +
        "For information about message filtering, see https://
docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n"
        +
        "For this example, you can filter messages by a \"tone\"
attribute.");
    System.out.println("Would you like to filter messages for " +
sqsQueueName + "'s subscription to the topic "
        + topicName + "? (y/n)");
    String filterAns = in.nextLine();
    if (filterAns.compareTo("y") == 0) {
        boolean moreAns = false;
        System.out.println("You can filter messages by one or more of the
following \"tone\" attributes.");
        System.out.println("1. cheerful");
        System.out.println("2. funny");
        System.out.println("3. serious");
        System.out.println("4. sincere");
        while (!moreAns) {
            System.out.println("Select a number or choose 0 to end.");
            String ans = in.nextLine();
            switch (ans) {
                case "1":
                    filterList.add("cheerful");
                    break;
                case "2":
                    filterList.add("funny");
                    break;
                case "3":
                    filterList.add("serious");
                    break;
                case "4":
                    filterList.add("sincere");
                    break;
                default:
                    moreAns = true;
                    break;
            }
        }
    }
}
```

```
        }
    }
}
subscriptionArn = subQueue(snsClient, topicArn, sqsQueueArn, filterList);
System.out.println(DASHES);

System.out.println(DASHES);
System.out.println("7. Publish a message to the topic.");
if (selectFIFO) {
    System.out.println("Would you like to add an attribute to this
message? (y/n)");
    String msgAns = in.nextLine();
    if (msgAns.compareTo("y") == 0) {
        System.out.println("You can filter messages by one or more of the
following \"tone\" attributes.");
        System.out.println("1. cheerful");
        System.out.println("2. funny");
        System.out.println("3. serious");
        System.out.println("4. sincere");
        System.out.println("Select a number or choose 0 to end.");
        String ans = in.nextLine();
        switch (ans) {
            case "1":
                msgAttValue = "cheerful";
                break;
            case "2":
                msgAttValue = "funny";
                break;
            case "3":
                msgAttValue = "serious";
                break;
            default:
                msgAttValue = "sincere";
                break;
        }

        System.out.println("Selected value is " + msgAttValue);
    }
    System.out.println("Enter a message.");
    message = in.nextLine();
    pubMessageFIFO(snsClient, message, topicArn, msgAttValue,
duplication, groupId, deduplicationID);

} else {
```

```
        System.out.println("Enter a message.");
        message = in.nextLine();
        pubMessage(snsClient, message, topicArn);
    }
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("8. Display the message. Press any key to continue.");
    in.nextLine();
    messageList = receiveMessages(sqsClient, sqsQueueUrl, msgAttValue);
    for (Message mes : messageList) {
        System.out.println("Message Id: " + mes.messageId());
        System.out.println("Full Message: " + mes.body());
    }
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("9. Delete the received message. Press any key to
continue.");
    in.nextLine();
    deleteMessages(sqsClient, sqsQueueUrl, messageList);
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("10. Unsubscribe from the topic and delete the queue.
Press any key to continue.");
    in.nextLine();
    unSub(snsClient, subscriptionArn);
    deleteSQSQueue(sqsClient, sqsQueueName);
    System.out.println(DASHES);

    System.out.println(DASHES);
    System.out.println("11. Delete the topic. Press any key to continue.");
    in.nextLine();
    deleteSNSTopic(snsClient, topicArn);

    System.out.println(DASHES);
    System.out.println("The SNS/SQS workflow has completed successfully.");
    System.out.println(DASHES);
}

public static void deleteSNSTopic(SnsClient snsClient, String topicArn) {
    try {
        DeleteTopicRequest request = DeleteTopicRequest.builder()
```

```
        .topicArn(topicArn)
        .build();

        DeleteTopicResponse result = snsClient.deleteTopic(request);
        System.out.println("Status was " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void deleteSQSQueue(SqsClient sqsClient, String queueName) {
    try {
        GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
            .queueName(queueName)
            .build();

        String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
        DeleteQueueRequest deleteQueueRequest = DeleteQueueRequest.builder()
            .queueUrl(queueUrl)
            .build();

        sqsClient.deleteQueue(deleteQueueRequest);
        System.out.println(queueName + " was successfully deleted.");

    } catch (SqsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void unSub(SnsClient snsClient, String subscriptionArn) {
    try {
        UnsubscribeRequest request = UnsubscribeRequest.builder()
            .subscriptionArn(subscriptionArn)
            .build();

        UnsubscribeResponse result = snsClient.unsubscribe(request);
        System.out.println("Status was " +
result.sdkHttpResponse().statusCode()
        + "\nSubscription was removed for " + request.subscriptionArn());
    }
```

```
        } catch (SnsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteMessages(SqsClient sqsClient, String queueUrl,
    List<Message> messages) {
        try {
            List<DeleteMessageBatchRequestEntry> entries = new ArrayList<>();
            for (Message msg : messages) {
                DeleteMessageBatchRequestEntry entry =
                DeleteMessageBatchRequestEntry.builder()
                    .id(msg.messageId())
                    .build();

                entries.add(entry);
            }

            DeleteMessageBatchRequest deleteMessageBatchRequest =
            DeleteMessageBatchRequest.builder()
                .queueUrl(queueUrl)
                .entries(entries)
                .build();

            sqsClient.deleteMessageBatch(deleteMessageBatchRequest);
            System.out.println("The batch delete of messages was successful");

        } catch (SqsException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static List<Message> receiveMessages(SqsClient sqsClient, String
    queueUrl, String msgAttValue) {
        try {
            if (msgAttValue.isEmpty()) {
                ReceiveMessageRequest receiveMessageRequest =
                ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .numberOfMessages(5)
                    .build();
```

```

        return
sqsClient.receiveMessage(receiveMessageRequest).messages();
    } else {
        // We know there are filters on the message.
        ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
            .queueUrl(queueUrl)
            .messageAttributeName(msgAttValue) // Include other message
attributes if needed.
            .maxNumberOfMessages(5)
            .build();

        return sqsClient.receiveMessage(receiveRequest).messages();
    }

} catch (SqsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}
return null;
}

public static void pubMessage(SnsClient snsClient, String message, String
topicArn) {
    try {
        PublishRequest request = PublishRequest.builder()
            .message(message)
            .topicArn(topicArn)
            .build();

        PublishResponse result = snsClient.publish(request);
        System.out
            .println(result.messageId() + " Message sent. Status is " +
result.sdkHttpResponse().statusCode());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static void pubMessageFIFO(SnsClient snsClient,
                                String message,
                                String topicArn,

```

```
        String msgAttValue,
        String duplication,
        String groupId,
        String deduplicationID) {

    try {
        PublishRequest request;
        // Means the user did not choose to use a message attribute.
        if (msgAttValue.isEmpty()) {
            if (duplication.compareTo("y") == 0) {
                request = PublishRequest.builder()
                    .message(message)
                    .messageGroupId(groupId)
                    .topicArn(topicArn)
                    .build();
            } else {
                request = PublishRequest.builder()
                    .message(message)
                    .messageDeduplicationId(deduplicationID)
                    .messageGroupId(groupId)
                    .topicArn(topicArn)
                    .build();
            }
        } else {
            Map<String, MessageAttributeValue> messageAttributes = new
HashMap<>();
            messageAttributes.put(msgAttValue,
MessageAttributeValue.builder()
                .dataType("String")
                .stringValue("true")
                .build());

            if (duplication.compareTo("y") == 0) {
                request = PublishRequest.builder()
                    .message(message)
                    .messageGroupId(groupId)
                    .topicArn(topicArn)
                    .build();
            } else {
                // Create a publish request with the message and attributes.
                request = PublishRequest.builder()
                    .topicArn(topicArn)
                    .message(message)
```

```
        .messageDeduplicationId(deduplicationID)
        .messageGroupId(groupId)
        .messageAttributes(messageAttributes)
        .build();
    }
}

// Publish the message to the topic.
PublishResponse result = snsClient.publish(request);
System.out
    .println(result.messageId() + " Message sent. Status was " +
result.sdkHttpResponse().statusCode());

} catch (SnsException e) {
    System.err.println(e.awsErrorDetails().errorMessage());
    System.exit(1);
}

}

// Subscribe to the SQS queue.
public static String subQueue(SnsClient snsClient, String topicArn, String
queueArn, List<String> filterList) {
    try {
        SubscribeRequest request;
        if (filterList.isEmpty()) {
            // No filter subscription is added.
            request = SubscribeRequest.builder()
                .protocol("sqs")
                .endpoint(queueArn)
                .returnSubscriptionArn(true)
                .topicArn(topicArn)
                .build();

            SubscribeResponse result = snsClient.subscribe(request);
            System.out.println("The queue " + queueArn + " has been
subscribed to the topic " + topicArn + "\n" +
                "with the subscription ARN " + result.subscriptionArn());
            return result.subscriptionArn();
        } else {
            request = SubscribeRequest.builder()
                .protocol("sqs")
                .endpoint(queueArn)
                .returnSubscriptionArn(true)
                .topicArn(topicArn)
```



```

        .build();

        SubscribeResponse result = snsClient.subscribe(request);
        System.out.println("The queue " + queueArn + " has been
subscribed to the topic " + topicArn + "\n" +
        "with the subscription ARN " + result.subscriptionArn());

        String attributeName = "FilterPolicy";
        Gson gson = new Gson();
        String jsonString = "{\"tone\": []}";
        JsonObject jsonObject = gson.fromJson(jsonString,
JsonObject.class);
        JsonArray toneArray = jsonObject.getAsJsonArray("tone");
        for (String value : filterList) {
            toneArray.add(new JsonPrimitive(value));
        }

        String updatedJsonString = gson.toJson(jsonObject);
        System.out.println(updatedJsonString);
        SetSubscriptionAttributesRequest attRequest =
SetSubscriptionAttributesRequest.builder()
            .subscriptionArn(result.subscriptionArn())
            .attributeName(attributeName)
            .attributeValue(updatedJsonString)
            .build();

        snsClient.setSubscriptionAttributes(attRequest);
        return result.subscriptionArn();
    }

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

// Attach a policy to the queue.
public static void setQueueAttr(SqsClient sqsClient, String queueUrl, String
policy) {
    try {
        Map<software.amazon.awssdk.services.sqs.model.QueueAttributeName,
String> attrMap = new HashMap<>();
        attrMap.put(QueueAttributeName.POLICY, policy);
    }

```

```
        SetQueueAttributesRequest attributesRequest =
SetQueueAttributesRequest.builder()
        .queueUrl(queueUrl)
        .attributes(attrMap)
        .build();

        sqsClient.setQueueAttributes(attributesRequest);
        System.out.println("The policy has been successfully attached.");

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}

public static String getSQSQueueAttrs(SqsClient sqsClient, String queueUrl) {
    // Specify the attributes to retrieve.
    List<QueueAttributeName> atts = new ArrayList<>();
    atts.add(QueueAttributeName.QUEUE_ARN);

    GetQueueAttributesRequest attributesRequest =
GetQueueAttributesRequest.builder()
        .queueUrl(queueUrl)
        .attributeNames(atts)
        .build();

    GetQueueAttributesResponse response =
sqsClient.getQueueAttributes(attributesRequest);
    Map<String, String> queueAtts = response.attributesAsStrings();
    for (Map.Entry<String, String> queueAtt : queueAtts.entrySet())
        return queueAtt.getValue();

    return "";
}

public static String createQueue(SqsClient sqsClient, String queueName,
Boolean selectFIFO) {
    try {
        System.out.println("\nCreate Queue");
        if (selectFIFO) {
            Map<QueueAttributeName, String> attrs = new HashMap<>();
            attrs.put(QueueAttributeName.FIFO_QUEUE, "true");
```

```
        CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                    .queueName(queueName)
                    .attributes(attrs)
                    .build();

        sqsClient.createQueue(createQueueRequest);
        System.out.println("\nGet queue url");
        GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
        return getQueueUrlResponse.queueUrl();
    } else {
        CreateQueueRequest createQueueRequest =
CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();

        sqsClient.createQueue(createQueueRequest);
        System.out.println("\nGet queue url");
        GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
        return getQueueUrlResponse.queueUrl();
    }

    } catch (SqsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

public static String createSNSTopic(SnsClient snsClient, String topicName) {
    CreateTopicResponse result;
    try {
        CreateTopicRequest request = CreateTopicRequest.builder()
            .name(topicName)
            .build();

        result = snsClient.createTopic(request);
        return result.topicArn();

    } catch (SnsException e) {
```

```
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}

public static String createFIFO(SnsClient snsClient, String topicName, String
duplication) {
    try {
        // Create a FIFO topic by using the SNS service client.
        Map<String, String> topicAttributes = new HashMap<>();
        if (duplication.compareTo("n") == 0) {
            topicAttributes.put("FifoTopic", "true");
            topicAttributes.put("ContentBasedDeduplication", "false");
        } else {
            topicAttributes.put("FifoTopic", "true");
            topicAttributes.put("ContentBasedDeduplication", "true");
        }

        CreateTopicRequest topicRequest = CreateTopicRequest.builder()
            .name(topicName)
            .attributes(topicAttributes)
            .build();

        CreateTopicResponse response = snsClient.createTopic(topicRequest);
        return response.topicArn();

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
    return "";
}
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)

- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Publish](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

JavaScript

SDK for JavaScript (v3)

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。

これがこのシナリオのエントリーポイントです。

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

上記のコードにより必要な依存関係が提供され、シナリオが開始されます。次のセクションには、この例の大部分が含まれています。

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
string }[]}
   */
  queues = [];
  prompter;

  /**
   * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
   * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
   * @param {import('../libs/prompter.js').Prompter} prompter
   * @param {import('../libs/logger.js').Logger} logger
   */
  constructor(snsClient, sqsClient, prompter, logger) {
    this.snsClient = snsClient;
    this.sqsClient = sqsClient;
    this.prompter = prompter;
    this.logger = logger;
  }
}
```

```
async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
  }

  const response = await this.snsClient.send(
    new CreateTopicCommand({
      Name: this.topicName,
      Attributes: {
        FifoTopic: this.isFifo ? "true" : "false",
        ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
      },
    }),
  );

  this.topicArn = response.TopicArn;

  await this.logger.log(
```

```
    MESSAGES.topicCreatedNotice
      .replace("${TOPIC_NAME}", this.topicName)
      .replace("${TOPIC_ARN}", this.topicArn),
    );
  }

  async createQueues() {
    await this.logger.log(MESSAGES.createQueuesNotice);
    // Increase this number to add more queues.
    const maxQueues = 2;

    for (let i = 0; i < maxQueues; i++) {
      await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
      let queueName = await this.prompter.input({
        message: MESSAGES.queueNamePrompt.replace(
          "${EXAMPLE_NAME}",
          i === 0 ? "good-news" : "bad-news",
        ),
      });

      if (this.isFifo) {
        queueName += ".fifo";
        await this.logger.log(MESSAGES.appendFifoNotice);
      }

      const response = await this.sqsClient.send(
        new CreateQueueCommand({
          QueueName: queueName,
          Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
        }),
      );

      const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
          QueueUrl: response.QueueUrl,
          AttributeNames: ["QueueArn"],
        }),
      );

      this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
      });
    }
  }
}
```



```
    await this.logger.log(
      MESSAGES.queueCreatedNotice
        .replace("${QUEUE_NAME}", queueName)
        .replace("${QUEUE_URL}", response.QueueUrl)
        .replace("${QUEUE_ARN}", Attributes.QueueArn),
    );
  }
}

async attachQueueIamPolicies() {
  for (const [index, queue] of this.queues.entries()) {
    const policy = JSON.stringify(
      {
        Statement: [
          {
            Effect: "Allow",
            Principal: {
              Service: "sns.amazonaws.com",
            },
            Action: "sqs:SendMessage",
            Resource: queue.queueArn,
            Condition: {
              ArnEquals: {
                "aws:SourceArn": this.topicArn,
              },
            },
          },
        ],
      },
      null,
      2,
    );

    if (index !== 0) {
      this.logger.logSeparator();
    }

    await this.logger.log(MESSAGES.attachPolicyNotice);
    console.log(policy);
    const addPolicy = await this.prompter.confirm({
      message: MESSAGES.addPolicyConfirmation.replace(
        "${QUEUE_NAME}",
        queue.queueName,
      ),
    });
  }
}
```

```

    ),
  });

  if (addPolicy) {
    await this.sqsClient.send(
      new SetQueueAttributesCommand({
        QueueUrl: queue.queueUrl,
        Attributes: {
          Policy: policy,
        },
      }),
    );
    queue.policy = policy;
  } else {
    await this.logger.log(
      MESSAGES.policyNotAttachedNotice.replace(
        "${QUEUE_NAME}",
        queue.queueName,
      ),
    );
  }
}

async subscribeQueuesToTopic() {
  for (const [index, queue] of this.queues.entries()) {
    /**
     * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
     */
    const subscribeParams = {
      TopicArn: this.topicArn,
      Protocol: "sqs",
      Endpoint: queue.queueArn,
    };
    let tones = [];

    if (this.isFifo) {
      if (index === 0) {
        await this.logger.log(MESSAGES.fifoFilterNotice);
      }
      tones = await this.prompter.checkbox({
        message: MESSAGES.fifoFilterSelect.replace(
          "${QUEUE_NAME}",
          queue.queueName,

```

```
    ),
    choices: toneChoices,
  });

  if (tones.length) {
    subscribeParams.Attributes = {
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        tone: tones,
      }),
    };
  }
}

const { SubscriptionArn } = await this.snsClient.send(
  new SubscribeCommand(subscribeParams),
);

this.subscriptionArns.push(SubscriptionArn);

await this.logger.log(
  MESSAGES.queueSubscribedNotice
    .replace("${QUEUE_NAME}", queue.queueName)
    .replace("${TOPIC_NAME}", this.topicName)
    .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
);
}
}

async publishMessages() {
  const message = await this.prompter.input({
    message: MESSAGES.publishMessagePrompt,
  });

  let groupId;
  let deduplicationId;
  let choices;

  if (this.isFifo) {
    await this.logger.log(MESSAGES.groupIdNotice);
    groupId = await this.prompter.input({
      message: MESSAGES.groupIdPrompt,
    });
  }
```

```
    if (this.autoDedup === false) {
      await this.logger.log(MESSAGES.deduplicationIdNotice);
      deduplicationId = await this.prompter.input({
        message: MESSAGES.deduplicationIdPrompt,
      });
    }

    choices = await this.prompter.checkbox({
      message: MESSAGES.messageAttributesPrompt,
      choices: toneChoices,
    });
  }

  await this.snsClient.send(
    new PublishCommand({
      TopicArn: this.topicArn,
      Message: message,
      ...(groupId
        ? {
            MessageGroupId: groupId,
          }
        : {}),
      ...(deduplicationId
        ? {
            MessageDeduplicationId: deduplicationId,
          }
        : {}),
      ...(choices
        ? {
            MessageAttributes: {
              tone: {
                DataType: "String.Array",
                StringValue: JSON.stringify(choices),
              },
            },
          }
        : {}),
    })),
  );

  const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
  });
```

```
    if (publishAnother) {
      await this.publishMessages();
    }
  }

  async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
      const { Messages } = await this.sqsClient.send(
        new ReceiveMessageCommand({
          QueueUrl: queue.queueUrl,
        }),
      );

      if (Messages) {
        await this.logger.log(
          MESSAGES.messagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
        console.log(Messages);

        await this.sqsClient.send(
          new DeleteMessageBatchCommand({
            QueueUrl: queue.queueUrl,
            Entries: Messages.map((message) => ({
              Id: message.MessageId,
              ReceiptHandle: message.ReceiptHandle,
            })),
          }),
        );
      } else {
        await this.logger.log(
          MESSAGES.noMessagesReceivedNotice.replace(
            "${QUEUE_NAME}",
            queue.queueName,
          ),
        );
      }
    }
  }

  const deleteAndPoll = await this.prompter.confirm({
    message: MESSAGES.deleteAndPollConfirmation,
  });
```

```
    if (deleteAndPoll) {
      await this.receiveAndDeleteMessages();
    }
  }

  async destroyResources() {
    for (const subscriptionArn of this.subscriptionArns) {
      await this.snsClient.send(
        new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
      );
    }

    for (const queue of this.queues) {
      await this.sqsClient.send(
        new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
      );
    }

    if (this.topicArn) {
      await this.snsClient.send(
        new DeleteTopicCommand({ TopicArn: this.topicArn }),
      );
    }
  }

  async start() {
    console.clear();

    try {
      this.logger.logSeparator(MESSAGES.headerWelcome);
      await this.welcome();
      this.logger.logSeparator(MESSAGES.headerFifo);
      await this.confirmFifo();
      this.logger.logSeparator(MESSAGES.headerCreateTopic);
      await this.createTopic();
      this.logger.logSeparator(MESSAGES.headerCreateQueues);
      await this.createQueues();
      this.logger.logSeparator(MESSAGES.headerAttachPolicy);
      await this.attachQueueIamPolicies();
      this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
      await this.subscribeQueuesToTopic();
      this.logger.logSeparator(MESSAGES.headerPublishMessage);
      await this.publishMessages();
    }
  }
}
```

```
        this.logger.logSeparator(MESSAGES.headerReceiveMessages);
        await this.receiveAndDeleteMessages();
    } catch (err) {
        console.error(err);
    } finally {
        await this.destroyResources();
    }
}
}
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の以下のトピックを参照してください。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

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。

```
package com.example.sns
```

```
import aws.sdk.kotlin.services.sns.SnsClient
import aws.sdk.kotlin.services.sns.model.CreateTopicRequest
import aws.sdk.kotlin.services.sns.model.DeleteTopicRequest
import aws.sdk.kotlin.services.sns.model.PublishRequest
import aws.sdk.kotlin.services.sns.model.SetSubscriptionAttributesRequest
import aws.sdk.kotlin.services.sns.model.SubscribeRequest
import aws.sdk.kotlin.services.sns.model.UnsubscribeRequest
import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.model.CreateQueueRequest
import aws.sdk.kotlin.services.sqs.model.DeleteMessageBatchRequest
import aws.sdk.kotlin.services.sqs.model.DeleteMessageBatchRequestEntry
import aws.sdk.kotlin.services.sqs.model.DeleteQueueRequest
import aws.sdk.kotlin.services.sqs.model.GetQueueAttributesRequest
import aws.sdk.kotlin.services.sqs.model.GetQueueUrlRequest
import aws.sdk.kotlin.services.sqs.model.Message
import aws.sdk.kotlin.services.sqs.model.QueueAttributeName
import aws.sdk.kotlin.services.sqs.model.ReceiveMessageRequest
import aws.sdk.kotlin.services.sqs.model.SetQueueAttributesRequest
import com.google.gson.Gson
import com.google.gson.JsonObject
import com.google.gson.JsonPrimitive
import java.util.Scanner

/**
Before running this Kotlin code example, set up your development environment,
including your AWS credentials.

For more information, see the following documentation topic:
https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This Kotlin example performs the following tasks:

1. Gives the user three options to choose from.
2. Creates an Amazon Simple Notification Service (Amazon SNS) topic.
3. Creates an Amazon Simple Queue Service (Amazon SQS) queue.
4. Gets the SQS queue Amazon Resource Name (ARN) attribute.
5. Attaches an AWS Identity and Access Management (IAM) policy to the queue.
6. Subscribes to the SQS queue.
7. Publishes a message to the topic.
8. Displays the messages.
9. Deletes the received message.
10. Unsubscribes from the topic.
11. Deletes the SNS topic.
*/
```



```

val DASHES: String = String(CharArray(80)).replace("\u0000", "-")
suspend fun main() {
    val input = Scanner(System.`in`)
    val useFIFO: String
    var duplication = "n"
    var topicName: String
    var deduplicationID: String? = null
    var groupId: String? = null
    val topicArn: String?
    var sqsQueueName: String
    val sqsQueueUrl: String?
    val sqsQueueArn: String
    val subscriptionArn: String?
    var selectFIFO = false
    val message: String
    val messageList: List<Message?>?
    val filterList = ArrayList<String>()
    var msgAttValue = ""

    println(DASHES)
    println("Welcome to the AWS SDK for Kotlin messaging with topics and
queues.")
    println(
        """
            In this scenario, you will create an SNS topic and subscribe an
            SQS queue to the topic.
            You can select from several options for configuring the topic and
            the subscriptions for the queue.
            You can then post to the topic and see the results in the queue.
        """).trimIndent(),
    )
    println(DASHES)

    println(DASHES)
    println(
        """
            SNS topics can be configured as FIFO (First-In-First-Out).
            FIFO topics deliver messages in order and support deduplication
            and message filtering.
            Would you like to work with FIFO topics? (y/n)
        """).trimIndent(),
    )
    useFIFO = input.nextLine()

```

```

    if (useFIFO.compareTo("y") == 0) {
        selectFIFO = true
        println("You have selected FIFO")
        println(
            """" Because you have chosen a FIFO topic, deduplication is supported.
            Deduplication IDs are either set in the message or automatically
            generated from content using a hash function.
            If a message is successfully published to an SNS FIFO topic, any message
            published and determined to have the same deduplication ID,
            within the five-minute deduplication interval, is accepted but not
            delivered.
            For more information about deduplication, see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.""",
        )

        println("Would you like to use content-based deduplication instead of
        entering a deduplication ID? (y/n)")
        duplication = input.nextLine()
        if (duplication.compareTo("y") == 0) {
            println("Enter a group id value")
            groupId = input.nextLine()
        } else {
            println("Enter deduplication Id value")
            deduplicationID = input.nextLine()
            println("Enter a group id value")
            groupId = input.nextLine()
        }
    }
    println(DASHES)

    println(DASHES)
    println("2. Create a topic.")
    println("Enter a name for your SNS topic.")
    topicName = input.nextLine()
    if (selectFIFO) {
        println("Because you have selected a FIFO topic, '.fifo' must be appended
        to the topic name.")
        topicName = "$topicName.fifo"
        println("The name of the topic is $topicName")
        topicArn = createFIFO(topicName, duplication)
        println("The ARN of the FIFO topic is $topicArn")
    } else {
        println("The name of the topic is $topicName")
        topicArn = createSNSTopic(topicName)
    }

```

```
        println("The ARN of the non-FIFO topic is $topicArn")
    }
    println(DASHES)

    println(DASHES)
    println("3. Create an SQS queue.")
    println("Enter a name for your SQS queue.")
    sqsQueueName = input.nextLine()
    if (selectFIFO) {
        sqsQueueName = "$sqsQueueName.fifo"
    }
    sqsQueueUrl = createQueue(sqsQueueName, selectFIFO)
    println("The queue URL is $sqsQueueUrl")
    println(DASHES)

    println(DASHES)
    println("4. Get the SQS queue ARN attribute.")
    sqsQueueArn = getSqsQueueAttrs(sqsQueueUrl)
    println("The ARN of the new queue is $sqsQueueArn")
    println(DASHES)

    println(DASHES)
    println("5. Attach an IAM policy to the queue.")
    // Define the policy to use.
    val policy = """{
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "sns.amazonaws.com"
                },
                "Action": "sqs:SendMessage",
                "Resource": "$sqsQueueArn",
                "Condition": {
                    "ArnEquals": {
                        "aws:SourceArn": "$topicArn"
                    }
                }
            }
        ]
    }"""
    setQueueAttr(sqsQueueUrl, policy)
    println(DASHES)
```

```

println(DASHES)
println("6. Subscribe to the SQS queue.")
if (selectFIFO) {
    println(
        """"If you add a filter to this subscription, then only the filtered
        messages will be received in the queue.
        For information about message filtering, see https://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html
        For this example, you can filter messages by a "tone" attribute.""",
    )
    println("Would you like to filter messages for $sqsQueueName's
    subscription to the topic $topicName? (y/n)")
    val filterAns: String = input.nextLine()
    if (filterAns.compareTo("y") == 0) {
        var moreAns = false
        println("You can filter messages by using one or more of the
        following \"tone\" attributes.")
        println("1. cheerful")
        println("2. funny")
        println("3. serious")
        println("4. sincere")
        while (!moreAns) {
            println("Select a number or choose 0 to end.")
            val ans: String = input.nextLine()
            when (ans) {
                "1" -> filterList.add("cheerful")
                "2" -> filterList.add("funny")
                "3" -> filterList.add("serious")
                "4" -> filterList.add("sincere")
                else -> moreAns = true
            }
        }
    }
}
subscriptionArn = subQueue(topicArn, sqsQueueArn, filterList)
println(DASHES)

println(DASHES)
println("7. Publish a message to the topic.")
if (selectFIFO) {
    println("Would you like to add an attribute to this message? (y/n)")
    val msgAns: String = input.nextLine()
    if (msgAns.compareTo("y") == 0) {

```

```
println("You can filter messages by one or more of the following
\"tone\" attributes.")
println("1. cheerful")
println("2. funny")
println("3. serious")
println("4. sincere")
println("Select a number or choose 0 to end.")
val ans: String = input.nextLine()
msgAttValue = when (ans) {
    "1" -> "cheerful"
    "2" -> "funny"
    "3" -> "serious"
    else -> "sincere"
}
println("Selected value is $msgAttValue")
}
println("Enter a message.")
message = input.nextLine()
pubMessageFIFO(message, topicArn, msgAttValue, duplication, groupId,
deduplicationID)
} else {
    println("Enter a message.")
    message = input.nextLine()
    pubMessage(message, topicArn)
}
println(DASHES)

println(DASHES)
println("8. Display the message. Press any key to continue.")
input.nextLine()
messageList = receiveMessages(sqsQueueUrl, msgAttValue)
if (messageList != null) {
    for (mes in messageList) {
        println("Message Id: ${mes.messageId}")
        println("Full Message: ${mes.body}")
    }
}
println(DASHES)

println(DASHES)
println("9. Delete the received message. Press any key to continue.")
input.nextLine()
if (messageList != null) {
    deleteMessages(sqsQueueUrl, messageList)
```

```
    }
    println(DASHES)

    println(DASHES)
    println("10. Unsubscribe from the topic and delete the queue. Press any key
to continue.")
    input.nextLine()
    unSub(subscriptionArn)
    deleteSQSQueue(sqsQueueName)
    println(DASHES)

    println(DASHES)
    println("11. Delete the topic. Press any key to continue.")
    input.nextLine()
    deleteSNSTopic(topicArn)
    println(DASHES)

    println(DASHES)
    println("The SNS/SQS workflow has completed successfully.")
    println(DASHES)
}

suspend fun deleteSNSTopic(topicArnVal: String?) {
    val request = DeleteTopicRequest {
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.deleteTopic(request)
        println("$topicArnVal was deleted")
    }
}

suspend fun deleteSQSQueue(queueNameVal: String) {
    val getQueueRequest = GetQueueUrlRequest {
        queueName = queueNameVal
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val queueUrlVal = sqsClient.getQueueUrl(getQueueRequest).queueUrl
        val deleteQueueRequest = DeleteQueueRequest {
            queueUrl = queueUrlVal
        }
    }
}
```

```
sqsClient.deleteQueue(deleteQueueRequest)
println("$queueNameVal was successfully deleted.")
}
}

suspend fun unSub(subscripArn: String?) {
    val request = UnsubscribeRequest {
        subscriptionArn = subscripArn
    }
    SnsClient { region = "us-east-1" }.use { snsClient ->
        snsClient.unsubscribe(request)
        println("Subscription was removed for $subscripArn")
    }
}

suspend fun deleteMessages(queueUrlVal: String?, messages: List<Message>) {
    val entriesVal: MutableList<DeleteMessageBatchRequestEntry> = mutableListOf()
    for (msg in messages) {
        val entry = DeleteMessageBatchRequestEntry {
            id = msg.messageId
        }
        entriesVal.add(entry)
    }

    val deleteMessageBatchRequest = DeleteMessageBatchRequest {
        queueUrl = queueUrlVal
        entries = entriesVal
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteMessageBatch(deleteMessageBatchRequest)
        println("The batch delete of messages was successful")
    }
}

suspend fun receiveMessages(queueUrlVal: String?, msgAttValue: String):
List<Message>? {
    if (msgAttValue.isEmpty()) {
        val request = ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }
        SqsClient { region = "us-east-1" }.use { sqsClient ->
            return sqsClient.receiveMessage(request).messages
        }
    }
}
```

```
    }
  } else {
    val receiveRequest = ReceiveMessageRequest {
      queueUrl = queueUrlVal
      waitTimeSeconds = 1
      maxNumberOfMessages = 5
    }
    SqsClient { region = "us-east-1" }.use { sqsClient ->
      return sqsClient.receiveMessage(receiveRequest).messages
    }
  }
}

suspend fun pubMessage(messageVal: String?, topicArnVal: String?) {
  val request = PublishRequest {
    message = messageVal
    topicArn = topicArnVal
  }

  SnsClient { region = "us-east-1" }.use { snsClient ->
    val result = snsClient.publish(request)
    println("${result.messageId} message sent.")
  }
}

suspend fun pubMessageFIFO(
  messageVal: String?,
  topicArnVal: String?,
  msgAttValue: String,
  duplication: String,
  groupIdVal: String?,
  deduplicationID: String?,
) {
  // Means the user did not choose to use a message attribute.
  if (msgAttValue.isEmpty()) {
    if (duplication.compareTo("y") == 0) {
      val request = PublishRequest {
        message = messageVal
        messageGroupId = groupIdVal
        topicArn = topicArnVal
      }

      SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
      }
    }
  }
}
```



```
        println(result.messageId.toString() + " Message sent.")
    }
} else {
    val request = PublishRequest {
        message = messageVal
        messageDeduplicationId = deduplicationID
        messageGroupId = groupIdVal
        topicArn = topicArnVal
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.publish(request)
        println(result.messageId.toString() + " Message sent.")
    }
}
} else {
    val messAttr = aws.sdk.kotlin.services.sns.model.MessageAttributeValue {
        dataType = "String"
        stringValue = "true"
    }

    val mapAtt: Map<String,
aws.sdk.kotlin.services.sns.model.MessageAttributeValue> =
        mapOf(msgAttValue to messAttr)
    if (duplication.compareTo("y") == 0) {
        val request = PublishRequest {
            message = messageVal
            messageGroupId = groupIdVal
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.publish(request)
            println(result.messageId.toString() + " Message sent.")
        }
    } else {
        // Create a publish request with the message and attributes.
        val request = PublishRequest {
            topicArn = topicArnVal
            message = messageVal
            messageDeduplicationId = deduplicationID
            messageGroupId = groupIdVal
            messageAttributes = mapAtt
        }
    }
}
```

```

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.publish(request)
            println(result.messageId.toString() + " Message sent.")
        }
    }
}

// Subscribe to the SQS queue.
suspend fun subQueue(topicArnVal: String?, queueArnVal: String, filterList:
List<String?>): String? {
    val request: SubscribeRequest
    if (filterList.isEmpty()) {
        // No filter subscription is added.
        request = SubscribeRequest {
            protocol = "sqs"
            endpoint = queueArnVal
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.subscribe(request)
            println(
                "The queue " + queueArnVal + " has been subscribed to the topic "
+ topicArnVal + "\n" +
                "with the subscription ARN " + result.subscriptionArn,
            )
            return result.subscriptionArn
        }
    } else {
        request = SubscribeRequest {
            protocol = "sqs"
            endpoint = queueArnVal
            returnSubscriptionArn = true
            topicArn = topicArnVal
        }

        SnsClient { region = "us-east-1" }.use { snsClient ->
            val result = snsClient.subscribe(request)
            println("The queue $queueArnVal has been subscribed to the topic
$topicArnVal with the subscription ARN ${result.subscriptionArn}")
        }
    }
}

```

```

        val attributeNameVal = "FilterPolicy"
        val gson = Gson()
        val jsonString = "{\"tone\": []}"
        val jsonObject = gson.fromJson(jsonString, JsonObject::class.java)
        val toneArray = jsonObject.getAsJsonArray("tone")
        for (value: String? in filterList) {
            toneArray.add(JsonPrimitive(value))
        }

        val updatedJsonString: String = gson.toJson(jsonObject)
        println(updatedJsonString)
        val attRequest = SetSubscriptionAttributesRequest {
            subscriptionArn = result.subscriptionArn
            attributeName = attributeNameVal
            attributeValue = updatedJsonString
        }

        snsClient.setSubscriptionAttributes(attRequest)
        return result.subscriptionArn
    }
}

suspend fun setQueueAttr(queueUrlVal: String?, policy: String) {
    val attrMap: MutableMap<String, String> = HashMap()
    attrMap[QueueAttributeName.Policy.toString()] = policy

    val attributesRequest = SetQueueAttributesRequest {
        queueUrl = queueUrlVal
        attributes = attrMap
    }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        sqsClient.setQueueAttributes(attributesRequest)
        println("The policy has been successfully attached.")
    }
}

suspend fun getSqsQueueAttrs(queueUrlVal: String?): String {
    val atts: MutableList<QueueAttributeName> = ArrayList()
    atts.add(QueueAttributeName.QueueArn)

    val attributesRequest = GetQueueAttributesRequest {
        queueUrl = queueUrlVal
    }

```

```
        attributeNames = atts
    }
    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.getQueueAttributes(attributesRequest)
        val mapAtts = response.attributes
        if (mapAtts != null) {
            mapAtts.forEach { entry ->
                println("${entry.key} : ${entry.value}")
                return entry.value
            }
        }
    }
    return ""
}

suspend fun createQueue(queueNameVal: String?, selectFIFO: Boolean): String? {
    println("\nCreate Queue")
    if (selectFIFO) {
        val attrs = mutableMapOf<String, String>()
        attrs[QueueAttributeName.FifoQueue.toString()] = "true"

        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
            attributes = attrs
        }

        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
            println("\nGet queue url")

            val urlRequest = GetQueueUrlRequest {
                queueName = queueNameVal
            }

            val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
            return getQueueUrlResponse.queueUrl
        }
    } else {
        val createQueueRequest = CreateQueueRequest {
            queueName = queueNameVal
        }

        SqsClient { region = "us-east-1" }.use { sqsClient ->
            sqsClient.createQueue(createQueueRequest)
        }
    }
}
```

```
        println("Get queue url")

        val urlRequest = GetQueueUrlRequest {
            queueName = queueNameVal
        }

        val getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest)
        return getQueueUrlResponse.queueUrl
    }
}

suspend fun createSNSTopic(topicName: String?): String? {
    val request = CreateTopicRequest {
        name = topicName
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val result = snsClient.createTopic(request)
        return result.topicArn
    }
}

suspend fun createFIFO(topicName: String?, duplication: String): String? {
    val topicAttributes: MutableMap<String, String> = HashMap()
    if (duplication.compareTo("n") == 0) {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "false"
    } else {
        topicAttributes["FifoTopic"] = "true"
        topicAttributes["ContentBasedDeduplication"] = "true"
    }

    val topicRequest = CreateTopicRequest {
        name = topicName
        attributes = topicAttributes
    }

    SnsClient { region = "us-east-1" }.use { snsClient ->
        val response = snsClient.createTopic(topicRequest)
        return response.topicArn
    }
}
```

- API の詳細については、『AWS SDK for Kotlin API リファレンス』の以下のトピックを参照してください。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

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。

コマンドプロンプトからインタラクティブなシナリオを実行します。

```
class TopicsAndQueuesScenario:
    """Manages the Topics and Queues feature scenario."""

    DASHES = "-" * 80

    def __init__(self, sns_wrapper: SnsWrapper, sqs_wrapper: SqsWrapper) -> None:
        """
        Initialize the Topics and Queues scenario.

        :param sns_wrapper: SnsWrapper instance for SNS operations.
        :param sqs_wrapper: SqsWrapper instance for SQS operations.
```

```
"""
self.sns_wrapper = sns_wrapper
self.sqs_wrapper = sqs_wrapper

# Scenario state
self.use_fifo_topic = False
self.use_content_based_deduplication = False
self.topic_name = None
self.topic_arn = None
self.queue_count = 2
self.queue_urls = []
self.subscription_arns = []
self.tones = ["cheerful", "funny", "serious", "sincere"]

def run_scenario(self) -> None:
    """Run the Topics and Queues feature scenario."""
    print(self.DASHES)
    print("Welcome to messaging with topics and queues.")
    print(self.DASHES)
    print(f"""
    In this scenario, you will create an SNS topic and subscribe
    {self.queue_count} SQS queues to the topic.
    You can select from several options for configuring the topic and the
    subscriptions for the queues.
    You can then post to the topic and see the results in the queues.
    """)

    try:
        # Setup Phase
        print(self.DASHES)
        self._setup_topic()
        print(self.DASHES)

        self._setup_queues()
        print(self.DASHES)

        # Demonstration Phase
        self._publish_messages()
        print(self.DASHES)

        # Examination Phase
        self._poll_queues_for_messages()
        print(self.DASHES)
```

```

        # Cleanup Phase
        self._cleanup_resources()
        print(self.DASHES)

    except Exception as e:
        logger.error(f"Scenario failed: {e}")
        print(f"There was a problem with the scenario: {e}")
        print("\nInitiating cleanup...")
        try:
            self._cleanup_resources()
        except Exception as cleanup_error:
            logger.error(f"Error during cleanup: {cleanup_error}")

    print("Messaging with topics and queues scenario is complete.")
    print(self.DASHES)

def _setup_topic(self) -> None:
    """Set up the SNS topic to be used with the queues."""
    print("SNS topics can be configured as FIFO (First-In-First-Out).")
    print("FIFO topics deliver messages in order and support deduplication
and message filtering.")
    print()

    self.use_fifo_topic = q.ask("Would you like to work with FIFO topics? (y/
n): ", q.is_yesno)

    if self.use_fifo_topic:
        print(self.DASHES)
        self.topic_name = q.ask("Enter a name for your SNS topic: ",
q.non_empty)
        print("Because you have selected a FIFO topic, '.fifo' must be
appended to the topic name.")
        print()

        print(self.DASHES)
        print("""
Because you have chosen a FIFO topic, deduplication is supported.
Deduplication IDs are either set in the message or automatically generated
from content using a hash function.

If a message is successfully published to an SNS FIFO topic, any message
published and determined to have the same deduplication ID,
within the five-minute deduplication interval, is accepted but not delivered.

```



```

For more information about deduplication,
see https://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.
    """

    self.use_content_based_deduplication = q.ask(
        "Use content-based deduplication instead of entering a
deduplication ID? (y/n): ",
        q.is_yesno
    )
    else:
        self.topic_name = q.ask("Enter a name for your SNS topic: ",
q.non_empty)

    print(self.DASHES)

    # Create the topic
    self.topic_arn = self.sns_wrapper.create_topic(
        self.topic_name,
        self.use_fifo_topic,
        self.use_content_based_deduplication
    )

    print(f"Your new topic with the name {self.topic_name}")
    print(f" and Amazon Resource Name (ARN) {self.topic_arn}")
    print(f" has been created.")
    print()

def _setup_queues(self) -> None:
    """Set up the SQS queues and subscribe them to the topic."""
    print(f"Now you will create {self.queue_count} Amazon Simple Queue
Service (Amazon SQS) queues to subscribe to the topic.")

    for i in range(self.queue_count):
        queue_name = q.ask(f"Enter a name for SQS queue #{i+1}: ",
q.non_empty)

        if self.use_fifo_topic and i == 0:
            print("Because you have selected a FIFO topic, '.fifo' must be
appended to the queue name.")

        # Create the queue
        queue_url = self.sqs_wrapper.create_queue(queue_name,
self.use_fifo_topic)
        self.queue_urls.append(queue_url)

```

```
print(f"Your new queue with the name {queue_name}")
print(f"  and queue URL {queue_url}")
print(f"  has been created.")
print()

if i == 0:
    print("The queue URL is used to retrieve the queue ARN,")
    print("which is used to create a subscription.")
    print(self.DASHES)

# Get queue ARN
queue_arn = self.sqs_wrapper.get_queue_arn(queue_url)

if i == 0:
    print("An AWS Identity and Access Management (IAM) policy must be
attached to an SQS queue,")
    print("enabling it to receive messages from an SNS topic.")

# Set queue policy to allow SNS to send messages
self.sqs_wrapper.set_queue_policy_for_topic(queue_arn,
self.topic_arn, queue_url)

# Set up message filtering if using FIFO
subscription_arn = self._setup_subscription_with_filter(i, queue_arn,
queue_name)
self.subscription_arns.append(subscription_arn)

def _setup_subscription_with_filter(self, queue_index: int, queue_arn: str,
queue_name: str) -> str:
    """Set up subscription with optional message filtering."""
    filter_policy = None

    if self.use_fifo_topic:
        print(self.DASHES)
        if queue_index == 0:
            print("Subscriptions to a FIFO topic can have filters.")
            print("If you add a filter to this subscription, then only the
filtered messages")
            print("will be received in the queue.")
            print()
            print("For information about message filtering,")
            print("see https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html")
```

```

        print()
        print("For this example, you can filter messages by a TONE
attribute.")

        use_filter = q.ask(f"Filter messages for {queue_name}'s subscription
to the topic? (y/n): ", q.is_yesno)

        if use_filter:
            filter_policy = self._create_filter_policy()

            subscription_arn = self.sns_wrapper.subscribe_queue_to_topic(
                self.topic_arn, queue_arn, filter_policy
            )

            print(f"The queue {queue_name} has been subscribed to the topic
{self.topic_name}")
            print(f"  with the subscription ARN {subscription_arn}")

            return subscription_arn

    def _create_filter_policy(self) -> str:
        """Create a message filter policy based on user selections."""
        print(self.DASHES)
        print("You can filter messages by one or more of the following TONE
attributes.")

        filter_selections = []
        selection_number = 0

        while True:
            print("Enter a number to add a TONE filter, or enter 0 to stop adding
filters.")
            for i, tone in enumerate(self.tones, 1):
                print(f"  {i}. {tone}")

            selection = q.ask("Your choice: ", q.is_int, q.in_range(0,
len(self.tones)))

            if selection == 0:
                break
            elif selection > 0 and self.tones[selection - 1] not in
filter_selections:
                filter_selections.append(self.tones[selection - 1])
                print(f"Added '{self.tones[selection - 1]}' to filter list.")

```

```
    if filter_selections:
        filters = {"tone": filter_selections}
        return json.dumps(filters)
    return None

def _publish_messages(self) -> None:
    """Publish messages to the topic with various options."""
    print("Now we can publish messages.")

    keep_sending = True
    while keep_sending:
        print()
        message = q.ask("Enter a message to publish: ", q.non_empty)

        message_group_id = None
        deduplication_id = None
        tone_attribute = None

        if self.use_fifo_topic:
            print("Because you are using a FIFO topic, you must set a message group ID.")
            print("All messages within the same group will be received in the order they were published.")
            print()
            message_group_id = q.ask("Enter a message group ID for this message: ", q.non_empty)

            if not self.use_content_based_deduplication:
                print("Because you are not using content-based deduplication,")
                print("you must enter a deduplication ID.")
                deduplication_id = q.ask("Enter a deduplication ID for this message: ", q.non_empty)

            # Ask about tone attribute
            add_attribute = q.ask("Add an attribute to this message? (y/n): ", q.is_yn)
            if add_attribute:
                print("Enter a number for an attribute:")
                for i, tone in enumerate(self.tones, 1):
                    print(f" {i}. {tone}")
```

```

        selection = q.ask("Your choice: ", q.is_int, q.in_range(1,
len(self.tones)))
        if 1 <= selection <= len(self.tones):
            tone_attribute = self.tones[selection - 1]

    # Publish the message
    message_id = self.sns_wrapper.publish_message(
        self.topic_arn,
        message,
        tone_attribute,
        deduplication_id,
        message_group_id
    )

    print(f"Message published with ID: {message_id}")

    keep_sending = q.ask("Send another message? (y/n): ", q.is_yesno)

def _poll_queues_for_messages(self) -> None:
    """Poll all queues for messages and display results."""
    for i, queue_url in enumerate(self.queue_urls):
        print(f"Polling queue #{i+1} at {queue_url} for messages...")

        q.ask("Press Enter to continue...")

        messages = self._poll_queue_for_messages(queue_url)

        if messages:
            print(f"{len(messages)} message(s) were received by queue #{i
+1}")

            for j, message in enumerate(messages, 1):
                print(f"  Message {j}:")
                # Parse the SNS message body to get the actual message
                try:
                    sns_message = json.loads(message['Body'])
                    actual_message = sns_message.get('Message',
message['Body'])

                    print(f"    {actual_message}")
                except (json.JSONDecodeError, KeyError):
                    print(f"    {message['Body']}")

            # Delete the messages
            self.sqs_wrapper.delete_messages(queue_url, messages)
            print(f"Messages deleted from queue #{i+1}")

```

```

        else:
            print(f"No messages received by queue #{i+1}")

        print(self.DASHES)

def _poll_queue_for_messages(self, queue_url: str) -> List[Dict[str, Any]]:
    """Poll a single queue for messages."""
    all_messages = []
    max_polls = 3 # Limit polling to avoid infinite loops

    for poll_count in range(max_polls):
        messages = self.sqs_wrapper.receive_messages(queue_url, 10)

        if messages:
            all_messages.extend(messages)
            print(f" Received {len(messages)} messages in poll {poll_count +
1}")

            # Small delay between polls
            time.sleep(1)
        else:
            print(f" No messages in poll {poll_count + 1}")
            break

    return all_messages

def _cleanup_resources(self) -> None:
    """Clean up all resources created during the scenario."""
    print("Cleaning up resources...")

    # Delete queues
    for i, queue_url in enumerate(self.queue_urls):
        if queue_url:
            delete_queue = q.ask(f"Delete queue #{i+1} with URL {queue_url}?
(y/n): ", q.is_yesno)
            if delete_queue:
                try:
                    self.sqs_wrapper.delete_queue(queue_url)
                    print(f"Deleted queue #{i+1}")
                except Exception as e:
                    print(f"Error deleting queue #{i+1}: {e}")

    # Unsubscribe from topic
    for i, subscription_arn in enumerate(self.subscription_arns):
        if subscription_arn:

```

```

        try:
            self.sns_wrapper.unsubscribe(subscription_arn)
            print(f"Unsubscribed subscription #{i+1}")
        except Exception as e:
            print(f"Error unsubscribing #{i+1}: {e}")

    # Delete topic
    if self.topic_arn:
        delete_topic = q.ask(f"Delete topic {self.topic_name}? (y/n): ",
                              q.is_yesno)
        if delete_topic:
            try:
                self.sns_wrapper.delete_topic(self.topic_arn)
                print(f"Deleted topic {self.topic_name}")
            except Exception as e:
                print(f"Error deleting topic: {e}")

    print("Resource cleanup complete.")

```

シナリオで使用する Amazon SNS および Amazon SQS オペレーションをラップするクラスを作成します。

```

class SnsWrapper:
    """Wrapper class for managing Amazon SNS operations."""

    def __init__(self, sns_client: Any) -> None:
        """
        Initialize the SnsWrapper.

        :param sns_client: A Boto3 Amazon SNS client.
        """
        self.sns_client = sns_client

    @classmethod
    def from_client(cls) -> 'SnsWrapper':
        """
        Create an SnsWrapper instance using a default boto3 client.

        :return: An instance of this class.
        """
        sns_client = boto3.client('sns')

```

```

        return cls(sns_client)

    def create_topic(
        self,
        topic_name: str,
        is_fifo: bool = False,
        content_based_deduplication: bool = False
    ) -> str:
        """
        Create an SNS topic.

        :param topic_name: The name of the topic to create.
        :param is_fifo: Whether to create a FIFO topic.
        :param content_based_deduplication: Whether to use content-based
        deduplication for FIFO topics.
        :return: The ARN of the created topic.
        :raises ClientError: If the topic creation fails.
        """
        try:
            # Add .fifo suffix for FIFO topics
            if is_fifo and not topic_name.endswith('.fifo'):
                topic_name += '.fifo'

            attributes = {}
            if is_fifo:
                attributes['FifoTopic'] = 'true'
                if content_based_deduplication:
                    attributes['ContentBasedDeduplication'] = 'true'

            response = self.sns_client.create_topic(
                Name=topic_name,
                Attributes=attributes
            )

            topic_arn = response['TopicArn']
            logger.info(f"Created topic: {topic_name} with ARN: {topic_arn}")
            return topic_arn

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error creating topic {topic_name}: {error_code} -
            {e}")
            raise

```



```
def subscribe_queue_to_topic(
    self,
    topic_arn: str,
    queue_arn: str,
    filter_policy: Optional[str] = None
) -> str:
    """
    Subscribe an SQS queue to an SNS topic.

    :param topic_arn: The ARN of the SNS topic.
    :param queue_arn: The ARN of the SQS queue.
    :param filter_policy: Optional JSON filter policy for message filtering.
    :return: The ARN of the subscription.
    :raises ClientError: If the subscription fails.
    """
    try:
        attributes = {}
        if filter_policy:
            attributes['FilterPolicy'] = filter_policy

        response = self.sns_client.subscribe(
            TopicArn=topic_arn,
            Protocol='sqs',
            Endpoint=queue_arn,
            Attributes=attributes
        )

        subscription_arn = response['SubscriptionArn']
        logger.info(f"Subscribed queue {queue_arn} to topic {topic_arn}")
        return subscription_arn

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error subscribing queue to topic: {error_code} - {e}")
        raise

def publish_message(
    self,
    topic_arn: str,
    message: str,
    tone_attribute: Optional[str] = None,
```

```

        deduplication_id: Optional[str] = None,
        message_group_id: Optional[str] = None
    ) -> str:
        """
        Publish a message to an SNS topic.

        :param topic_arn: The ARN of the SNS topic.
        :param message: The message content to publish.
        :param tone_attribute: Optional tone attribute for message filtering.
        :param deduplication_id: Optional deduplication ID for FIFO topics.
        :param message_group_id: Optional message group ID for FIFO topics.
        :return: The message ID of the published message.
        :raises ClientError: If the message publication fails.
        """
        try:
            publish_args = {
                'TopicArn': topic_arn,
                'Message': message
            }

            # Add message attributes if tone is specified
            if tone_attribute:
                publish_args['MessageAttributes'] = {
                    'tone': {
                        'DataType': 'String',
                        'StringValue': tone_attribute
                    }
                }

            # Add FIFO-specific parameters
            if message_group_id:
                publish_args['MessageGroupId'] = message_group_id

            if deduplication_id:
                publish_args['MessageDeduplicationId'] = deduplication_id

            response = self.sns_client.publish(**publish_args)

            message_id = response['MessageId']
            logger.info(f"Published message to topic {topic_arn} with ID:
{message_id}")
            return message_id

        except ClientError as e:

```

```
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error publishing message to topic: {error_code} -
{e}")
        raise

def unsubscribe(self, subscription_arn: str) -> bool:
    """
    Unsubscribe from an SNS topic.

    :param subscription_arn: The ARN of the subscription to remove.
    :return: True if successful.
    :raises ClientError: If the unsubscribe operation fails.
    """
    try:
        self.sns_client.unsubscribe(SubscriptionArn=subscription_arn)

        logger.info(f"Unsubscribed: {subscription_arn}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')

        if error_code == 'NotFound':
            logger.warning(f"Subscription not found: {subscription_arn}")
            return True  # Already unsubscribed
        else:
            logger.error(f"Error unsubscribing: {error_code} - {e}")
            raise

def delete_topic(self, topic_arn: str) -> bool:
    """
    Delete an SNS topic.

    :param topic_arn: The ARN of the topic to delete.
    :return: True if successful.
    :raises ClientError: If the topic deletion fails.
    """
    try:
        self.sns_client.delete_topic(TopicArn=topic_arn)

        logger.info(f"Deleted topic: {topic_arn}")
        return True
```

```

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')

    if error_code == 'NotFound':
        logger.warning(f"Topic not found: {topic_arn}")
        return True # Already deleted
    else:
        logger.error(f"Error deleting topic: {error_code} - {e}")
        raise

def list_topics(self) -> list:
    """
    List all SNS topics in the account using pagination.

    :return: List of topic ARNs.
    :raises ClientError: If listing topics fails.
    """
    try:
        topics = []
        paginator = self.sns_client.get_paginator('list_topics')

        for page in paginator.paginate():
            topics.extend([topic['TopicArn'] for topic in page.get('Topics',
[[]])]

        logger.info(f"Found {len(topics)} topics")
        return topics

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        if error_code == 'AuthorizationError':
            logger.error("Authorization error listing topics - check IAM
permissions")
        else:
            logger.error(f"Error listing topics: {error_code} - {e}")
            raise

class SqsWrapper:
    """Wrapper class for managing Amazon SQS operations."""

    def __init__(self, sqs_client: Any) -> None:

```

```
"""
Initialize the SqsWrapper.

:param sqs_client: A Boto3 Amazon SQS client.
"""
self.sqs_client = sqs_client

@classmethod
def from_client(cls) -> 'SqsWrapper':
    """
    Create an SqsWrapper instance using a default boto3 client.

    :return: An instance of this class.
    """
    sqs_client = boto3.client('sqs')
    return cls(sqs_client)

def create_queue(self, queue_name: str, is_fifo: bool = False) -> str:
    """
    Create an SQS queue.

    :param queue_name: The name of the queue to create.
    :param is_fifo: Whether to create a FIFO queue.
    :return: The URL of the created queue.
    :raises ClientError: If the queue creation fails.
    """
    try:
        # Add .fifo suffix for FIFO queues
        if is_fifo and not queue_name.endswith('.fifo'):
            queue_name += '.fifo'

        attributes = {}
        if is_fifo:
            attributes['FifoQueue'] = 'true'

        response = self.sqs_client.create_queue(
            QueueName=queue_name,
            Attributes=attributes
        )

        queue_url = response['QueueUrl']
        logger.info(f"Created queue: {queue_name} with URL: {queue_url}")
        return queue_url
```

```
except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error creating queue {queue_name}: {error_code} - {e}")
    raise

def get_queue_arn(self, queue_url: str) -> str:
    """
    Get the ARN of an SQS queue.

    :param queue_url: The URL of the queue.
    :return: The ARN of the queue.
    :raises ClientError: If getting queue attributes fails.
    """
    try:
        response = self.sqs_client.get_queue_attributes(
            QueueUrl=queue_url,
            AttributeNames=['QueueArn']
        )

        queue_arn = response['Attributes']['QueueArn']
        logger.info(f"Queue ARN for {queue_url}: {queue_arn}")
        return queue_arn

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        logger.error(f"Error getting queue ARN: {error_code} - {e}")
        raise

def set_queue_policy_for_topic(self, queue_arn: str, topic_arn: str,
queue_url: str) -> bool:
    """
    Set the queue policy to allow SNS to send messages to the queue.

    :param queue_arn: The ARN of the SQS queue.
    :param topic_arn: The ARN of the SNS topic.
    :param queue_url: The URL of the SQS queue.
    :return: True if successful.
    :raises ClientError: If setting the queue policy fails.
    """
    try:
```

```

    # Create policy that allows SNS to send messages to the queue
    policy = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "sns.amazonaws.com"
                },
                "Action": "sqs:SendMessage",
                "Resource": queue_arn,
                "Condition": {
                    "ArnEquals": {
                        "aws:SourceArn": topic_arn
                    }
                }
            }
        ]
    }

    self.sqs_client.set_queue_attributes(
        QueueUrl=queue_url,
        Attributes={
            'Policy': json.dumps(policy)
        }
    )

    logger.info(f"Set queue policy for {queue_url} to allow messages from
{topic_arn}")
    return True

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error setting queue policy: {error_code} - {e}")
    raise

def receive_messages(self, queue_url: str, max_messages: int = 10) ->
List[Dict[str, Any]]:
    """
    Receive messages from an SQS queue.

    :param queue_url: The URL of the queue to receive messages from.
    :param max_messages: Maximum number of messages to receive (1-10).

```

```

        :return: List of received messages.
        :raises ClientError: If receiving messages fails.
        """
        try:
            # Ensure max_messages is within valid range
            max_messages = max(1, min(10, max_messages))

            response = self.sqs_client.receive_message(
                QueueUrl=queue_url,
                MaxNumberOfMessages=max_messages,
                WaitTimeSeconds=2, # Short polling
                MessageAttributeNames=['All']
            )

            messages = response.get('Messages', [])
            logger.info(f"Received {len(messages)} messages from {queue_url}")
            return messages

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error receiving messages: {error_code} - {e}")
            raise

    def delete_messages(self, queue_url: str, messages: List[Dict[str, Any]]) ->
    bool:
        """
        Delete messages from an SQS queue in batches.

        :param queue_url: The URL of the queue.
        :param messages: List of messages to delete.
        :return: True if successful.
        :raises ClientError: If deleting messages fails.
        """
        try:
            if not messages:
                return True

            # Build delete entries for batch delete
            delete_entries = []
            for i, message in enumerate(messages):
                delete_entries.append({
                    'Id': str(i),
                    'ReceiptHandle': message['ReceiptHandle']
                })

```



```

    })

    # Delete messages in batches of 10 (SQS limit)
    batch_size = 10
    for i in range(0, len(delete_entries), batch_size):
        batch = delete_entries[i:i + batch_size]

        response = self.sqs_client.delete_message_batch(
            QueueUrl=queue_url,
            Entries=batch
        )

        # Check for failures
        if 'Failed' in response and response['Failed']:
            for failed in response['Failed']:
                logger.warning(f"Failed to delete message: {failed}")

    logger.info(f"Deleted {len(messages)} messages from {queue_url}")
    return True

except ClientError as e:
    error_code = e.response.get('Error', {}).get('Code', 'Unknown')
    logger.error(f"Error deleting messages: {error_code} - {e}")
    raise

def delete_queue(self, queue_url: str) -> bool:
    """
    Delete an SQS queue.

    :param queue_url: The URL of the queue to delete.
    :return: True if successful.
    :raises ClientError: If the queue deletion fails.
    """
    try:
        self.sqs_client.delete_queue(QueueUrl=queue_url)

        logger.info(f"Deleted queue: {queue_url}")
        return True

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')

        if error_code == 'AWS.SimpleQueueService.NonExistentQueue':

```

```

        logger.warning(f"Queue not found: {queue_url}")
        return True # Already deleted
    else:
        logger.error(f"Error deleting queue: {error_code} - {e}")
        raise

def list_queues(self, queue_name_prefix: Optional[str] = None) -> List[str]:
    """
    List all SQS queues in the account using pagination.

    :param queue_name_prefix: Optional prefix to filter queue names.
    :return: List of queue URLs.
    :raises ClientError: If listing queues fails.
    """
    try:
        queue_urls = []
        paginator = self.sqs_client.get_paginator('list_queues')

        page_params = {}
        if queue_name_prefix:
            page_params['QueueNamePrefix'] = queue_name_prefix

        for page in paginator.paginate(**page_params):
            queue_urls.extend(page.get('QueueUrls', []))

        logger.info(f"Found {len(queue_urls)} queues")
        return queue_urls

    except ClientError as e:
        error_code = e.response.get('Error', {}).get('Code', 'Unknown')
        if error_code == 'AccessDenied':
            logger.error("Access denied listing queues - check IAM
permissions")
        else:
            logger.error(f"Error listing queues: {error_code} - {e}")
            raise

def send_message(self, queue_url: str, message_body: str, **kwargs) -> str:
    """
    Send a message to an SQS queue.

    :param queue_url: The URL of the queue.
    :param message_body: The message content.

```

```
        :param kwargs: Additional message parameters (DelaySeconds,
        MessageAttributes, etc.).
        :return: The message ID.
        :raises ClientError: If sending the message fails.
        """
        try:
            send_params = {
                'QueueUrl': queue_url,
                'MessageBody': message_body,
                **kwargs
            }

            response = self.sqs_client.send_message(**send_params)

            message_id = response['MessageId']
            logger.info(f"Sent message to {queue_url} with ID: {message_id}")
            return message_id

        except ClientError as e:
            error_code = e.response.get('Error', {}).get('Code', 'Unknown')
            logger.error(f"Error sending message: {error_code} - {e}")
            raise
```

- APIの詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。

- [CreateQueue](#)
- [CreateTopic](#)
- [DeleteMessageBatch](#)
- [DeleteQueue](#)
- [DeleteTopic](#)
- [GetQueueAttributes](#)
- [Publish](#)
- [ReceiveMessage](#)
- [SetQueueAttributes](#)
- [Subscribe](#)
- [Unsubscribe](#)

Swift

SDK for Swift

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 ArgumentParser
import AWSClientRuntime
import AWSSNS
import AWSSQS
import Foundation

struct ExampleCommand: ParsableCommand {
    @Option(help: "Name of the Amazon Region to use")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "queue-scenario",
        abstract: ""
        This example interactively demonstrates how to use Amazon Simple
        Notification Service (Amazon SNS) and Amazon Simple Queue Service
        (Amazon SQS) together to publish and receive messages using queues.
        "",
        discussion: ""
        Supports filtering using a "tone" attribute.
        ""
    )

    /// Prompt for an input string. Only non-empty strings are allowed.
    ///
    /// - Parameter prompt: The prompt to display.
    ///
    /// - Returns: The string input by the user.
    func stringRequest(prompt: String) -> String {
        var str: String?

        while str == nil {
            print(prompt, terminator: "")
```

```
        str = readLine()

        if str != nil && str?.count == 0 {
            str = nil
        }
    }

    return str!
}

/// Ask a yes/no question.
///
/// - Parameter prompt: A prompt string to print.
///
/// - Returns: `true` if the user answered "Y", otherwise `false`.
func yesNoRequest(prompt: String) -> Bool {
    while true {
        let answer = stringRequest(prompt: prompt).lowercased()
        if answer == "y" || answer == "n" {
            return answer == "y"
        }
    }
}

/// Display a menu of options then request a selection.
///
/// - Parameters:
///   - prompt: A prompt string to display before the menu.
///   - options: An array of strings giving the menu options.
///
/// - Returns: The index number of the selected option or 0 if no item was
///   selected.
func menuRequest(prompt: String, options: [String]) -> Int {
    let numOptions = options.count

    if numOptions == 0 {
        return 0
    }

    print(prompt)

    for (index, value) in options.enumerated() {
        print("\(index) \(value)")
    }
}
```

```
repeat {
    print("Enter your selection (0 - \$(numOptions-1)): ", terminator: "")
    if let answer = readLine() {
        guard let answer = Int(answer) else {
            print("Please enter the number matching your selection.")
            continue
        }

        if answer >= 0 && answer < numOptions {
            return answer
        } else {
            print("Please enter the number matching your selection.")
        }
    }
} while true
}

/// Ask the user too press RETURN. Accepts any input but ignores it.
///
/// - Parameter prompt: The text prompt to display.
func returnRequest(prompt: String) {
    print(prompt, terminator: "")
    _ = readLine()
}

var attrValues = [
    "<none>",
    "cheerful",
    "funny",
    "serious",
    "sincere"
]

/// Ask the user to choose one of the attribute values to use as a filter.
///
/// - Parameters:
///   - message: A message to display before the menu of values.
///   - attrValues: An array of strings giving the values to choose from.
///
/// - Returns: The string corresponding to the selected option.
func askForFilter(message: String, attrValues: [String]) -> String? {
    print(message)
    for (index, value) in attrValues.enumerated() {
```

```

        print("  [\\(index)] \\(value)")
    }

    var answer: Int?
    repeat {
        answer = Int(stringRequest(prompt: "Select an value for the 'tone'
attribute or 0 to end: "))
    } while answer == nil || answer! < 0 || answer! > attrValues.count + 1

    if answer == 0 {
        return nil
    }
    return attrValues[answer!]
}

/// Prompts the user for filter terms and constructs the attribute
/// record that specifies them.
///
/// - Returns: A mapping of "FilterPolicy" to a JSON string representing
///   the user-defined filter.
func buildFilterAttributes() -> [String:String] {
    var attr: [String:String] = [:]
    var filterString = ""

    var first = true

    while let ans = askForFilter(message: "Choose a value to apply to the
'tone' attribute.",
                                attrValues: attrValues) {
        if !first {
            filterString += ","
        }
        first = false

        filterString += "\\\"\\(ans)\\\""
    }

    let filterJSON = "{ \"tone\": [\\(filterString)]}"
    attr["FilterPolicy"] = filterJSON

    return attr
}
/// Create a queue, returning its URL string.
///

```

```

    /// - Parameters:
    ///   - prompt: A prompt to ask for the queue name.
    ///   - isFIFO: Whether or not to create a FIFO queue.
    ///
    /// - Returns: The URL of the queue.
    func createQueue(prompt: String, sqsClient: SQSClient, isFIFO: Bool) async
    throws -> String? {
        repeat {
            var queueName = stringRequest(prompt: prompt)
            var attributes: [String: String] = [:]

            if isFIFO {
                queueName += ".fifo"
                attributes["FifoQueue"] = "true"
            }

            do {
                let output = try await sqsClient.createQueue(
                    input: CreateQueueInput(
                        attributes: attributes,
                        queueName: queueName
                    )
                )
                guard let url = output.queueUrl else {
                    return nil
                }

                return url
            } catch _ as QueueDeletedRecently {
                print("You need to use a different queue name. A queue by that
name was recently deleted.")
                continue
            }
        } while true
    }

    /// Return the ARN of a queue given its URL.
    ///
    /// - Parameter queueUrl: The URL of the queue for which to return the
    ///   ARN.
    ///
    /// - Returns: The ARN of the specified queue.
    func getQueueARN(sqsClient: SQSClient, queueUrl: String) async throws ->
    String? {

```



```

    let output = try await sqsClient.getQueueAttributes(
        input: GetQueueAttributesInput(
            attributeNames: [.queueArn],
            queueUrl: queueUrl
        )
    )

    guard let attributes = output.attributes else {
        return nil
    }

    return attributes["QueueArn"]
}

/// Applies the needed policy to the specified queue.
///
/// - Parameters:
///   - sqsClient: The Amazon SQS client to use.
///   - queueUrl: The queue to apply the policy to.
///   - queueArn: The ARN of the queue to apply the policy to.
///   - topicArn: The topic that should have access via the policy.
///
/// - Throws: Errors from the SQS `SetQueueAttributes` action.
func setQueuePolicy(sqsClient: SQSClient, queueUrl: String,
                    queueArn: String, topicArn: String) async throws {
    _ = try await sqsClient.setQueueAttributes(
        input: SetQueueAttributesInput(
            attributes: [
                "Policy":
                """
                {
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "sns.amazonaws.com"
                            },
                            "Action": "sqs:SendMessage",
                            "Resource": "\(queueArn)",
                            "Condition": {
                                "ArnEquals": {
                                    "aws:SourceArn": "\(topicArn)"
                                }
                            }
                        }
                    ]
                }
                """
            ]
        )
    )
}

```

```

        }
    ]
}
"""

    ],
    queueUrl: queueUrl
)
)
}

/// Receive the available messages on a queue, outputting them to the
/// screen. Returns a dictionary you pass to DeleteMessageBatch to delete
/// all the received messages.
///
/// - Parameters:
///   - sqsClient: The Amazon SQS client to use.
///   - queueUrl: The SQS queue on which to receive messages.
///
/// - Throws: Errors from `SQSClient.receiveMessage()`
///
/// - Returns: An array of SQSClientTypes.DeleteMessageBatchRequestEntry
///   items, each describing one received message in the format needed to
///   delete it.
func receiveAndListMessages(sqsClient: SQSClient, queueUrl: String) async
throws
    ->
[SQSClientTypes.DeleteMessageBatchRequestEntry] {
    let output = try await sqsClient.receiveMessage(
        input: ReceiveMessageInput(
            maxNumberOfMessages: 10,
            queueUrl: queueUrl
        )
    )

    guard let messages = output.messages else {
        print("No messages received.")
        return []
    }

    var deleteList: [SQSClientTypes.DeleteMessageBatchRequestEntry] = []

    // Print out all the messages that were received, including their
    // attributes, if any.

```

```

    for message in messages {
        print("Message ID:      \$(message.messageId ?? "<unknown>")")
        print("Receipt handle: \$(message.receiptHandle ?? "<unknown>")")
        print("Message JSON:    \$(message.body ?? "<body missing>")")

        if message.receiptHandle != nil {
            deleteList.append(
                SQSClientTypes.DeleteMessageBatchRequestEntry(
                    id: message.messageId,
                    receiptHandle: message.receiptHandle
                )
            )
        }
    }

    return deleteList
}

/// Delete all the messages in the specified list.
///
/// - Parameters:
///   - sqsClient: The Amazon SQS client to use.
///   - queueUrl: The SQS queue to delete messages from.
///   - deleteList: A list of `DeleteMessageBatchRequestEntry` objects
///     describing the messages to delete.
///
/// - Throws: Errors from `SQSClient.deleteMessageBatch()`.
func deleteMessageList(sqsClient: SQSClient, queueUrl: String,
                      deleteList:
[SQSClientTypes.DeleteMessageBatchRequestEntry]) async throws {
    let output = try await sqsClient.deleteMessageBatch(
        input: DeleteMessageBatchInput(entries: deleteList, queueUrl:
queueUrl)
    )

    if let failed = output.failed {
        print("\$(failed.count) errors occurred deleting messages from the
queue.")
        for message in failed {
            print("---> Failed to delete message \$(message.id ?? "<unknown
ID>") with error: \$(message.code ?? "<unknown>") (\$(message.message ?? "..."))")
        }
    }
}

```

```

    }

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let rowOfStars = String(repeating: "*", count: 75)

        print("""
            \(\rowOfStars)
            Welcome to the cross-service messaging with topics and queues
example.

            In this workflow, you'll create an SNS topic, then create two SQS
            queues which will be subscribed to that topic.

            You can specify several options for configuring the topic, as well
as
            the queue subscriptions. You can then post messages to the topic
and
            receive the results on the queues.
            \(\rowOfStars)\n
            """)
        )

        // 0. Create SNS and SQS clients.

        let snsConfig = try await SNSClient.SNSClientConfiguration(region:
region)
        let snsClient = SNSClient(config: snsConfig)

        let sqsConfig = try await SQSClient.SQSClientConfiguration(region:
region)
        let sqsClient = SQSClient(config: sqsConfig)

        // 1. Ask the user whether to create a FIFO topic. If so, ask whether
        //     to use content-based deduplication instead of requiring a
        //     deduplication ID.

        let isFIFO = yesNoRequest(prompt: "Do you want to create a FIFO topic (Y/
N)? ")
        var isContentBasedDeduplication = false

        if isFIFO {
            print("""
                \(\rowOfStars)
                Because you've chosen to create a FIFO topic, deduplication is

```

```

        supported.

        Deduplication IDs are either set in the message or are
automatically
        generated from the content using a hash function.

        If a message is successfully published to an SNS FIFO topic,
any
        message published and found to have the same deduplication ID
        (within a five-minute deduplication interval), is accepted but
        not delivered.

        For more information about deduplication, see:
        https://docs.aws.amazon.com/sns/latest/dg/fifo-message-
dedup.html.
        """
    )

    isContentBasedDeduplication = yesNoRequest(
        prompt: "Use content-based deduplication instead of entering a
deduplication ID (Y/N)? ")
    print(rowOfStars)
}

var topicName = stringRequest(prompt: "Enter the name of the topic to
create: ")

// 2. Create the topic. Append ".fifo" to the name if FIFO was
// requested, and set the "FifoTopic" attribute to "true" if so as
// well. Set the "ContentBasedDeduplication" attribute to "true" if
// content-based deduplication was requested.

if isFIFO {
    topicName += ".fifo"
}

print("Topic name: \(topicName)")

var attributes = [
    "FifoTopic": (isFIFO ? "true" : "false")
]

// If it's a FIFO topic with content-based deduplication, set the
// "ContentBasedDeduplication" attribute.

```

```
    if isContentBasedDeduplication {
        attributes["ContentBasedDeduplication"] = "true"
    }

    // Create the topic and retrieve the ARN.

    let output = try await snsClient.createTopic(
        input: CreateTopicInput(
            attributes: attributes,
            name: topicName
        )
    )

    guard let topicArn = output.topicArn else {
        print("No topic ARN returned!")
        return
    }

    print("""
        Topic '\(topicName)' has been created with the
        topic ARN '\(topicArn)'.
    """)

    print(rowOfStars)

    // 3. Create an SQS queue. Append ".fifo" to the name if one of the
    //     FIFO topic configurations was chosen, and set "FifoQueue" to
    //     "true" if the topic is FIFO.

    print("""
        Next, you will create two SQS queues that will be subscribed
        to the topic you just created.
    """)

    let q1Url = try await createQueue(prompt: "Enter the name of the first
queue: ",
                                     sqsClient: sqsClient, isFIFO: isFIFO)

    guard let q1Url else {
        print("Unable to create queue 1!")
        return
    }
}
```

```
// 4. Get the SQS queue's ARN attribute using `GetQueueAttributes`.

let q1Arn = try await getQueueARN(sqsClient: sqsClient, queueUrl: q1Url)

guard let q1Arn else {
    print("Unable to get ARN of queue 1!")
    return
}
print("Got queue 1 ARN: \(q1Arn)")

// 5. Attach an AWS IAM policy to the queue using
//     `SetQueueAttributes`.

try await setQueuePolicy(sqsClient: sqsClient, queueUrl: q1Url,
                        queueArn: q1Arn, topicArn: topicArn)

// 6. Subscribe the SQS queue to the SNS topic. Set the topic ARN in
//     the request. Set the protocol to "sqs". Set the queue ARN to the
//     ARN just received in step 5. For FIFO topics, give the option to
//     apply a filter. A filter allows only matching messages to enter
//     the queue.

var q1Attributes: [String:String]? = nil

if isFIFO {
    print(
        """
        If you add a filter to this subscription, then only the filtered
messages will
        be received in the queue. For information about message
filtering, see
https://docs.aws.amazon.com/sns/latest/dg/sns-message-
filtering.html
        For this example, you can filter messages by a 'tone' attribute.
        """
    )

    let subPrompt = """
        Would you like to filter messages for the first queue's
subscription to the
        topic \(topicName) (Y/N)?
    """
}
```

```
        ""
        if (yesNoRequest(prompt: subPrompt)) {
            q1Attributes = buildFilterAttributes()
        }
    }

    let sub1Output = try await snsClient.subscribe(
        input: SubscribeInput(
            attributes: q1Attributes,
            endpoint: q1Arn,
            protocol: "sqs",
            topicArn: topicArn
        )
    )

    guard let q1SubscriptionArn = sub1Output.subscriptionArn else {
        print("Invalid subscription ARN returned for queue 1!")
        return
    }

    // 7. Repeat steps 3-6 for the second queue.

    let q2Url = try await createQueue(prompt: "Enter the name of the second
queue: ",
                                     sqsClient: sqsClient, isFIFO: isFIFO)

    guard let q2Url else {
        print("Unable to create queue 2!")
        return
    }

    let q2Arn = try await getQueueARN(sqsClient: sqsClient, queueUrl: q2Url)

    guard let q2Arn else {
        print("Unable to get ARN of queue 2!")
        return
    }
    print("Got queue 2 ARN: \(q2Arn)")

    try await setQueuePolicy(sqsClient: sqsClient, queueUrl: q2Url,
                             queueArn: q2Arn, topicArn: topicArn)

    var q2Attributes: [String:String]? = nil
```



```

        if isFIFO {
            let subPrompt = ""
            Would you like to filter messages for the second queue's
subscription to the
            topic \((topicName) (Y/N)?
            ""
            if (yesNoRequest(prompt: subPrompt)) {
                q2Attributes = buildFilterAttributes()
            }
        }

        let sub2Output = try await snsClient.subscribe(
            input: SubscribeInput(
                attributes: q2Attributes,
                endpoint: q2Arn,
                protocol: "sqs",
                topicArn: topicArn
            )
        )

        guard let q2SubscriptionArn = sub2Output.subscriptionArn else {
            print("Invalid subscription ARN returned for queue 1!")
            return
        }

        // 8. Let the user publish messages to the topic, asking for a message
        //     body for each message. Handle the types of topic correctly (SEE
        //     MVP INFORMATION AND FIX THESE COMMENTS!!!

        print("\n\((rowOfStars)\n")

        var first = true

        repeat {
            var publishInput = PublishInput(
                topicArn: topicArn
            )

            publishInput.message = stringRequest(prompt: "Enter message text to
publish: ")

            // If using a FIFO topic, a message group ID must be set on the
            // message.

```

```

        if isFIFO {
            if first {
                print("""
                    Because you're using a FIFO topic, you must set a message
                    group ID. All messages within the same group will be
                    received in the same order in which they were published.

\n
                    """)
            }
            publishInput.messageGroupId = stringRequest(prompt: "Enter a
message group ID for this message: ")

            if !isContentBasedDeduplication {
                if first {
                    print("""
                        Because you're not using content-based
deduplication, you
                        must enter a deduplication ID. If other messages
with the
                        same deduplication ID are published within the same
deduplication interval, they will not be delivered.
                    """)
                }
            }
            publishInput.messageDeduplicationId = stringRequest(prompt:
"Enter a deduplication ID for this message: ")
        }
    }

    // Allow the user to add a value for the "tone" attribute if they
    // wish to do so.

    var messageAttributes: [String:SNSClientTypes.MessageAttributeValue]
= [:]

    let attrValSelection = menuRequest(prompt: "Choose a tone to apply to
this message.", options: attrValues)

    if attrValSelection != 0 {
        let val = SNSClientTypes.MessageAttributeValue(dataType:
"String", stringValue: attrValues[attrValSelection])
        messageAttributes["tone"] = val
    }

```

```
        publishInput.messageAttributes = messageAttributes

        // Publish the message and display its ID.

        let publishOutput = try await snsClient.publish(input: publishInput)

        guard let messageId = publishOutput.messageId else {
            print("Unable to get the published message's ID!")
            return
        }

        print("Message published with ID \(messageId).")
        first = false

        // 9. Repeat step 8 until the user says they don't want to post
        //     another.

    } while (yesNoRequest(prompt: "Post another message (Y/N)? "))

    // 10. Display a list of the messages in each queue by using
    //     `ReceiveMessage`. Show at least the body and the attributes.

    print(rowOfStars)
    print("Contents of queue 1:")
    let q1DeleteList = try await receiveAndListMessages(sqsClient: sqsClient,
queueUrl: q1Url)
    print("\n\nContents of queue 2:")
    let q2DeleteList = try await receiveAndListMessages(sqsClient: sqsClient,
queueUrl: q2Url)
    print(rowOfStars)

    returnRequest(prompt: "\nPress return to clean up: ")

    // 11. Delete the received messages using `DeleteMessageBatch`.

    print("Deleting the messages from queue 1...")
    try await deleteMessageList(sqsClient: sqsClient, queueUrl: q1Url,
deleteList: q1DeleteList)
    print("\nDeleting the messages from queue 2...")
    try await deleteMessageList(sqsClient: sqsClient, queueUrl: q2Url,
deleteList: q2DeleteList)

    // 12. Unsubscribe and delete both queues.
```

```
    print("\nUnsubscribing from queue 1...")
    _ = try await snsClient.unsubscribe(
        input: UnsubscribeInput(subscriptionArn: q1SubscriptionArn)
    )

    print("Unsubscribing from queue 2...")
    _ = try await snsClient.unsubscribe(
        input: UnsubscribeInput(subscriptionArn: q2SubscriptionArn)
    )

    print("Deleting queue 1...")
    _ = try await sqsClient.deleteQueue(
        input: DeleteQueueInput(queueUrl: q1Url)
    )

    print("Deleting queue 2...")
    _ = try await sqsClient.deleteQueue(
        input: DeleteQueueInput(queueUrl: q2Url)
    )

    // 13. Delete the topic.

    print("Deleting the SNS topic...")
    _ = try await snsClient.deleteTopic(
        input: DeleteTopicInput(topicArn: topicArn)
    )
}

}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- API の詳細については、AWS SDK for Swift API リファレンスの以下のトピックを参照してください。
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Publish](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Subscribe](#)
 - [Unsubscribe](#)

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 での Amazon SNS 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。

API Gateway を使用して Lambda 関数を呼び出す

次のコード例は、Amazon API Gateway によって呼び出される AWS Lambda 関数を作成する方法を示しています。

Java

SDK for Java 2.x

Lambda Java ランタイム API を使用して AWS Lambda 関数を作成する方法を示します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、Amazon API Gateway によって呼び出される Lambda 関数を作成する方法を示します。この関数は、Amazon DynamoDB テーブルをスキャンして、Amazon Simple Notification Service (Amazon SNS) を使用して、従業員に年間の記念日を祝福するテキストメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

Lambda JavaScript ランタイム API を使用して AWS Lambda 関数を作成する方法を示します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、Amazon API Gateway によって呼び出される Lambda 関数を作成する方法を示します。この関数は、Amazon DynamoDB テーブルをスキャンして、Amazon Simple Notification Service (Amazon SNS) を使用して、従業員に年間の記念日を祝福するテキストメッセージを送信します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例は、[AWS SDK for JavaScript v3 デベロッパーガイド](#)でも使用できます。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon SNS

Python

SDK for Python (Boto3)

この例は、AWS Lambda 関数を対象とする Amazon API Gateway REST API を作成して使用する方法を示しています。Lambda ハンドラーは、HTTP メソッドに基づいてルーティングす

る方法を示します。クエリ文字列、ヘッダー、および本文からデータを取得する方法。そして、JSON 応答を返す方法。

- Lambda 関数をデプロイします。
- API ゲートウェイ REST API を作成します。
- Lambda 関数をターゲットとする REST リソースを作成します。
- API Gateway に Lambda 関数を呼び出す権限を付与します。
- リクエストパッケージを使用して、REST API にリクエストを送信します。
- デモ中に作成されたすべてのリソースをクリーンアップします。

この例は GitHub で最もよく確認できます。完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- API ゲートウェイ
- DynamoDB
- Lambda
- Amazon SNS

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 での Amazon SNS 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。

スケジュールされたイベントを使用した Lambda 関数の呼び出し

次のコード例は、Amazon EventBridge スケジュールされたイベントによって呼び出される AWS Lambda 関数を作成する方法を示しています。

Java

SDK for Java 2.x

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda Java ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定の

ユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK for JavaScript (v3)

AWS Lambda 関数を呼び出す Amazon EventBridge スケジュールされたイベントを作成する方法を示します。cron 式を使用して Lambda 関数が呼び出されるタイミングをスケジュールするように EventBridge を設定します。この例では、Lambda JavaScript ランタイム API を使用して Lambda 関数を作成します。この例では、さまざまな AWS サービスを呼び出して、特定のユースケースを実行します。この例では、年間の記念日に従業員を祝福するモバイルテキストメッセージを従業員に送信するアプリを作成する方法を示します。

完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例は、[AWS SDK for JavaScript v3 デベロッパーガイド](#)でも使用できます。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK for Python (Boto3)

この例では、スケジュールされた Amazon EventBridge イベントのターゲットとして AWS Lambda 関数を登録する方法を示します。Lambda ハンドラーは、後で取得するために Amazon CloudWatch Logs にわかりやすいメッセージと完全なイベントデータを書き込みます。

- Lambda 関数をデプロイします。
- EventBridge スケジュールイベントを作成し、Lambda 関数をターゲットにします。
- EventBridge に Lambda 関数を呼び出す許可を付与します
- CloudWatch Logs から最新のデータを出力して、スケジュールされた呼び出しの結果を表示しています。
- デモ中に作成されたすべてのリソースをクリーンアップします。

この例は GitHub で最もよく確認できます。完全なソースコードとセットアップおよび実行の手順については、[GitHub](#) で完全な例を参照してください。

この例で使用されているサービス

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

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 での Amazon SNS 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。

Amazon SNS のサーバーレス例

次のコード例は、AWS SDKs で Amazon SNS を使用する方法を示しています。

例

- [Amazon SNS トリガーから Lambda 関数を呼び出す](#)

Amazon SNS トリガーから Lambda 関数を呼び出す

次のコード例は、SNS トピックからメッセージを受信することによってトリガーされるイベントを受け取る Lambda 関数を実装する方法を示しています。この関数はイベントパラメータからメッセージを取得し、各メッセージの内容を記録します。

.NET

SDK for .NET

Note

GitHub には、その他のリソースもあります。[サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行する方法を確認してください。

.NET を使用した Lambda での SNS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SNSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
// into a .NET class.
[assembly:
    LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer))]

namespace SnsIntegration;

public class Function
{
    public async Task FunctionHandler(SNSEvent evnt, ILambdaContext context)
    {
        foreach (var record in evnt.Records)
        {
            await ProcessRecordAsync(record, context);
        }
        context.Logger.LogInformation("done");
    }
}
```

```
private async Task ProcessRecordAsync(SNSEvent.SNSRecord record,
ILambdaContext context)
{
    try
    {
        context.Logger.LogInformation($"Processed record
{record.Sns.Message}");

        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
    catch (Exception e)
    {
        //You can use Dead Letter Queue to handle failures. By configuring a
        Lambda DLQ.
        context.Logger.LogError($"An error occurred");
        throw;
    }
}
}
```

Go

SDK for Go V2

Note

GitHub には、その他のリソースもあります。[サーバーレスサンプル](#)リポジトリで完全な例を見つけて、設定と実行の方法を確認してください。

Go を使用して Lambda で SNS イベントを消費します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-lambda-go/events"
```

```
"github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
    for _, record := range snsEvent.Records {
        processMessage(record)
    }
    fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
    message := record.SNS.Message
    fmt.Printf("Processed message: %s\n", message)
    // TODO: Process your record here
}

func main() {
    lambda.Start(handler)
}
```

Java

SDK for Java 2.x

Note

GitHub には、その他のリソースもあります。[サーバーレスサンプル](#)リポジトリで完全な例を見つけて、設定と実行の方法を確認してください。

Java を使用した Lambda での SNS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;
import com.amazonaws.services.lambda.runtime.events.SNSEvent.SNSRecord;
```

```
import java.util.Iterator;
import java.util.List;

public class SNSEventHandler implements RequestHandler<SNSEvent, Boolean> {
    LambdaLogger logger;

    @Override
    public Boolean handleRequest(SNSEvent event, Context context) {
        logger = context.getLogger();
        List<SNSRecord> records = event.getRecords();
        if (!records.isEmpty()) {
            Iterator<SNSRecord> recordsIter = records.iterator();
            while (recordsIter.hasNext()) {
                processRecord(recordsIter.next());
            }
        }
        return Boolean.TRUE;
    }

    public void processRecord(SNSRecord record) {
        try {
            String message = record.getSNS().getMessage();
            logger.log("message: " + message);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

JavaScript

SDK for JavaScript (v3)

Note

GitHub には、その他のリソースもあります。[サーバーレスサンプル](#)リポジトリで完全な例を検索し、設定および実行する方法を確認してください。

JavaScript を使用した Lambda での SNS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

TypeScript を使用した Lambda での SNS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
```

```
) : Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

PHP

SDK for PHP

Note

GitHub には、その他のリソースもあります。[サーバーレスサンプル](#)リポジトリで完全な例を見つけて、設定と実行の方法を確認してください。

PHP を使用して Lambda で SNS イベントを消費します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

/*
Since native PHP support for AWS Lambda is not available, we are utilizing Bref's
PHP functions runtime for AWS Lambda.
For more information on Bref's PHP runtime for Lambda, refer to: https://bref.sh/docs/runtimes/function

Another approach would be to create a custom runtime.
```

```
A practical example can be found here: https://aws.amazon.com/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/  
*/  
  
// Additional composer packages may be required when using Bref or any other PHP  
// functions runtime.  
// require __DIR__ . '/vendor/autoload.php';  
  
use Bref\Context\Context;  
use Bref\Event\Sns\SnsEvent;  
use Bref\Event\Sns\SnsHandler;  
  
class Handler extends SnsHandler  
{  
    public function handleSns(SnsEvent $event, Context $context): void  
    {  
        foreach ($event->getRecords() as $record) {  
            $message = $record->getMessage();  
  
            // TODO: Implement your custom processing logic here  
            // Any exception thrown will be logged and the invocation will be  
            marked as failed  
  
            echo "Processed Message: $message" . PHP_EOL;  
        }  
    }  
}  
  
return new Handler();
```

Python

SDK for Python (Boto3)

Note

GitHub には、その他のリソースもあります。[サーバーレスサンプル](#)リポジトリで完全な例を見つけて、設定と実行の方法を確認してください。

Python を使用して Lambda で SNS イベントを消費します。


```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event, context):
    for record in event['Records']:
        process_message(record)
    print("done")

def process_message(record):
    try:
        message = record['Sns']['Message']
        print(f"Processed message {message}")
        # TODO; Process your record here

    except Exception as e:
        print("An error occurred")
        raise e
```

Ruby

SDK for Ruby

Note

GitHub には、その他のリソースもあります。[サーバーレスサンプル](#)リポジトリで完全な例を見つけて、設定と実行の方法を確認してください。

Ruby を使用した Lambda での SNS イベントの消費。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def lambda_handler(event:, context:)
    event['Records'].map { |record| process_message(record) }
end

def process_message(record)
    message = record['Sns']['Message']
    puts("Processing message: #{message}")
    rescue StandardError => e
        puts("Error processing message: #{e}")
```

```
raise
end
```

Rust

SDK for Rust

Note

GitHub には、その他のリソースもあります。[サーバーレスサンプル](#)リポジトリで完全な例を見つけて、設定と実行の方法を確認してください。

Rust を使用して Lambda で SNS イベントを消費します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::sns::SnsEvent;
use aws_lambda_events::sns::SnsRecord;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};
use tracing::info;

// Built with the following dependencies:
// aws_lambda_events = { version = "0.10.0", default-features = false, features
//   = ["sns"] }
// lambda_runtime = "0.8.1"
// tokio = { version = "1", features = ["macros"] }
// tracing = { version = "0.1", features = ["log"] }
// tracing-subscriber = { version = "0.3", default-features = false, features =
//   ["fmt"] }

async fn function_handler(event: LambdaEvent<SnsEvent>) -> Result<(), Error> {
    for event in event.payload.records {
        process_record(&event)?;
    }

    Ok(())
}

fn process_record(record: &SnsRecord) -> Result<(), Error> {
    info!("Processing SNS Message: {}", record.sns.message);
}
```

```
    // Implement your record handling code here.

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        .with_target(false)
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

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 での Amazon SNS 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。

Amazon SNS セキュリティ

責任 AWS [共有モデル](#)、でのデータ保護に適用されます。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。このコンテンツには、使用する AWS サービスのセキュリティ設定および管理タスクが含まれます。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、(AWS Identity and Access Management IAM) を使用して AWS アカウント 認証情報を保護し、個々のユーザーアカウント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アクセス許可のみを各ユーザーに付与できます。また、次の方法でデータを保護することをお勧めします。

- 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 以降が推奨されます。
- 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。
- AWS 暗号化ソリューションと、サービス内のすべての AWS デフォルトのセキュリティコントロールを使用します。
- Amazon Macie などのアドバンスドマネージドセキュリティサービスを使用します。これは、Amazon S3 に保存されている個人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 140-2 検証済みの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 \(FIPS\) 140-2](#)」を参照してください。
- メッセージデータ保護
 - メッセージデータ保護は Amazon SNS の新しい主要機能です
 - MDP を使用してメッセージをスキャンして機密情報または重要情報がないか調べる
 - トピック内を流れるすべてのコンテンツにメッセージ監査を提供する
 - トピックに公開されたメッセージおよびトピックによって配信されるメッセージへのコンテンツアクセス制御を提供する

⚠ Important

顧客の E メールアドレスなどの機密情報やセンシティブ情報は、タグや 名前 フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これは、コンソール、API、AWS CLI または AWS SDK を使用して Amazon SNS または他の Amazon Web Services を使用する場合も同様です。SDKs タグまたは名前に使用する自由記入欄に入力したデータは、課金や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。

Amazon SNS データ暗号化

データ保護には、転送時 (Amazon SNS との間でデータを送受信するとき) のデータを保護するものと、保管時 (Amazon SNS データセンター内のディスクに格納されているとき) のデータを保護するものがあります。Secure Sockets Layer (SSL) またはクライアント側の暗号化を使用して、転送時のデータを保護できます。デフォルトでは、Amazon SNS は、ディスク暗号化を使用してメッセージとファイルを保存します。保管中のデータを保護するには、Amazon SNS にメッセージの暗号化をリクエストしてから、メッセージをデータセンターの暗号化されたファイルシステムに保存します。Amazon SNS では、SSE を使用してデータ暗号化を最適化することをお勧めします。

サーバー側の暗号化を使用した Amazon SNS データの保護

サーバー側の暗号化 (SSE) では、AWS Key Management Service () で管理されるキーを使用して Amazon SNS トピック内のメッセージの内容を保護することで、暗号化されたトピックに機密データを保存できます AWS KMS。

Amazon SNS が受信したメッセージはすぐに、SSE によって暗号化されます。メッセージは暗号化された形式で保存され、送信された場合のみ解読されます。

- AWS マネジメントコンソール または ([CreateTopic](#) および [SetTopicAttributes](#) API アクションを使用して KmsMasterKeyId 属性を設定 AWS SDK for Java) を使用して SSE を管理する方法については、「」を参照してください [サーバー側の暗号化を使用した Amazon SNS トピック暗号化のセットアップ](#)。
- ([AWS::SNS::Topic](#) リソースを使用して KmsMasterKeyId プロパティを設定 CloudFormation して) を使用して暗号化されたトピックを作成する方法については、AWS CloudFormation ユーザーガイドを参照してください。

⚠ Important

SSE が有効なトピックへのリクエストでは必ず、HTTPS と [署名バージョン 4](#) を使用する必要があります。

暗号化されたトピックとその他のサービスとの互換性については、サービスのドキュメントを参照してください。

Amazon SNS は、対称暗号化 KMS キーのみをサポートします。他のタイプの KMS キーを使用してサービスリソースを暗号化することはできません。KMS キーが対称暗号化キーかどうかを判別するには、「[非対称 KMS キーを識別する](#)」を参照してください。

AWS KMS は、安全で可用性の高いハードウェアとソフトウェアを組み合わせ、クラウド向けにスケールされたキー管理システムを提供します。で Amazon SNS を使用すると AWS KMS、メッセージデータを暗号化する [データキー](#) も暗号化され、保護するデータとともに保存されます。

AWS KMS を使用する利点は次のとおりです。

- お客様自身で [AWS KMS key](#) を作成および管理できます。
- Amazon SNS には、アカウントとリージョンごとに一意の AWS マネージド KMS キーを使用することもできます。
- AWS KMS セキュリティ標準は、暗号化関連のコンプライアンス要件を満たすのに役立ちます。

詳細については、「AWS Key Management Service デベロッパーガイド」の「[What is AWS Key Management Service?](#)」を参照してください。

暗号化スコープ

SSE では、Amazon SNS トピック内のメッセージの本文が暗号化されます。

SSE では、以下は暗号化されません。

- トピックのメタデータ (トピック名と属性)
- メッセージのメタデータ (件名、メッセージ ID、タイムスタンプ、属性)
- データ保護ポリシー
- トピックごとのメトリクス

Note

- メッセージが暗号化されるのは、トピックの暗号化が有効になった後に送信される場合のみです。Amazon SNS は、バックログされたメッセージを暗号化しません。
- トピックの暗号化が無効になっても、暗号化されたメッセージは暗号化された状態で維持されます。

重要な用語

以下の重要なキーは、SSEの機能を理解するうえで役立ちます。詳細については、「[Amazon Simple Notification Service API リファレンス](#)」を参照してください。

データキー

データ暗号化キー (DEK) は、Amazon SNS メッセージの内容を暗号化します。

詳細については、『AWS Key Management Service デベロッパーガイド』の「[データキー](#)」と、『AWS Encryption SDK デベロッパーガイド』の「[エンベロープ暗号化](#)」を参照してください。

AWS KMS key ID

AWS KMSアカウントまたは別のアカウントの またはカスタムのエイリアス、エイリアス ARN、キー ID AWS KMS key、またはキー ARN。Amazon SNS AWS KMS で AWS 管理される のエイリアスは常に ですがalias/aws/sns、カスタムのエイリアスは など AWS KMS で alias/*MyAlias*。これらの AWS KMS キーを使用して、Amazon SNS トピック内のメッセージを保護することができます。

Note

以下に留意してください。

- AWS マネジメントコンソール を使用してトピックの Amazon SNS の AWS マネージド KMS を初めて指定すると、 は Amazon SNS の AWS マネージド KMS AWS KMS を作成します。
- または、SSE が有効になっているトピックで Publishアクションを初めて使用すると、 は Amazon SNS の AWS マネージド KMS AWS KMS を作成します。

AWS KMS コンソールの AWS KMS keys セクション AWS KMS または [CreateKey](#) AWS KMS アクションを使用して、キーの作成、AWS KMS キーの使用方法を制御するポリシーの定義、AWS KMS 使用状況の監査を行うことができます。詳細については、「[AWS KMS keys](#)」および「AWS Key Management Service デベロッパーガイド」の「[キーの作成](#)」を参照してください。AWS KMS 識別子のその他の例については、AWS Key Management Service API リファレンスの [KeyId](#) を参照してください。AWS KMS 識別子の検索の詳細については、「AWS Key Management Service デベロッパーガイド」の「[キー ID と ARN の検索](#)」を参照してください。

 Important

の使用には追加料金がかかります AWS KMS。詳細については、「[AWS KMS コストの見積もり](#)」と「[AWS Key Management Service 料金表](#)」を参照してください。

Amazon SNS 暗号化キーとコストの管理

次のセクションでは、AWS Key Management Service (AWS KMS) で管理されるキーの操作方法について説明します。

 Note

Amazon SNS は、対称暗号化 KMS キーのみをサポートします。他のタイプの KMS キーを使用してサービスリソースを暗号化することはできません。KMS キーが対称暗号化キーかどうかを判別するには、「[非対称 KMS キーを識別する](#)」を参照してください。

AWS KMS コストの見積もり

コストを予測し、AWS 請求をよりよく理解するために、Amazon SNS が 使用する頻度を知りたい場合があります AWS KMS key。

 Note

コストは下の計算式でかなり正確に予測できますが、Amazon SNS の分散性により、実際のコストの方が高くなる場合があります。

API リクエスト (R) トピックごとの数を計算する場合は、次の式を使用します。

$$R = B / D * (2 * P)$$

B は請求期間(秒)です。

D は、データキー再利用期間 (秒単位で、Amazon SNS では、最大 5 分間データキーを再利用します) を表します。

P は、Amazon SNS トピックに送信する[プリンシパル](#)発行数です。

以下は計算の例です。正確な料金については、「[AWS Key Management Service 料金表](#)」を参照してください。

例 1: 1 つのパブリッシャーと 1 つのトピックの AWS KMS API コールの数进行計算する

この例では、以下を想定しています。

- 請求期間は1月1日から31日(2,678,400秒)です。
- データキー再利用期間は 5 分 (300 秒) です。
- 1 トピックあります。
- 公開中のプリンシパルが 1 つあります。

$$2,678,400 / 300 * (2 * 1) = 17,856$$

例 2: 複数のパブリッシャーと 2 つのトピックの AWS KMS API コールの数进行計算する

この例では、以下を想定しています。

- 請求期間は2月1日から28日(2,419,200秒) です。
- データキー再利用期間は 5 分 (300 秒) です。
- 2 トピックあります。
- 最初のトピックには、公開中のプリンシパルが 3 つあります。
- 2 つめのトピックには、公開中のプリンシパルが 5 つあります。

$$(2,419,200 / 300 * (2 * 3)) + (2,419,200 / 300 * (2 * 5)) = 129,024$$

AWS KMS アクセス許可の設定

SSE を使用する前に、トピックの暗号化とメッセージの暗号化と復号を許可するように AWS KMS key ポリシーを設定する必要があります。AWS KMS アクセス許可の詳細については、『AWS Key Management Service デベロッパーガイド』の「[AWS KMS API アクセス許可: アクションとリソースのリファレンス](#)」を参照してください。サーバー側の暗号化を使用して Amazon SNS トピックを設定する方法の詳細については、「[追加情報](#)」を参照してください。

Note

IAM ポリシーを使用して対称暗号化 KMS キーのアクセス許可を管理することもできます。詳細については、「[での IAM ポリシーの使用 AWS KMS](#)」を参照してください。Amazon SNS との間の送受信のグローバルなアクセス許可を設定できますが、AWS KMS では IAM ポリシーの Resource セクションにおいて、特定リージョンで KMS の完全 ARN を明示的に指定することが求められます。

また、のキーポリシーで必要なアクセス許可が AWS KMS key 付与されていることを確認する必要があります。そのためには、Amazon SNS で暗号化されたメッセージを作成するプリンシパルと消費するプリンシパルをユーザーとして KMS キーポリシーで指定します。

または、Amazon SNS で暗号化されたメッセージを受信するために発行およびサブスクライブするプリンシパルに割り当てられた IAM ポリシーで、必要な AWS KMS アクションと KMS ARN を指定することもできます。詳細については、AWS Key Management Service デベロッパーガイドの「[AWS KMSへのアクセス管理](#)」を参照してください。

Amazon SNS トピックのカスタマー管理キーを選択し、エイリアスを使用して IAM ポリシーまたは KMS キーポリシーと条件キー `kms:ResourceAliases` により KMS キーへのアクセスを制御している場合は、選択したカスタマー管理キーにもエイリアスが関連付けられていることを確認してください。エイリアスを使用して KMS キーへのアクセスを制御する方法の詳細については、「AWS Key Management Service デベロッパーガイド」の「[エイリアスを使用して KMS キーへのアクセスを制御する](#)」を参照してください。

SSE を使用したトピックへのメッセージの送信をユーザーに許可する

パブリッシャーには、AWS KMS key に対する `kms:GenerateDataKey*` および `kms:Decrypt` アクセス許可が必要です。

```
{
```

```
"Statement": [{
  "Effect": "Allow",
  "Action": [
    "kms:GenerateDataKey*",
    "kms:Decrypt"
  ],
  "Resource": "arn:aws:kms:us-east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}, {
  "Effect": "Allow",
  "Action": [
    "sns:Publish"
  ],
  "Resource": "arn:aws:sns:*:123456789012:MyTopic"
}]
}
```

AWS サービスからのイベントソースと暗号化されたトピック間の互換性を有効にする

いくつかの AWS サービスはAmazon SNSトピックにイベントを発行します。これらのイベントソースで暗号化されたトピックを操作できるようにするには、以下のステップを実行します。

1. カスタマーマネージドキーを使用します。詳細については、AWS Key Management Service デベロッパーガイドの[キーの作成](#)を参照してください。
2. AWS サービスに `kms:GenerateDataKey*` および `アクセスkms:Decrypt` 許可を付与するには、KMS ポリシーに次のステートメントを追加します。

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "service.amazonaws.com"
    },
    "Action": [
      "kms:GenerateDataKey*",
      "kms:Decrypt"
    ],
    "Resource": "*"
  }]
}
```

イベントソース	サービスプリンシパル
Amazon CloudWatch	cloudwatch.amazonaws.com
Amazon CloudWatch Events	events.amazonaws.com
AWS CodeCommit	codecommit.amazonaws.com
AWS Database Migration Service	dms.amazonaws.com
AWS Directory Service	ds.amazonaws.com
Amazon DynamoDB	dynamodb.amazonaws.com
Amazon Inspector	inspector.amazonaws.com
Amazon Redshift	redshift.amazonaws.com
Amazon RDS	events.rds.amazonaws.com
Amazon Glacier	glacier.amazonaws.com
Amazon Simple Email Service	ses.amazonaws.com
Amazon Simple Storage Service	s3.amazonaws.com
AWS Snowball Edge	importexport.amazonaws.com
AWS Systems Manager Incident Manager	AWS Systems Manager Incident Manager は、次の 2 つのサービス原則で構成されています。 ssm-incidents.amazonaws.com ; ssm-contacts.amazonaws.com

 Note

一部の Amazon SNS イベントソースでは、AWS KMS key ポリシーで (サービスプリンシパルではなく) IAM ロールを指定する必要があります。

- [Amazon EC2 Auto Scaling](#)
- [Amazon Elastic Transcoder](#)
- [AWS CodePipeline](#)
- [AWS Config](#)
- [AWS Elastic Beanstalk](#)
- [AWS IoT](#)
- [EC2 イメージビルダー](#)

3. KMS リソースポリシーに `aws:SourceAccount` キーと `aws:SourceArn` 条件キーを追加して、[混乱した代理](#) 攻撃から KMS キーをさらに保護します。それぞれの場合の正確な詳細については、サービス固有のドキュメントリスト (上記) を参照してください。

 Important

AWS KMS ポリシー `aws:SourceOrgID` への `aws:SourceAccount`、`aws:SourceArn` の追加は、EventBridge-to-encrypted トピックではサポートされていません。

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "service.amazonaws.com"
  },
  "Action": [
    "kms:GenerateDataKey*",
    "kms:Decrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "customer-account-id"
    },
    "ArnLike": {
      "aws:SourceArn": "arn:aws:service:region:customer-account-id:resource-type:customer-resource-id"
    }
  }
}
```

```
}
```

4. KMS を使用して、[トピックの SSE を有効化](#)します。
5. 暗号化されたトピックの ARN をイベントソースに追加します。

AWS KMS エラー

Amazon SNS と を使用すると AWS KMS、エラーが発生する可能性があります。次のリストは、エラーおよび考えられるトラブルシューティング策を示しています。

KMSAccessDeniedException

暗号化テキストは、存在しないキーまたはアクセス権限のないキーを参照しています。

HTTP ステータスコード: 400

KMSDisabledException

指定された KMS が有効ではないため、リクエストが拒否されました。

HTTP ステータスコード: 400

KMSInvalidStateException

指定されたリソースの状態がこのリクエストに対して有効ではないため、リクエストが拒否されました。詳細については、「AWS Key Management Service 開発者ガイド」の「[AWS KMS keys のキーステータス](#)」を参照してください。

HTTP ステータスコード: 400

KMSNotFoundException

指定されたエンティティまたはリソースが見つからないため、リクエストが拒否されました。

HTTP ステータスコード: 400

KMSOptInRequired

AWS アクセスキー ID には、サービスのサブスクリプションが必要です。

HTTP ステータスコード: 403

KMSThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。スロットリングの詳細については、「AWS Key Management Service デベロッパーガイド」の「[クォータ](#)」を参照してください。

HTTP ステータスコード: 400

サーバー側の暗号化を使用した Amazon SNS トピック暗号化のセットアップ

Amazon SNS は、サーバー側の暗号化 (SSE) をサポートし、AWS Key Management Service (AWS KMS) を使用してメッセージの内容を保護します。Amazon SNS コンソールまたは CDK を使用して SSE を有効にするには、次の手順に従います。

オプション 1: を使用して暗号化を有効にする AWS マネジメントコンソール

1. [Amazon SNS コンソール](#)にサインインします。
2. [トピック] ページに移動し、[トピック] を選択して、[編集] を選択します。
3. [暗号化] セクションを展開し、以下の操作を実行します。
 - 暗号化を [有効] に切り替えます。
 - [AWS マネージド SNS キー] (alias/aws/sns) を暗号化キーとして選択します。これはデフォルトで選択されています。
4. [Save changes] (変更の保存) をクリックします。

Note

- が存在しない場合 AWS マネージドキー、 は自動的に作成されます。
- キーが表示されない場合、またはアクセス許可が不十分な場合は、管理者に `kms:ListAliases` と `kms:DescribeKey` のアクセス許可を依頼してください。

オプション 2: を使用して暗号化を有効にする AWS CDK

CDK アプリケーションで AWS マネージド SNS キーを使用するには、次のスニペットを追加します。

```
import software.amazon.awscdk.services.sns.*;
import software.amazon.awscdk.services.kms.*;
import software.amazon.awscdk.core.*;

public class SnsEncryptionExample extends Stack {
    public SnsEncryptionExample(final Construct scope, final String id) {
        super(scope, id);

        // Define the managed SNS key
        IKey snsKey = Alias.fromAliasName(this, "helloKey", "alias/aws/sns");

        // Create the SNS Topic with encryption enabled
        Topic.Builder.create(this, "MyEncryptedTopic")
            .masterKey(snsKey)
            .build();
    }
}
```

追加情報

- カスタム KMS キー – 必要に応じてカスタムキーを指定できます。Amazon SNS コンソールで、リストからカスタム KMS キーを選択するか、ARN を入力します。
- カスタム KMS キーのアクセス許可 – カスタム KMS キーを使用する場合は、Amazon SNS がメッセージを暗号化および復号化できるように、キーポリシーに以下を含めます。

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
```



```
    "ArnLike": {
      "aws:SourceArn": "arn:aws:service:region:customer-account-id:resource-
type/customer-resource-id"
    },
    "StringEquals": {
      "kms:EncryptionContext:aws:sns:topicArn":
"arn:aws:sns:your_region:customer-account-id:your_sns_topic_name"
    }
  }
}
```

コンシューマーへの影響

SSE を有効にしても、サブスクライバーがメッセージを使用する方法は変わりません。は暗号化と復号を透過的に AWS 管理します。メッセージは保管中も暗号化されたままになり、サブスクライバーに配信する前に自動的に復号されます。セキュリティを最適化するために、AWS では、すべてのエンドポイントで HTTPS を有効にして、メッセージを確実かつ安全に送信することを推奨しています。

暗号化された Amazon SQS キューサブスクリプションによる Amazon SNS トピック暗号化の設定

トピックのサーバー側の暗号化 (SSE) を有効にしてトピックのデータを保護できます。暗号化された Amazon SQS キューにメッセージを送信することを Amazon SNS に許可するには、Amazon SQS キューに関連付けられたカスタマーマネージドキーのポリシーステートメントにより、AWS KMS API アクション (GenerateDataKey と Decrypt) へのアクセス権を Amazon SNS サービスプリンシパルに付与する必要があります。SSE の使用の詳細については、「[サーバー側の暗号化を使用した Amazon SNS データの保護](#)」を参照してください。

このトピックでは、AWS マネジメントコンソールを使用して、暗号化された Amazon SQS キューサブスクリプションを持つ Amazon SNS トピックの SSE を有効にする方法について説明します。

ステップ 1: カスタム KMS キーを作成する

1. 少なくとも AWSKeyManagementServicePowerUser ポリシーを持つユーザーで [AWS KMS コンソール](#) にサインインします。
2. [キーの作成] を選択します。
3. 対称暗号化 KMS キーを作成するには、[Key type] (キーのタイプ) で [Symmetric] (対称) を選択します。

AWS KMS コンソールで非対称 KMS キーを作成する方法については、「[非対称 KMS キーを作成する \(コンソール\)](#)」を参照してください。

4. [Key usage] (キーの使用) では、[Encrypt and decrypt] (暗号化および復号) オプションがすでに選択されています。

MAC コードを生成して検証する KMS キーの作成方法については、「[HMAC KMS キーの作成](#)」を参照してください。

[詳細オプション] については、「[特定用途のキー](#)」を参照してください。

5. [次へ] を選択します。
6. KMS キーのエイリアスを入力します。エイリアス名の先頭を **aws/** にすることはできません。**aws/** プレフィックスは、アカウント AWS マネージドキー で表すために Amazon Web Services によって予約されています。

 Note

エイリアスを追加、削除、更新すると、KMS キーに対するアクセス許可が許可または拒否される可能性があります。詳細については、「[AWS KMS の ABAC](#)」および「[エイリアスを使用して KMS キーへのアクセスを制御する](#)」を参照してください。

エイリアスは KMS キーを識別するために使用する表示名です。保護する予定のデータタイプ、または KMS キーで使用する予定のアプリケーションを示すエイリアスを選択することをお勧めします。

エイリアスは AWS マネジメントコンソールで KMS キーを作成するときに必要です。[CreateKey](#) オペレーションを使用する場合、これらのオペレーションはオプションです。

7. (オプション) KMS キーの説明を入力します。

今すぐ説明を追加するか、[キーの状態](#)が Pending Deletion または Pending Replica Deletion でない限り、後でいつでも更新できます。既存のカスタマーマネージドキーの説明を追加、変更、または削除するには、[説明を編集する](#) AWS マネジメントコンソールか、[UpdateKeyDescription](#) オペレーションを使用します。

8. (オプション) タグキーとオプションのタグ値を入力します。KMS キーに複数のタグを追加するには、[Add tag] (タグを追加) を選択します。

Note

KMS キーのタグ付けまたはタグ解除により、KMS キーに対するアクセス許可が許可または拒否される可能性があります。詳細については、「[AWS KMSの ABAC](#)」および「[タグを使用してKMS キーへのアクセスを制御する](#)」を参照してください。

AWS リソースにタグを追加すると、AWS は使用量とコストをタグ別に集計したコスト配分レポートを生成します。タグは、KMS キーへのアクセスの制御にも使用できます。KMS キーのタグ付けについては、「[キーのタグ付け](#)」および「[AWS KMSの ABAC](#)」を参照してください。

9. [次へ] を選択します。

10. KMS キーを管理できる IAM ユーザーとロールを選択します。

Note

このキーポリシーは、この KMS キー AWS アカウント を完全に制御します。これにより、アカウント管理者は IAM ポリシーを使用して、他のプリンシパルに KMS キーを管理する許可を付与できます。詳細については、「[デフォルトのキーポリシー](#)」を参照してください。

IAM ベストプラクティスでは、長期の認証情報を持つ IAM ユーザーの使用は推奨されていません。可能な限り、一時的な認証情報を提供する IAM ロールを使用してください。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。

11. (オプション) 選択した IAM ユーザーとロールがこの KMS キーを削除しないようにするには、ページの下部にある [Key deletion] (キーの削除) セクションで、[Allow key administrators to delete this key] (キー管理者にこのキーの削除を許可する) のチェックボックスをオフにします。

12. [次へ] を選択します。

13. [暗号化オペレーション](#) でキーを使用できる IAM ユーザーとロールを選択します。[次へ] を選択します。

14. [キーポリシーの表示と編集] ページで、次のステートメントをキーポリシーに追加し、[完了] を選択します。

```
{  
  "Sid": "Allow Amazon SNS to use this key",
```

```
"Effect": "Allow",
"Principal": {
  "Service": "sns.amazonaws.com"
},
"Action": [
  "kms:Decrypt",
  "kms:GenerateDataKey*"
],
"Resource": "*"
}
```

新しいカスタマーマネージドキーがキーのリストに表示されます。

ステップ 2: 暗号化された Amazon SNS トピックを作成する

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[トピック] を選択します。
3. [トピックを作成] を選択します。
4. [新しいトピックの作成] ダイアログボックスの [トピック名] に、トピックの名前 (例: MyEncryptedTopic) を入力し、[トピックの作成] を選択します。
5. [暗号化] セクションを展開し、以下の操作を実行します。
 - a. [サーバー側の暗号化を有効にする] を選択します。
 - b. カスタマーマネージドキーを指定します。詳細については、「[重要な用語](#)」を参照してください。

カスタマーマネージドキーのタイプごとに、説明、アカウント、およびカスタマーマネージドキー ARN が表示されます。

Important

カスタマーマネージドキーの所有者ではない場合、または `kms:ListAliases` および `kms:DescribeKey` の許可がないアカウントでログインした場合、Amazon SNS コンソールでカスタマーマネージドキーに関する情報を表示できません。これらの許可を付与するように、カスタマーマネージドキーの所有者へ依頼してください。詳細については、「AWS Key Management Service デベロッパーガイド」

の「[AWS KMS API アクセス権限: アクションとリソースのリファレンス](#)」を参照してください。

- c. [カスタマーマネージドキー] で、[以前に作成した](#) [MyCustomKey] を選択してから、[サーバー側の暗号化を有効化] を選択します。
6. [Save changes] (変更の保存) をクリックします。

トピックの SSE が有効になり、[マイトピック] ページが表示されます。

トピックの暗号化ステータス、AWS アカウント、カスタマーマネージドキー、カスタマーマネージドキーの ARN、および説明が [暗号化] タブに表示されます。

暗号化された新しいトピックが、トピックのリストに表示されます。

ステップ 3: 暗号化された Amazon SQS キューを作成してサブスクライブする

1. [Amazon SQS コンソール](#)にサインインします。
2. [新しいキューの作成] を選択します。
3. [新しいキューの作成] ページで、次の操作を実行します。
 - a. [キュー名] (例: MyEncryptedQueue1) を入力します。
 - b. [標準キュー]、[キューの設定] の順に選択します。
 - c. [SSE の使用] を選択します。
 - d. [AWS KMS key] で、[以前に作成した](#) [MyCustomKey] を選択してから、[キューの作成] を選択します。
4. プロセスを繰り返して、2 つめのキュー (例えば、MyEncryptedQueue2) を作成します。

暗号化された新しいキューがキューのリストに表示されます。

5. Amazon SQS コンソールで、MyEncryptedQueue1 および MyEncryptedQueue2 を選択して、[キューのアクション]、[SNS トピックへのキューのサブスクライブ] の順に選択します。
6. [トピックへのサブスクライブ] ダイアログボックスの [トピックの選択] で、[MyEncryptedTopic]、[サブスクライブ] の順に選択します。

暗号化されたトピックに対する暗号化されたキューのサブスクリプションは、[トピックのサブスクリプション結果] ダイアログボックスに表示されます。

7. [OK] を選択してください。

ステップ 4: 暗号化されたトピックにメッセージを発行する

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションパネルで、[トピック] を選択します。
3. トピックのリストから [MyEncryptedTopic] を選択し、[メッセージの発行] を選択します。
4. [メッセージの発行] ページで、次の操作を行います。
 - a. (オプション) [メッセージの詳細] セクションで、[件名] (Testing message publishing など) を入力します。
 - b. [メッセージの本文] セクションで、メッセージの本文 (My message body is encrypted at rest. など) を入力します。
 - c. [メッセージの発行] を選択します。

メッセージは、サブスクライブされた暗号化キューに発行されます。

ステップ 5: メッセージの配信を確認する

1. [Amazon SQSコンソール](#)にサインインします。
2. キューのリストから [MyEncryptedQueue1] を選択し、次に [Send and receive messages] (メッセージの送受信) を選択します。
3. [Send and receive messages in MyEncryptedQueue1] (MyEncryptedQueue1 のメッセージの送信と受信) ページで、[Poll for messages] (メッセージのポーリング) を選択します。

[先ほど送信した](#)メッセージが表示されます。

4. メッセージを表示するには、[詳細] を選択します。
5. 完了したら、[閉じる] をクリックします。
6. [MyEncryptedQueue2] に対してこの処理を繰り返します。

VPC エンドポイントによる Amazon SNS トラフィックの保護

Amazon SNS の Amazon Virtual Private Cloud (Amazon VPC) エンドポイントは、Amazon SNS への接続のみを許可する VPC 内の論理エンティティです。VPC はリクエストを Amazon SNS にルーティングし、レスポンスを VPC にルーティングします。以下のセクションでは、VPC エンドポイントの使用と VPC エンドポイントポリシーの作成について説明します。

Amazon Virtual Private Cloud (Amazon VPC) を使用して AWS リソースをホストする場合は、VPC と Amazon SNS の間にプライベート接続を確立できます。この接続では、メッセージをパブリックインターネット経由で送信せずに、Amazon SNS トピックに発行できます。

Amazon VPC は、定義した仮想ネットワークで AWS リソースを起動するために使用できる AWS サービスです。VPC を使用することで、IP アドレス範囲、サブネット、ルートテーブル、ネットワークゲートウェイなどのネットワーク設定を制御できます。VPC を Amazon SNS に接続するには、インターフェイス VPC エンドポイントを定義します。このタイプのエンドポイントを使用すると、VPC を AWS サービスに接続できます。このエンドポイントは、インターネットゲートウェイ、ネットワークアドレス変換 (NAT) インスタンス、または VPN 接続を必要とせず、信頼性が高くスケーラブルな Amazon SNS への接続を提供します。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 VPC エンドポイント AWS のサービスを使用してにアクセスする](#)」を参照してください。

このセクションの情報は、Amazon VPC のユーザーを対象にしています。VPC の詳細を確認したり、作成を開始したりするには、「Amazon VPC ユーザーガイド」の「[VPC のプランニング](#)」を参照してください。

Note

VPC エンドポイントでは、Amazon SNS トピックをプライベート IP アドレスにサブスクライブすることはできません。

Amazon SNS 用の Amazon VPC エンドポイントの作成

Amazon VPC から Amazon SNS トピックにメッセージを発行するには、インターフェイス VPC エンドポイントを作成します。次に、VPC で管理するネットワーク内でトラフィックを維持しながら、トピックにメッセージを発行できます。

次の情報を使用してエンドポイントを作成し、VPC と Amazon SNS 間の接続をテストします。一から開始する作業に役立つチュートリアルについては、「[Amazon VPC から Amazon SNS メッセージを発行する](#)」を参照してください。

エンドポイントの作成

VPC に Amazon SNS エンドポイントを作成するには、AWS マネジメントコンソール、AWS CLI、AWS SDK、Amazon SNS API、または AWS CloudFormation を使用します。

Amazon VPC コンソールまたは AWS CLI を使用して、エンドポイントを作成および設定する方法については、『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の作成](#)」を参照してください。

Important

Amazon Virtual Private Cloud は HTTPS Amazon SNS エンドポイントでのみ使用できます。エンドポイントを作成するとき、VPC の接続先のサービスとして Amazon SNS を指定します。Amazon VPC コンソールで、このサービス名は選択したリージョンによって異なります。例えば、米国東部 (バージニア北部) を選択した場合、サービス名は `com.amazonaws.us-east-1.states` になります。

Amazon VPC からメッセージを送信するように Amazon SNS を設定する場合、プライベート DNS を有効にして、`sns.us-east-2.amazonaws.com` の形式でエンドポイントを指定する必要があります。

プライベート DNS は、`queue.amazonaws.com` や `us-east-2.queue.amazonaws.com` などのレガシーエンドポイントをサポートしていません。

AWS CloudFormation を使用してエンドポイントを作成および設定する方法については、『AWS CloudFormation ユーザーガイド』の「[AWS::EC2::VPCEndpoint](#) リソース」を参照してください。

VPC と Amazon SNS との間の接続をテストする

Amazon SNS のエンドポイントを作成したら、VPC から Amazon SNS トピックにメッセージを発行できます。この接続をテストするには、以下の手順を実行します。

1. VPC にある Amazon EC2 インスタンスに接続します。接続の詳細については、Amazon EC2 ドキュメントの [Linux インスタンスへの接続](#) または [Windows インスタンスへの接続](#) を参照してください。

例えば、SSH クライアントを使用して Linux インスタンスに接続するには、ターミナルから以下のコマンドを実行します。

```
$ ssh -i ec2-key-pair.pem ec2-user@instance-hostname
```

実行する条件は以下の通りです。

- `ec2-key-pair.pem` は、インスタンスの作成時に Amazon EC2 から提供されたキーペアを含むファイルです。
 - `instance-hostname` は、インスタンスのパブリックホスト名です。[Amazon EC2 コンソール](#)でホスト名を取得するには、[インスタンス] を選択してから、目的のインスタンスを選び、[パブリック DNS] の値を見つけます。
2. インスタンスから、AWS CLI で Amazon SNS [publish](#) コマンドを使用します。以下のコマンドを使用して、シンプルなメッセージをトピックに送信できます。

```
$ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"
```

コードの説明は以下のとおりです。

- `aws-region` は、トピックがある AWS リージョンです。
- `sns-topic-arn` は、トピックの Amazon リソースネーム (ARN) です。[Amazon SNS コンソール](#)から ARN を取得するには: [トピック] を選択し、自分のトピックを検索して、[ARN] カラムでその値を見つけます。

メッセージが Amazon SNS によって正常に受信された場合、ターミナルには以下のようなメッセージ ID が表示されます。

```
{
  "MessageId": "6c96dfff-0fdf-5b37-88d7-8cba910a8b64"
}
```

Amazon SNS 用の VPC エンドポイントポリシーを作成する

Amazon SNS の Amazon VPC エンドポイントに対するポリシーを作成して、以下を指定することができます。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、『Amazon VPC ユーザーガイド』の「[VPC エンドポイントでサービスへのアクセスを制御する](#)」を参照してください。

以下の例の VPC エンドポイントポリシーでは、Amazon SNS トピック MyTopic に対する発行を IAM ユーザー MyUser に許可することを指定します。

```
{
  "Statement": [{
    "Action": ["sns:Publish"],
    "Effect": "Allow",
    "Resource": "arn:aws:sns:us-east-2:123456789012:MyTopic",
    "Principal": {
      "AWS": "arn:aws:iam:123456789012:user/MyUser"
    }
  }]
}
```

以下は拒否されます：

- 他の Amazon SNS API アクション (sns:Subscribe や sns:Unsubscribe など)。
- この VPC エンドポイントを使用しようとする IAM の他のユーザーおよびルール。
- MyUser 別の Amazon SNS トピックに発行する。

Note

IAM ユーザーはさらに、他の Amazon SNS API アクションを VPC の外側 から使用します。

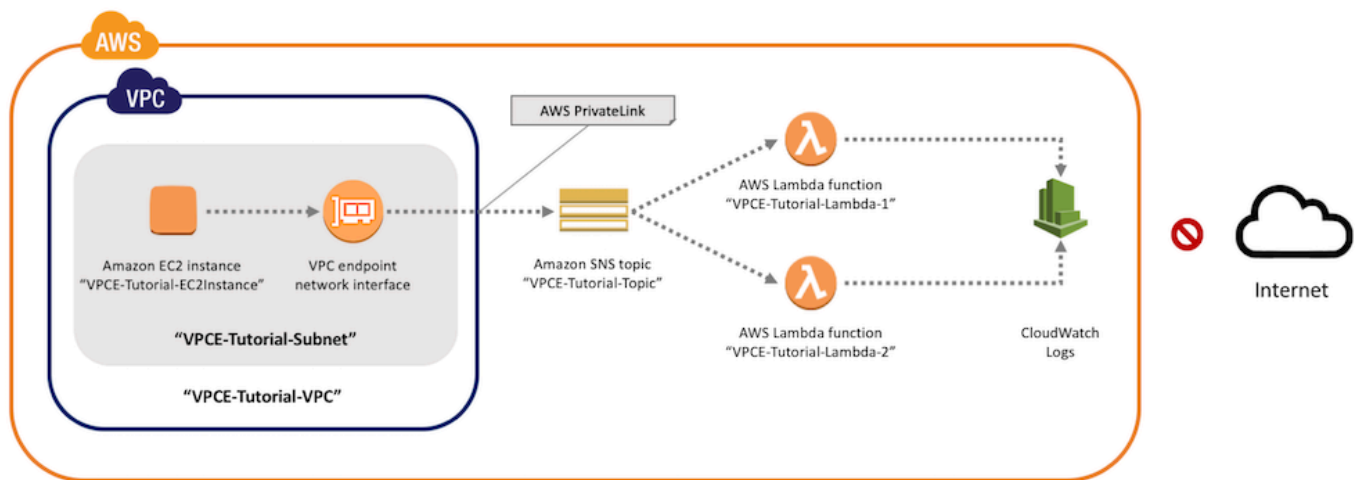
Amazon VPC から Amazon SNS メッセージを発行する

このセクションでは、プライベートネットワークでメッセージを安全に保ちながら、Amazon SNS トピックに発行する方法について説明します。Amazon Virtual Private Cloud (Amazon VPC) でホストされている Amazon EC2 インスタンスからメッセージを発行します。メッセージは、パブリックインターネットを経由せずに AWS ネットワーク内に留まります。VPC からプライベートにメッセージを発行することにより、アプリケーションと Amazon SNS 間のトラフィックのセキュリティを強化できます。このセキュリティは、顧客に関する個人を特定できる情報 (PII) を公開する場合や、アプリケーションが市場規制の対象となる場合に重要になります。例えば、プライベートな発行は、Health Insurance Portability and Accountability Act (HIPAA) に準拠する必要があるヘルスケアシステムや、Payment Card Industry Data Security Standard (PCI DSS) に準拠する必要がある財務システムがある場合に役立ちます。

一般的なステップは次のとおりです。

- AWS CloudFormation テンプレートを使用して、に一時的なプライベートネットワークを自動的に作成します AWS アカウント。
- VPC を Amazon SNS に接続する VPC エンドポイントを作成します。
- Amazon EC2 インスタンスにログインし、Amazon SNS トピックに対してプライベートにメッセージを発行します。
- メッセージが正常に配信されたことを確認します。
- このプロセス中に作成したリソースを削除して、に残らないようにします AWS アカウント。

次の図は、これらのステップを完了する際に AWS アカウントで作成するプライベートネットワークを示しています。



このネットワークは、Amazon EC2 インスタンスを含む VPC で構成されます。インスタンスは、インターフェイス VPC エンドポイントを介して Amazon SNS に接続します。このタイプのエンドポイントは、AWS PrivateLink を使用するサービスに接続します。この接続が確立されたら、ネットワークがパブリックインターネットから切断されている場合でも、Amazon EC2 インスタンスにログインして Amazon SNS トピックにメッセージを発行できます。トピックは、受信したメッセージを 2 つのサブスクライブ AWS Lambda 関数にファンアウトします。これらの関数は、受け取ったメッセージを Amazon CloudWatch Logs に記録します。

このステップの完了には 20 分ほどかかります。

トピック

- [\[開始する前に\]](#)
- [ステップ 1: Amazon EC2 キーペアを作成する](#)

- [ステップ 2: AWS リソースを作成する](#)
- [ステップ 3: Amazon EC2 インスタンスにインターネットアクセスがないことを確認する](#)
- [ステップ 4: Amazon SNS の Amazon VPC エンドポイントを作成する](#)
- [ステップ 5: Amazon SNS トピックにメッセージを発行する](#)
- [ステップ 6: メッセージの配信を確認する](#)
- [ステップ 7: クリーンアップする](#)
- [関連リソース](#)

[開始する前に]

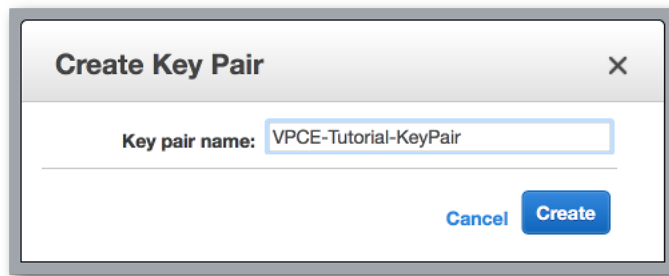
開始する前に、Amazon Web Services (AWS) アカウントが必要です。サインアップすると、Amazon SNS や Amazon VPC を含む AWS のすべてのサービスにアカウントが自動的にサインアップされます。アカウントをまだ作成していない場合は、<https://aws.amazon.com/> に移動し、[まずは無料で始める] を選択します。

ステップ 1: Amazon EC2 キーペアを作成する

Amazon EC2 インスタンスへのログインには、キーペアが使用されます。これは、ログイン情報の暗号化に使用されるパブリックキーと、その復号に使用されるプライベートキーで構成されます。キーペアを作成するときは、プライベートキーのコピーをダウンロードします。後で、キーペアを使用して Amazon EC2 インスタンスにログインします。ログインするには、キーペアの名前を指定し、プライベートキーを指定します。

キーペアを作成するには

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/ec2/> で Amazon EC2 コンソールを開きます。
2. 左のナビゲーションメニューで、[ネットワーク & セキュリティ] セクションを見つけます。次に、[キーペア] を選択します。
3. [キーペアの作成] を選択します。
4. [キーペアの作成] ウィンドウで、[キーペア名] に「**VPCE-Tutorial-KeyPair**」と入力します。続いて、[作成] を選択します。



5. ブラウザによってプライベートキーファイルが自動的にダウンロードされます。これを安全な場所に保存します。Amazon EC2 により、ファイルに拡張子 `.pem` が付けられます。
6. (オプション) Mac または Linux コンピュータで SSH クライアントを使用してインスタンスに接続している場合は、`chmod` コマンドを使用してプライベートキーファイルの権限を設定すると、お客様のみがそれを読み取ることができます。
 - a. ターミナルを開き、プライベートキーを含むディレクトリに移動します。

```
$ cd /filepath_to_private_key/
```

- b. 次のコマンドを使用してアクセス権限を設定します。

```
$ chmod 400 VPCE-Tutorial-KeyPair.pem
```

ステップ 2: AWS リソースを作成する

インフラストラクチャを設定するには、CloudFormation テンプレートを使用します。テンプレートは、Amazon EC2 インスタンスや Amazon SNS トピックなどの AWS リソースを構築するための設計図として機能するファイルです。このプロセスのテンプレートは GitHub で提供されていて、ダウンロードできます。

テンプレートを に提供し CloudFormation、 でスタックとして必要なリソースを CloudFormation プロビジョニングします AWS アカウント。スタックは、単一のユニットとして管理できるリソースのコレクションです。これらのステップを完了すると、CloudFormation を使用してスタック内のすべてのリソースを一度に削除できます。これらのリソースは、必要でない限り AWS アカウント、に残りません。

このプロセスのスタックには、次のリソースが含まれます。

- VPC および関連するネットワーキングリソース (サブネット、セキュリティグループ、インターネットゲートウェイ、およびルートテーブルを含む)。

- VPC 内のサブネットに起動された Amazon EC2 インスタンス。
- Amazon SNS トピック
- 2 つの AWS Lambda 関数。これらの関数は、Amazon SNS トピックに発行されたメッセージを受け取り、CloudWatch Logs にイベントを記録します。
- Amazon CloudWatch メトリクスおよびログ
- Amazon EC2 インスタンスに Amazon SNSの使用を許可する IAM ロール、および CloudWatch Logs への書き込みを Lambda 関数に許可する IAM ロール。

AWS リソースを作成するには

1. GitHub ウェブサイトから [テンプレート ファイル](#) をダウンロードします。
2. [CloudFormation コンソール](#) にサインインします。
3. [スタックの作成] を選択します。
4. [テンプレートの選択] ページで、[テンプレートを Amazon S3 にアップロード] を選択してから、ファイルを選択して [次へ] をクリックします。
5. [詳細の指定] ページで、スタック名とキー名を指定します。
 - a. [スタックの名前] に **VPCE-Tutorial-Stack** を入力します。
 - b. [KeyName] で、[VPCE-Tutorial-KeyPair] を選択します。
 - c. [SSHLocation] で、デフォルト値の **0.0.0.0/0** のままにします。

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)

Stack name

Parameters

KeyName Name of an existing EC2 KeyPair to enable SSH access to the instance

SSHLocation The IP address range that can be used to SSH to the EC2 instance

- d. [次へ] を選択します。
6. [オプション] ページで、すべてのデフォルト値を受け入れ、[次へ] を選択します。
 7. [確認] ページで、スタックの詳細を確認します。

8. 機能で、ガカスタム名で IAM リソースを作成する CloudFormation 可能性があることを確認します。
9. [作成] を選択します。

CloudFormation コンソールで スタックページが開きます。VPCE-Tutorial-Stack のステータスは CREATE_IN_PROGRESS です。数分後に作成プロセスが完了し、ステータスが CREATE_COMPLETE に変わります。

Create Stack

Actions

Design template

Filter: Active

By Stack Name

	Stack Name	Created Time	Status	Description
☒	VPCE-Tutorial-Stack	2018-05-18 16:38:06 UTC-0700	CREATE_COMPLETE	CloudFormation Template for SNS VPC Endpoints Tutorial

Tip

[更新] ボタンを選択して、スタックの最新のステータスを表示します。

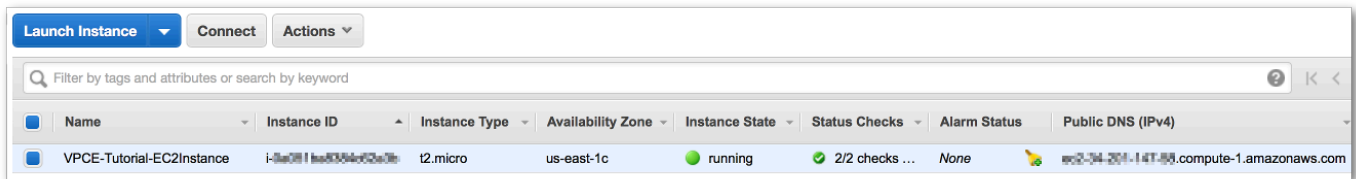
ステップ 3: Amazon EC2 インスタンスにインターネットアクセスがないことを確認する

前のステップの VPC で起動された Amazon EC2 インスタンスにはインターネットアクセスがありません。アウトバウンドトラフィックが禁止されているため、Amazon SNS にメッセージを発行することができません。インスタンスにログインしてこれを確認します。次に、パブリックエンドポイントへの接続と、Amazon SNS へのメッセージの発行を試みます。

この時点では、発行の試みは失敗します。後のステップで、Amazon SNS の VPC エンドポイントを作成すると、発行の試みは成功します。

Amazon EC2 インスタンスに接続します。

1. Amazon EC2 コンソールの <https://console.aws.amazon.com/ec2/> を開いてください。
2. 左のナビゲーションメニューで、[インスタンス] セクションを見つけます。続いて、[インスタンス] を選択します。
3. インスタンスの一覧で、[VPCE-Tutorial-EC2Instance] を選択します。
4. [Public DNS] 列に示されたホスト名をコピーします。



5. ターミナルを開きます。キーペアが含まれているディレクトリから、以下のコマンドを使用してインスタンスに接続します。ここで、*instance-hostname* は、Amazon EC2 コンソールからコピーしたホスト名です。

```
$ ssh -i VPCE-Tutorial-KeyPair.pem ec2-user@instance-hostname
```

インスタンスがインターネットに接続されていないことを確認するには

- ターミナルで、amazon.com など任意のパブリックエンドポイントへの接続を試します。

```
$ ping amazon.com
```

接続の試行が失敗するため、いつでもキャンセルできます (Windows では Ctrl + C、macOS では Command + C)。

インスタンスが Amazon SNS に接続されていないことを確認するには

- [Amazon SNS コンソール](#) にサインインします。
- 左側のナビゲーションメニューで、[トピック] を選択します。
- [トピック] ページで、トピック [VPCE-Tutorial-Topic] の Amazon リソースネーム (ARN) をコピーします。
- ターミナルで、トピックへのメッセージを発行を試みます。

```
$ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"
```

発行の試みが失敗するため、いつでもキャンセルできます。

ステップ 4: Amazon SNS の Amazon VPC エンドポイントを作成する

VPC を Amazon SNS に接続するには、インターフェイス VPC エンドポイントを定義します。エンドポイントを追加したら、VPC の Amazon EC2 インスタンスにログインし、そこから Amazon

SNS API を使用できます。トピックにメッセージを発行でき、メッセージはプライベートに発行されます。AWS ネットワーク内にとどまり、パブリックインターネットは移動しません。

Note

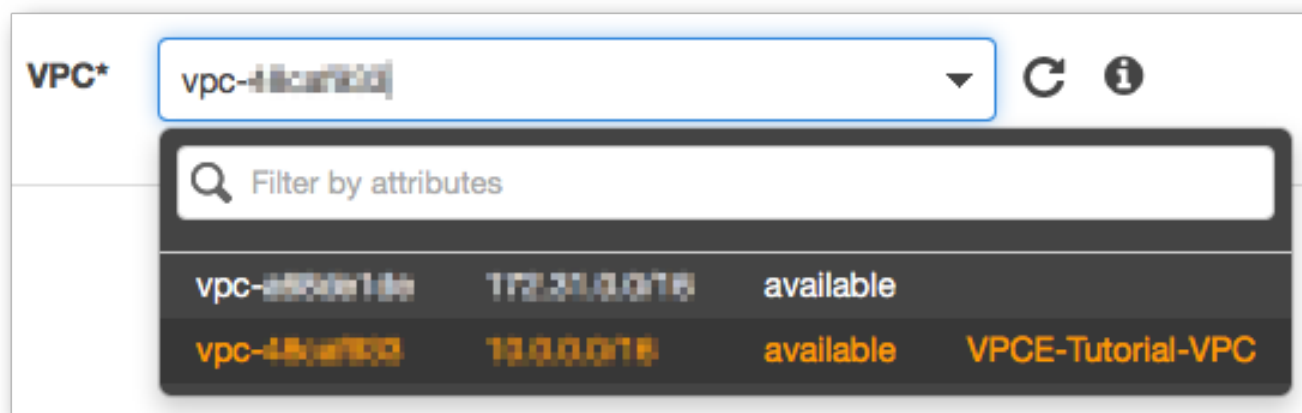
インスタンスは、インターネット上の他の AWS サービスやエンドポイントにアクセスできません。

エンドポイントを作成するには

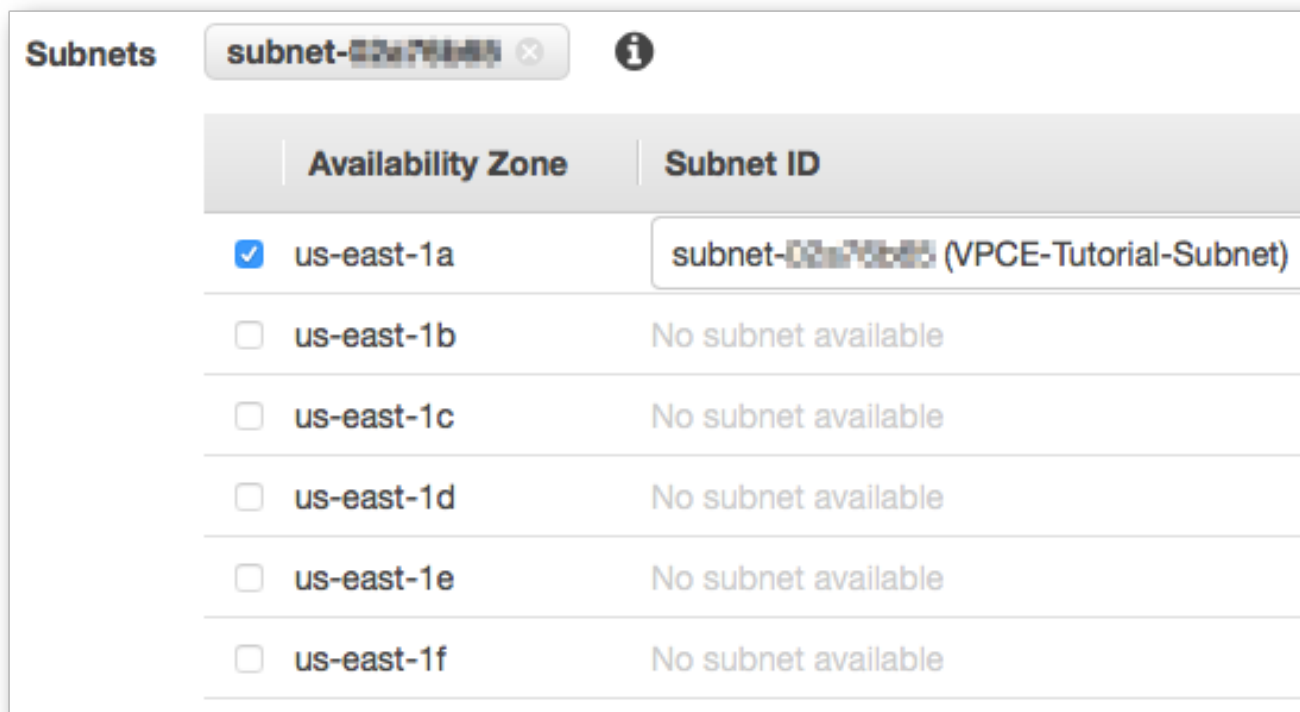
1. Amazon VPC コンソールの <https://console.aws.amazon.com/vpc/> を開いてください。
2. 左側のナビゲーションメニューで、[エンドポイント] を選択します。
3. [エンドポイントの作成] を選択します。
4. [エンドポイントの作成] ページの [サービスカテゴリ] で、デフォルトの選択である [AWS サービス] をそのままにします。
5. [サービス名] で、Amazon SNS のサービス名を選択します。

このサービス名は、選択したリージョンによって異なります。例えば、米国東部 (バージニア北部) を選択した場合、サービス名は com.amazonaws.**us-east-1**.sns になります。

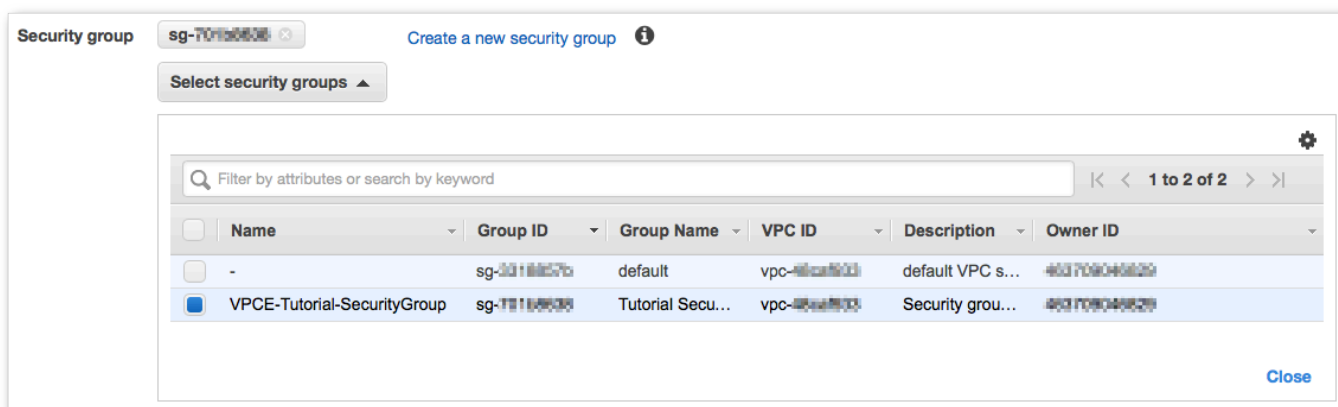
6. [VPC] で、[VPCE-Tutorial-VPC] という名前の VPC を選択します。



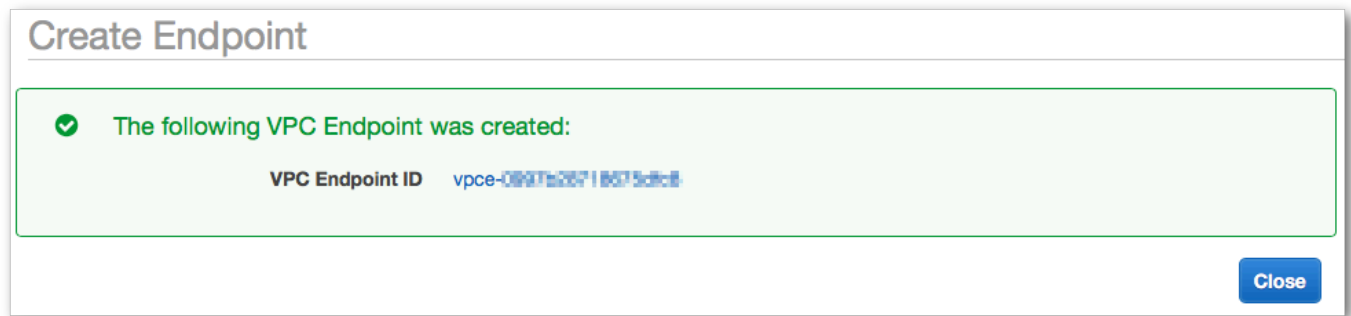
7. [サブネット] で、サブネット ID に VPCE-Tutorial-Subnet を持つサブネットを選択します。



8. [プライベート DNS 名を有効にする] で、[このエンドポイントで有効にする] を選択します。
9. [セキュリティグループ] で、[セキュリティグループの選択] を選択し、[VPC-Tutorial-SecurityGroup] を選択します。

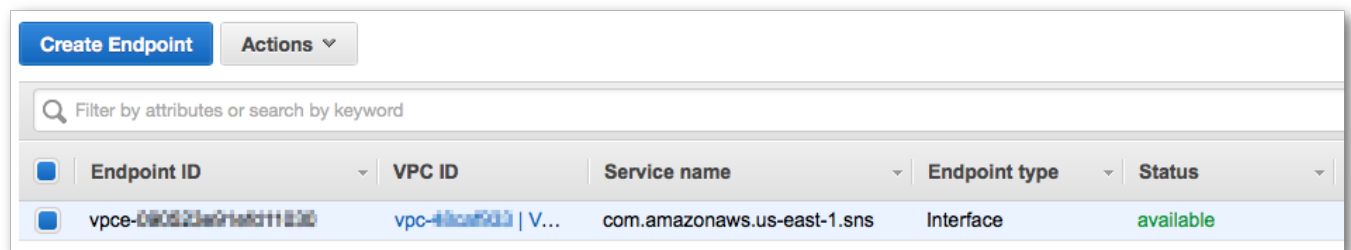


10. [エンドポイントの作成] を選択します。Amazon VPC コンソールで、VPC エンドポイントが作成されたことが確認されます。



11. [閉じる] を選択します。

Amazon VPC コンソールの [エンドポイント] ページを開きます。新しいエンドポイントのステータスは [保留中] です。数分で、作成プロセスが完了すると、ステータスが [利用可能] に変わります。



ステップ 5: Amazon SNS トピックにメッセージを発行する

これで VPC に Amazon SNS のエンドポイントが含まれたので、Amazon EC2 インスタンスにログインし、トピックにメッセージを発行できます。

メッセージを発行するには

1. ターミナルが Amazon EC2 インスタンスに接続されていない場合は、再度接続します。

```
$ ssh -i VPCE-Tutorial-KeyPair.pem ec2-user@instance-hostname
```

2. 以前の手順と同じコマンドを実行して、Amazon SNS トピックにメッセージを発行します。今回は、発行の試みが成功すると、Amazon SNS はメッセージ ID を返します。

```
$ aws sns publish --region aws-region --topic-arn sns-topic-arn --message "Hello"

{
```

```
"MessageId": "5b111270-d169-5be6-9042-410dfc9e86de"
}
```

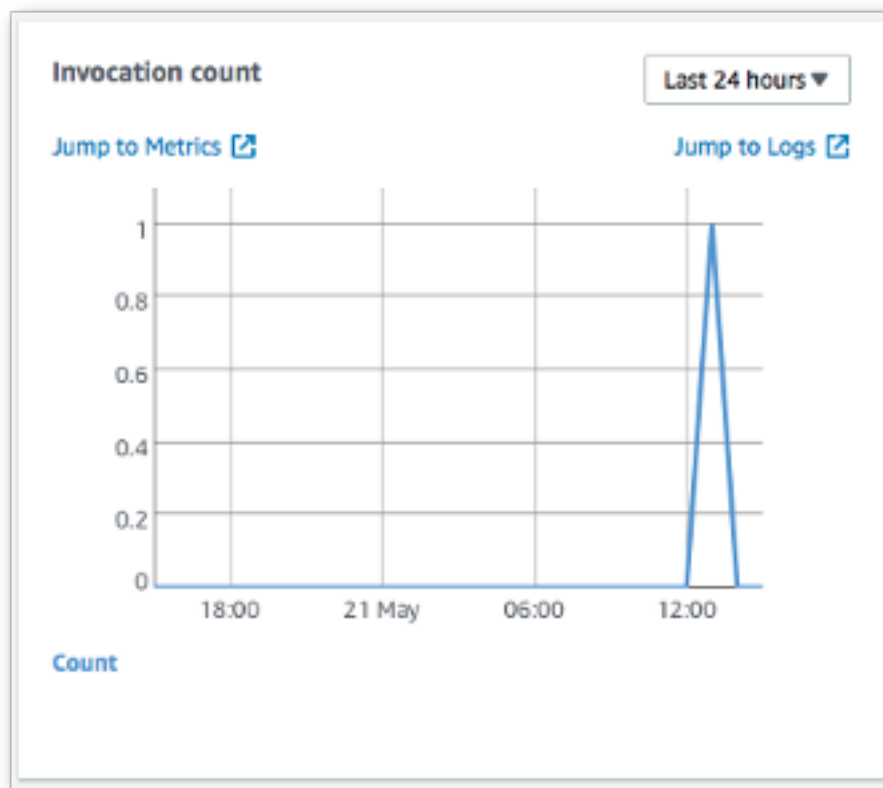
ステップ 6: メッセージの配信を確認する

Amazon SNS トピックがメッセージを受け取ると、2 つの Lambda サブスクライブ関数に送信して、メッセージをファンアウトします。これらの関数がメッセージを受け取ると、イベントを CloudWatch Logs に記録します。メッセージの配信が成功したことを確認するには、関数が呼び出されたこと、および CloudWatch Logs が更新されたことを確認します。

Lambda 関数が呼び出されたことを確認するには

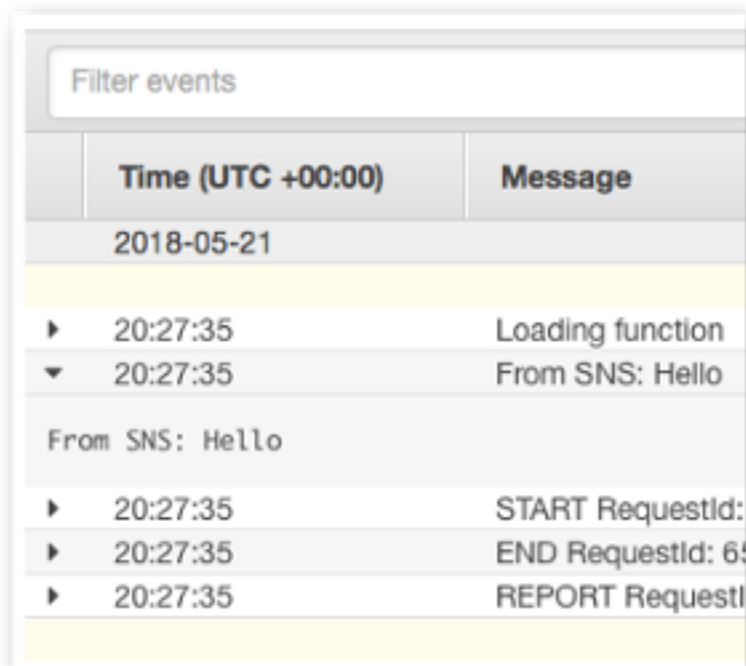
1. <https://console.aws.amazon.com/lambda/> で AWS Lambda コンソールを開きます。
2. [関数] ページで、[VPCE-Tutorial-Lambda-1] を選択します。
3. [モニタリング] を選択します。
4. [呼び出しカウント] グラフを確認します。このグラフには、Lambda 関数が実行された回数が表示されます。

呼び出しカウントは、トピックにメッセージを発行した回数に一致します。



CloudWatch Logs が更新されたことを確認するには

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. 左側のナビゲーションメニューで [ログ] を選択します。
3. Lambda 関数によって書き込まれたログを確認します。
 - a. [/aws/lambda/VPCE-Tutorial-Lambda-1/] ロググループを選択します。
 - b. ログストリームを選択します。
 - c. ログにエントリ From SNS: Hello が含まれていることを確認します。



The screenshot shows the AWS CloudWatch Logs console. At the top, there is a 'Filter events' input field. Below it is a table with two columns: 'Time (UTC +00:00)' and 'Message'. The table is filtered for the date '2018-05-21'. The log stream is expanded, showing several entries. The entry of interest is 'From SNS: Hello' at 20:27:35. Other entries include 'Loading function', 'START RequestId:', 'END RequestId: 65', and 'REPORT RequestId:'.

	Time (UTC +00:00)	Message
2018-05-21		
▶	20:27:35	Loading function
▼	20:27:35	From SNS: Hello
From SNS: Hello		
▶	20:27:35	START RequestId:
▶	20:27:35	END RequestId: 65
▶	20:27:35	REPORT RequestId:

- d. コンソール上部の [ロググループ] を選択して、[ロググループ] ページを表示します。次に、/aws/lambda/VPCE-Tutorial-Lambda-2/ ロググループに対して前のステップを繰り返します。

お疲れ様でした。Amazon SNS のエンドポイントを VPC に追加することにより、VPC で管理されるネットワーク内から、トピックにメッセージを発行することができました。メッセージは、パブリックインターネットに公開されることなくプライベートで発行されました。

ステップ 7: クリーンアップする

作成したリソースは、保持することを希望しない限り、今すぐ削除できます。使用しなくなった AWS リソースを削除することで、への不要な課金を防ぐことができます AWS アカウント。

まず、Amazon VPC コンソールを使用して VPC エンドポイントを削除します。次に、CloudFormation コンソールでスタックを削除して、作成した他のリソースを削除します。スタックを削除すると、CloudFormation はスタックのリソースを から削除します AWS アカウント。

VPC エンドポイントを削除するには

1. Amazon VPC コンソールの <https://console.aws.amazon.com/vpc/> を開いてください。
2. 左側のナビゲーションメニューで、[エンドポイント] を選択します。
3. 作成したエンドポイントを選択します。
4. [アクション] を選択してから、[エンドポイントの削除] を選択します。
5. [エンドポイントの削除] ウィンドウで、[はい、削除します] を選択します。

エンドポイントのステータスが [削除中] に変わります。削除が完了すると、エンドポイントがページから削除されます。

CloudFormation スタックを削除するには

1. <https://console.aws.amazon.com/cloudformation> で CloudFormation コンソールを開きます。
2. [VPCE-Tutorial-Stack] スタックを選択します。
3. [アクション] を選択してから、[スタックの削除] を選択します。
4. [スタックの削除] ウィンドウで、[はい、削除します] を選択します。

スタックのステータスが DELETE_IN_PROGRESS に変わります。削除が完了すると、スタックがページから削除されます。

関連リソース

詳細については、以下のリソースを参照してください。

- [AWS セキュリティブログ: PrivateLink を使用して AWS Amazon SNS に発行されたメッセージを保護する](#)
- [Amazon VPC とは?](#)
- [VPC エンドポイント](#)
- [Amazon EC2 とは?](#)
- [CloudFormation の概念](#)

デュアルスタック (IPv4 および IPv6) エンドポイントを使用して Amazon SNS に接続する

デュアルスタックエンドポイントは、IPv4 と IPv6 トラフィックの両方をサポートします。デュアルスタックの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 IPv4 または IPv6 アドレスに解決されます。デュアルスタックと FIPS エンドポイント詳細については、「[SDK リファレンスガイド](#)」を参照してください。

Amazon SNS はリージョンのデュアルスタックエンドポイントをサポートしています。つまり、エンドポイント名の一部として AWS リージョンを指定する必要があります。デュアルスタックエンドポイント名には、次の命名規則が使用されます。sns.*Region*.amazonaws.com例えば、eu-west-1 リージョンのデュアルスタックエンドポイント名は、sns.*eu-west-1*.amazonaws.comです。

Amazon SNS エンドポイントの完全なリストについては、「[AWS 全般のリファレンス](#)」を参照してください。

メッセージデータ保護による Amazon SNS セキュリティの強化

- [メッセージデータ保護](#)は Amazon SNS の機能で、保存中のデータではなく、移動中のデータのコンテンツを監査および管理するための独自のルールとポリシーを定義するために使用されます。
- メッセージデータ保護は、メッセージ中心のエンタープライズアプリケーションにガバナンス、コンプライアンス、監査サービスを提供するため、Amazon SNS トピックの所有者は、データイングレスおよびエグレスを制御し、コンテンツフローを追跡して記録できます。
- ペイロードベースのガバナンスルールを記述して、不正なペイロードコンテンツがメッセージストリームに入るのを防ぐことができます。
- 個々のサブスクライバーに異なるコンテンツアクセス権限を付与し、コンテンツフロープロセス全体を監査できます。

Amazon SNS での Identity and Access Management

Amazon SNS にアクセスするには AWS、ガリクエストの認証に使用できる認証情報が必要です。これらの認証情報には、Amazon SNS トピックやメッセージなどの AWS リソースにアクセスするためのアクセス許可が必要です。次のセクションでは、[AWS Identity and Access Management \(IAM\)](#) と Amazon SNS を使用して、リソースにアクセスできるユーザーを制御することで、リソースをセキュリティで保護する方法について詳しく説明します。

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 (サインインを許可) し、誰に Amazon SNS リソースの使用を承認する (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。

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 (「[Amazon Simple Notification Service アイデンティティ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 (「[Amazon SNS と IAM が連携する仕組み](#)」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 (「[Amazon Simple Notification Service のアイデンティティベースのポリシーの例](#)」を参照)

アイデンティティを使用した認証

認証は、ID 認証情報 AWS を使用して にサインインする方法です。AWS アカウント ルートユーザー、IAM 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 アイデンティティセンター (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、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 ルート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。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーテッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID Directory Service ソースの認証情報 AWS のサービス を使用して にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「[IAM アイデンティティセンターとは](#)」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ている場合の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。

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。

アクセスコントロール

Amazon SNS には、AWS Identity and Access Management (IAM) ポリシーと同じ言語で記述されたポリシーを使用する独自のリソースベースのアクセス許可システムがあります。これは、Amazon SNS ポリシーや IAM ポリシーで同様の結果を達成できることを意味します。

Note

すべての が自分のアカウントのユーザーに権限を委任 AWS アカウント できることを理解することが重要です。クロスアカウントアクセスによって、追加のユーザーを管理しなくても、AWS リソースへのアクセスを共有することができます。クロスアカウントのアクセスの詳細については、IAM ユーザーガイドの「[クロスアカウントアクセスの有効化](#)」を参照してください。

Amazon SNS アクセスコントロールのユースケース

リソースへのアクセス権限やアクセス拒否の方法に高い柔軟性があります。ただし、一般的なユースケースは、次に示すように非常にシンプルなものとなっています。

- 特定のタイプのトピックアクション (パブリッシュなど) AWS アカウント を別の に付与する場合。詳細については、「[トピック AWS アカウント へのアクセスを許可する](#)」を参照してください。
- HTTPS プロトコルに対してのみ、トピックのサブスクリプションを制限します。詳細については、「[HTTPS へのサブスクリプションの制限](#)」を参照してください。
- Amazon SNS が Amazon SQS キューにメッセージを発行することを許可する必要があります。詳細については、「[Amazon SQS キューに発行する。](#)」を参照してください。

Amazon SNS アクセスポリシーの主な概念

以下のセクションでは、アクセスポリシー言語を使用するにあたって理解しておくべきコンセプトを説明します。基本的なものから順に分かりやすく説明していきます。

アクセス許可

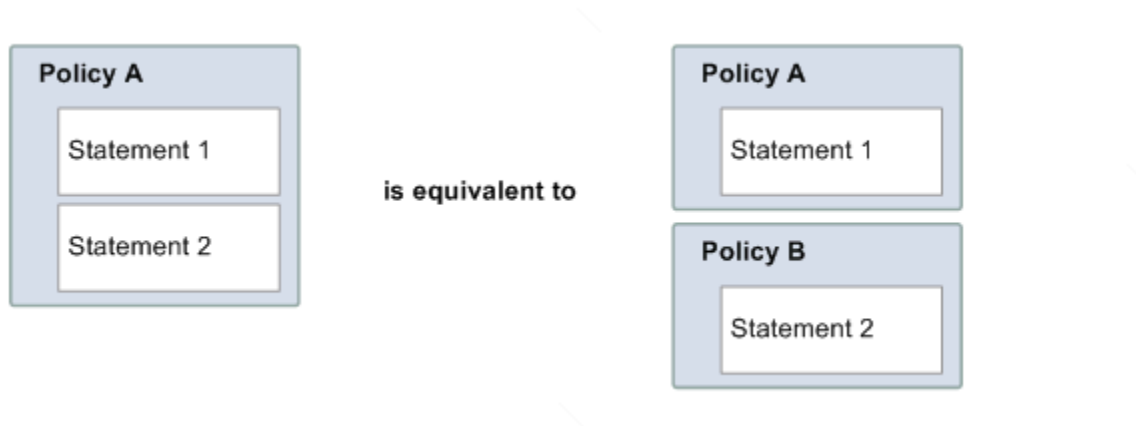
アクセス権限とは、特定のリソースへのある種のアクセスに対し、許可または拒否をするというコンセプトです。アクセス権限は、基本的に「A は、条件 D に該当する C を対象とするアクション B の実行を許可または禁止されている」という形態をとります。例えば、ジェーン (A) は HTTP プロトコルを使用する場合に限り (D)、トピック A (C) へ発行すること (B) が許可されています。ジェーンがトピック A をパブリッシュした時点で、ジェーンにアクセス権限があるかどうか、またそのリクエストが定められているアクセス権限の条件を満たしているかどうか、サービスによってチェックされます。

ステートメント

ステートメントとは、アクセスポリシー言語で使用するアクセス権限を定義する書式です。1 つのステートメントで 1 つのアクセス権限を定義します。ポリシー という、より広範囲のコンテナドキュメントの一部としてステートメントを書きます (次のコンセプト参照)。

ポリシー

ポリシーとは、1 つ以上のステートメントのコンテナの役目を果たすドキュメント (アクセスポリシー言語で記述) です。例えば、1 つのポリシーには「ジェーンは E メールプロトコルを使用してサブスクライブできる」というステートメントと、「ボブはトピック A を発行することができない」というステートメント、計 2 つのステートメントが含まれることがあります。以下の図に示されているように、同等のシナリオでは、「ジェーンは E メールプロトコルを使用してサブスクライブできる」というポリシーと、「ボブはトピック A を発行することができない」というポリシー、計 2 つのポリシーを含むことができます。



ポリシードキュメントでは ASCII 文字のみを使用できます。aws:SourceAccount と を使用してaws:SourceOwner、ASCII 以外の文字を含む他の AWS サービスの ARNs をプラグインする必要があるシナリオを回避できます。[aws:SourceAccount と aws:SourceOwner](#) の違いを確認してください。

Issuer

発行者とは、リソース用のアクセス権限についてのポリシーを記述する個人です。発行者 (定義上) は常にリソース所有者です。AWS は AWS 、サービスユーザーが所有していないリソースのポリシーを作成することを許可しません。John がリソース所有者である場合、 は、そのリソースに対するアクセス許可を付与するために記述されたポリシーを送信するときに、John の ID AWS を認証します。

Principal

プリンシパルとは、ポリシーのアクセス権限を適用される個人またはグループを指します。「A は、条件 D に該当する C を対象とするアクション B の実行を許可または禁止されている」というステートメントにおいては、A がプリンシパルに相当します。ポリシーでは、「誰でも」プリンシパルに設定することができます (例えば、ワイルドカードを指定して全員に設定できます)。例えば、リクエストの実際のアイデンティティをベースにアクセス制限をかけたくなければ、リクエストの IP アドレスのような、他のアイデンティティの特性をもとにして行うことができます。

Action

アクションとは、プリンシパルに対し、実行が許可されているアクティビティです。「A は、条件 D に該当する場合 C に対して B を実行することを許可されている」というステートメントにおいては、B がアクションに相当します。通常、 アクションはリクエストの オペレーションにすぎませ

ん AWS。例えば、ジェーンが Amazon SNS に Action =Subscribe というリクエストを送信します。1 つのポリシーに 1 つまたは複数のアクションを指定することができます。

[リソース]

リソースとは、プリンシパルがアクセスを要求するオブジェクトのことです。「A は、条件 D に該当する場合 C に対して B を実行することを許可されている」というステートメントにおいては、C がリソースに相当します。

条件とキー

条件とは、アクセス権限についての制限や詳細のことです。「A は、条件 D に該当する場合 C に対して B を実行することを許可されている」というステートメントにおいては、D が条件に相当します。ポリシーの中でも、記述が最も詳細かつ複雑になるのが、この条件部分です。よく使用される条件の設定項目は以下のとおりです。

- 日時 (特定の日付以前に到着したリクエストのみ処理するなど)
- IP アドレス (特定の CIDR 範囲内の IP アドレスからのリクエストのみ処理するなど)

キーは、アクセス制限に使用される基本項目です。例えば、リクエストの日時がこれに相当します。

制限は、条件とキーの両方を使用して定義します。具体例を挙げて説明します。2010 年 5 月 30 日以前のアクセスを制限するには、DateLessThan という条件を使用します。aws:CurrentTime と呼ばれるキーを使用して それを 2010-05-30T00:00:00Z の値に設定します。AWS では使用できる条件やキーを定義します。AWS サービス自体 (Amazon SQS や Amazon SNS など) も、サービス固有のキーを定義する場合があります。詳細については、「[Amazon SNS API のアクセス許可: アクションとリソースのリファレンス](#)」を参照してください。

リクエスト

リクエストは、AWS サービスにリクエストを送信し、特定のリソースへのアクセスを要求するユーザーです。リクエストは、「D が適用される C への B の実行を許可しますか? AWS」というリクエストを送信します。

評価

評価は、該当するポリシーに基づいて受信リクエストを拒否するか許可するかを判断するために AWS サービスが使用するプロセスです。評価論理の詳細については、「[評価論理](#)」を参照してください。

エフェクト

エフェクトとは、ポリシーのステートメントが評価時に返す結果のことです。この値はポリシーのステートメントを記述するときに指定します。使用可能な値は `deny` と `allow` です。

例えば、南極大陸からのすべてのリクエストを拒否するステートメントを含むポリシーを記述できます (リクエストの送信元 IP アドレスが南極大陸に割り当てられている場合、エフェクトの値を `deny` とします)。または、南極大陸以外からのすべてのリクエストを許可するステートメントを含むポリシーを記述できます (リクエストの送信元が南極大陸でなければ、エフェクトの値は `allow` となります)。2 つのステートメントは同じことを行うように見えますが、アクセスポリシー言語の論理上では異なるものです。詳細については、「[評価論理](#)」を参照してください。

エフェクトに特定できる値は `allow` と `deny` の 2 つだけですが、ポリシーの評価結果には、デフォルトで拒否、許可および明示的拒否の 3 種類があります。詳細については、以下のコンセプトおよび「[評価論理](#)」を参照してください。

デフォルトで拒否

デフォルトで拒否とは、ポリシーに許可または明示的拒否が指定されていない場合に、デフォルトで適用される拒否のことです。

許可

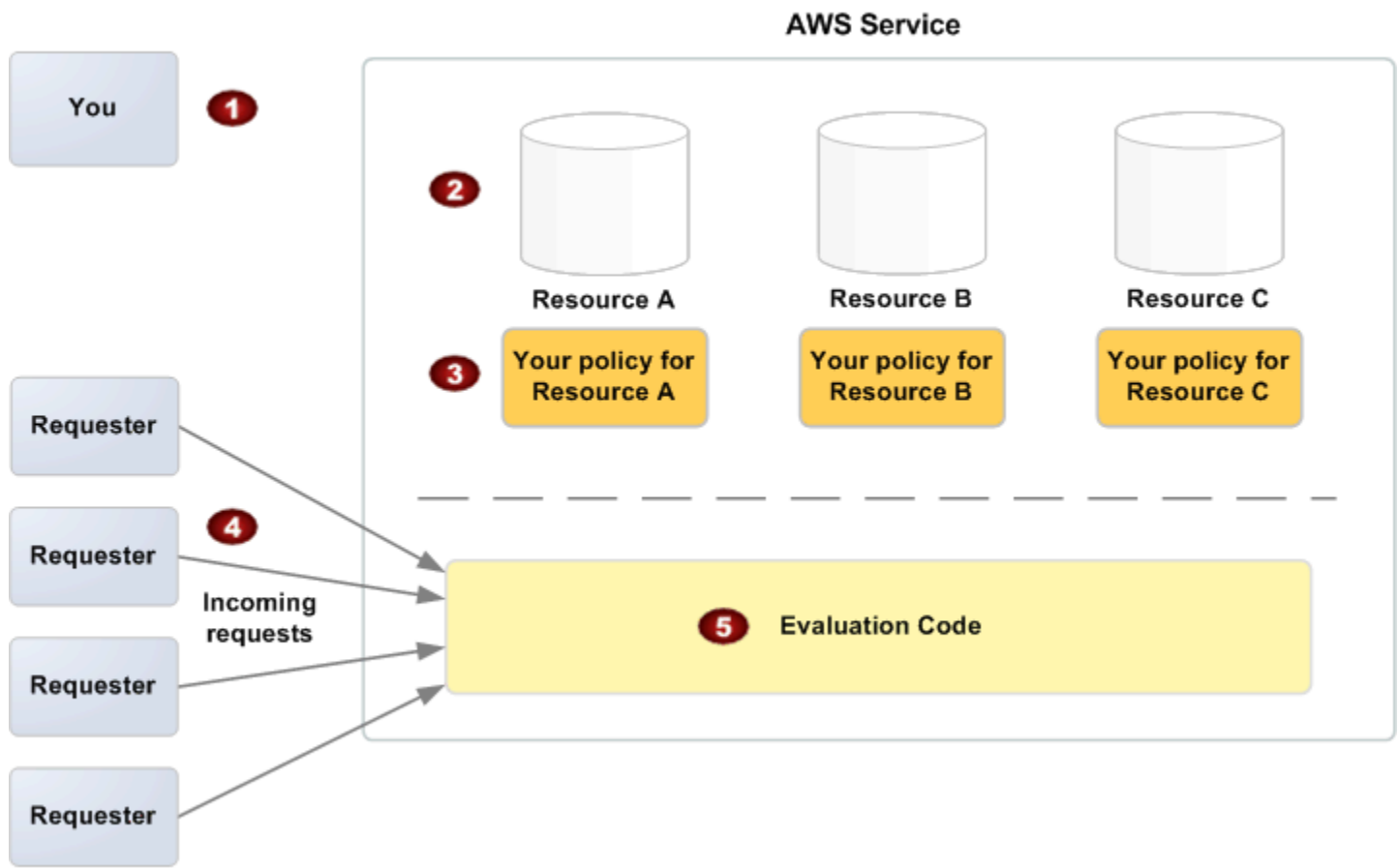
許可とは、ステートメントに `effect=allow` が指定されていて、許可条件がすべて満たされている場合に返される結果です。例えば、2010 年 4 月 30 日午後 1 時まで受信されたリクエストが許可されます。許可は、すべてのデフォルトで拒否に優先して適用されますが、明示的拒否が 1 つでもあれば適用されません。

明示的拒否

明示的拒否とは、ステートメントに `effect=deny` が指定されていて、拒否条件がすべて満たされている場合に返される結果です。例えば、送信元が南極大陸であるすべてのリクエストが拒否されます。その他のポリシーによって許可されている場合においても、南極から来たリクエストに対しては常に拒否します。

Amazon SNS のアクセスコントロールアーキテクチャの概要

以下の図と表に、リソースのアクセスコントロールに関与する主要コンポーネントとそのインタラクティブな関わり合いを表します。



1 お客様 = リソース所有者。

2 リソース (Amazon SQS キューなど、AWS サービス内に含まれています)。

3 ポリシー。

通常、1つのリソースに1つのポリシーを適用しますが、複数のポリシーを適用することも可能です。AWS サービス自体には、ポリシーのアップロードと管理に使用する API が用意されています。

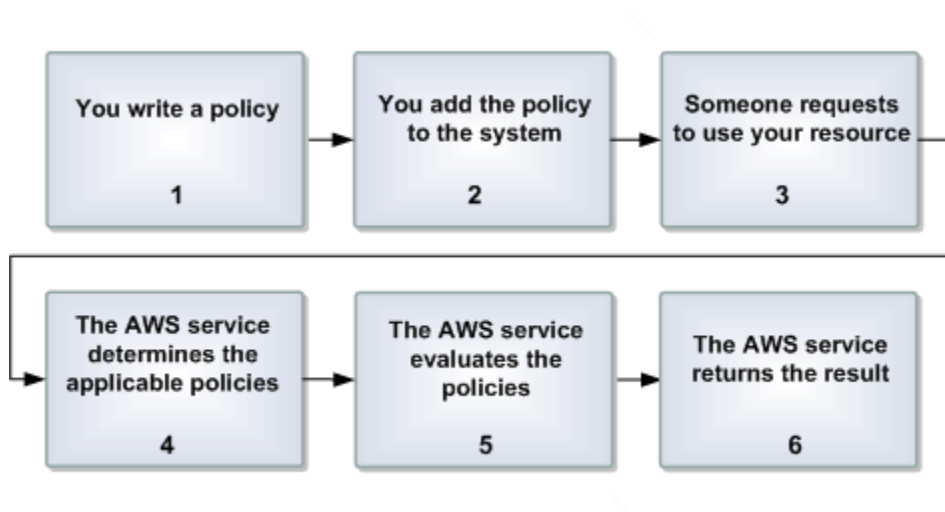
4 リクエスタとその AWS サービスへの受信リクエスト。

5 アクセスポリシー言語の評価コード。

これは、受信リクエストを該当するポリシーと照らし合わせて評価し、リクエスタがリソースへのアクセスを許可するかどうかを決定する AWS サービス内のコードのセットです。このサービスによる判定の詳細については、「[評価論理](#)」を参照してください。

Amazon SNS でアクセスポリシー言語を使用する

以下の図と表は、アクセスコントロールとアクセスポリシー言語の連携方法の通常プロセスを表しています。



アクセスコントロールをアクセスポリシー言語と使用するプロセス

- 1 リソース用ポリシーを記述します。

例えば、Amazon SNS トピックのアクセス権限を指定するポリシーを記述します。

- 2 ポリシーを にアップロードします AWS。

AWS サービス自体には、ポリシーのアップロードに使用する API が用意されています。例えば、特定の Amazon SNS トピック用のポリシーをアップロードするために Amazon SNS SetTopicAttributes アクションを使用します。

- 3 ある人物から、リソースの使用許可を求めるリクエストが送信されます。

例えば、ユーザーがトピックの 1 つを利用するために Amazon SNS へリクエストを送信します。

- 4 AWS サービスは、リクエストに適用できるポリシーを決定します。

例えば、Amazon SNS がすべての利用可能な Amazon SNS ポリシーを調べ、リソースの内容とリクエストに基づいて、どのポリシーが適用可能であるかを決定します。

- 5 AWS サービスはポリシーを評価します。

例えば、Amazon SNS は、ポリシーを評価し、トピックの使用許可をリクエストに付与するかどうかを決定します。決定論理の詳細については、「[評価論理](#)」を参照してください。

6 AWS サービスはリクエストを拒否するか、処理を続行します。

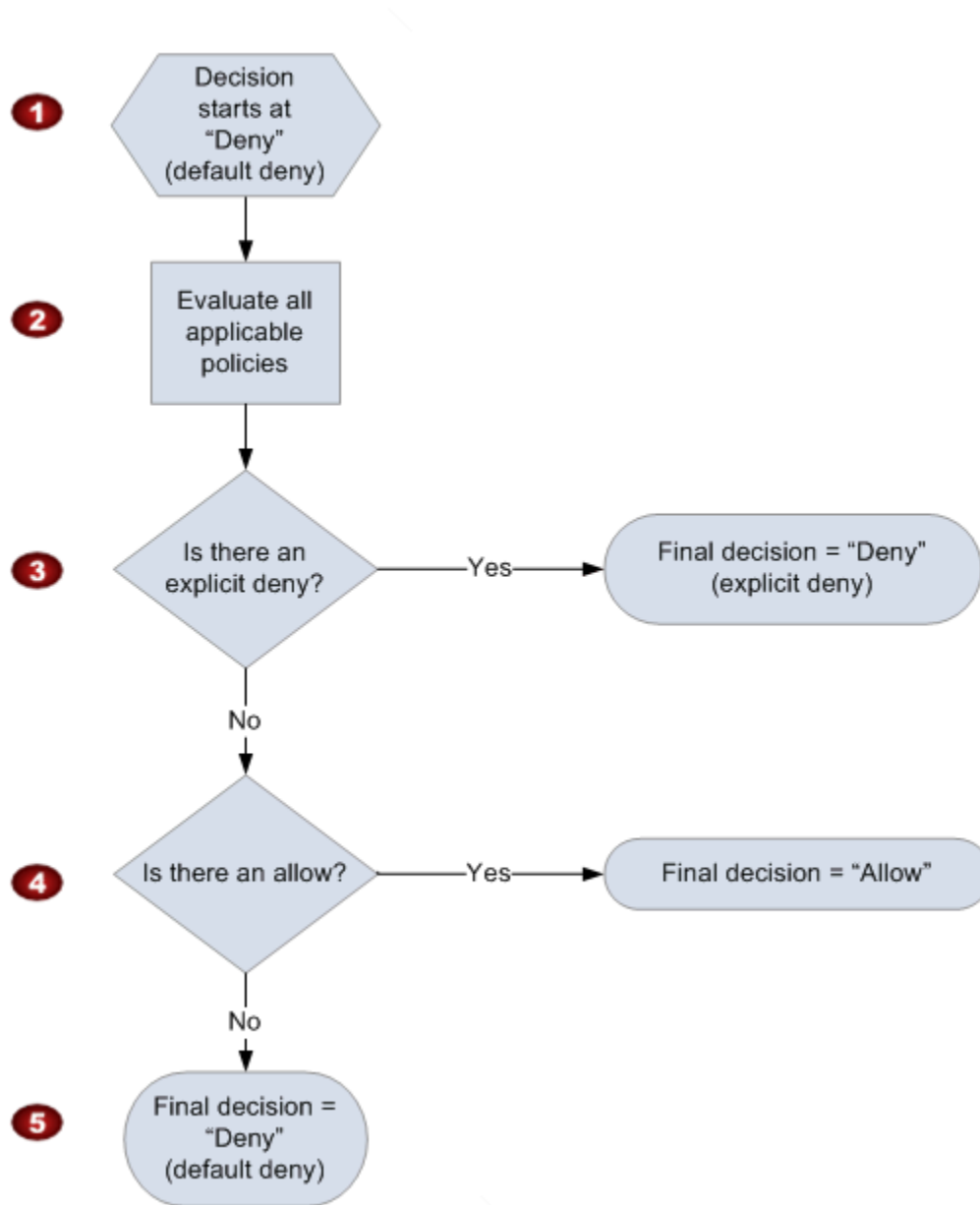
例えば、ポリシーの評価結果に基づいて、サービスによって「アクセス拒否」エラーがリクエストに返されるか、リクエストのプロセスを継続するかのどちらかが行われます。

評価論理

評価時の目標は、特定のリクエスト付与を許可するか拒否するかを判断することです。評価論理は、以下の複数の基本ルールに従っています。

- デフォルトでは、リソースの使用許可を求めるリクエストについては、リクエストが自分自身である場合を除いて、拒否を適用する
- 許可はすべてのデフォルトで拒否に優先する
- 明示的拒否はすべての許可に優先する
- ポリシー評価の順序は重要ではない

以下のフローチャートと考察では、決定方法についての詳細説明を紹介します。



- 1 決定はデフォルトで拒否から始まります。
- 2 次に、エンフォースメントコードは、リクエストに適用可能なポリシーすべてを、リソース、プリンシパル、アクション、および条件に基づいて評価します。
エンフォースメントコードによるポリシー評価の順序は重要ではありません。
- 3 前述のすべてのポリシーにおいて、リクエストに適應する明示的拒否のインストラクションがエンフォースメントコードによって検索されます。

仮に 1 つでも見つかった場合、エンフォースメントコードは「拒否」の決定を返し、プロセスを終了します (これは明示的拒否となります。詳細については、「[明示的拒否](#)」を参照してください)。

4 明示的拒否が見つからなかった場合、リクエストに適応する「許可」のインストラクションがエンフォースメントコードによって検索されます。

仮に 1 つでも見つかった場合、エンフォースメントコードは「許可」の決定を返し、プロセスは完了します (サービスはリクエストのプロセスを継続します)。

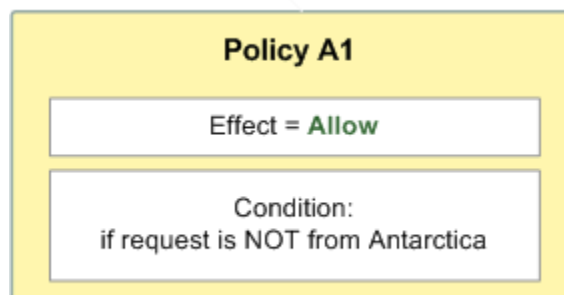
5 許可が見つからなかった場合、最終決定は「拒否」となります (明示的拒否または許可が見つからない場合、デフォルトで拒否として見なされるためです (詳細については、「[デフォルトで拒否](#)」を参照してください)。

明示的拒否とデフォルトで拒否の相互作用

ポリシーがリクエストに直接適用されない場合の結果は、デフォルトで拒否となります。たとえば、ユーザーが Amazon SNS の使用をリクエストしたが、トピックのポリシーがユーザーのポリシーを AWS アカウント まったく参照していない場合、そのポリシーはデフォルトの拒否になります。

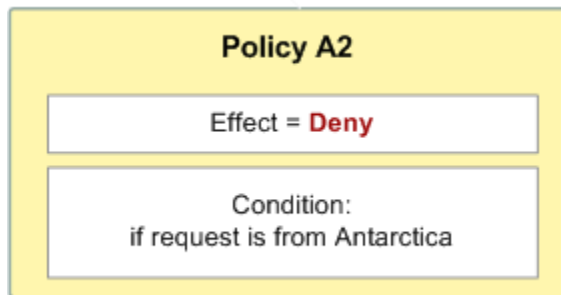
ステートメントの条件が満たされていない場合においても、ポリシーの結果としてデフォルトで拒否となります。ステートメントのすべての条件が満たされている場合、ポリシーのエフェクトエレメントの値に基づいて、ポリシーの結果は許可または明示的拒否のどちらかとなります。条件が満たされていない際にポリシーが行為を特定していない場合、デフォルトの結果としてデフォルトで拒否となります。

例えば、南極大陸から来るリクエストを防ぐとします。その場合、南極大陸から来ていないリクエストにのみ許可を与えるポリシー (ポリシー A1 とする) を記述します。以下の図はポリシーについて解説しています。



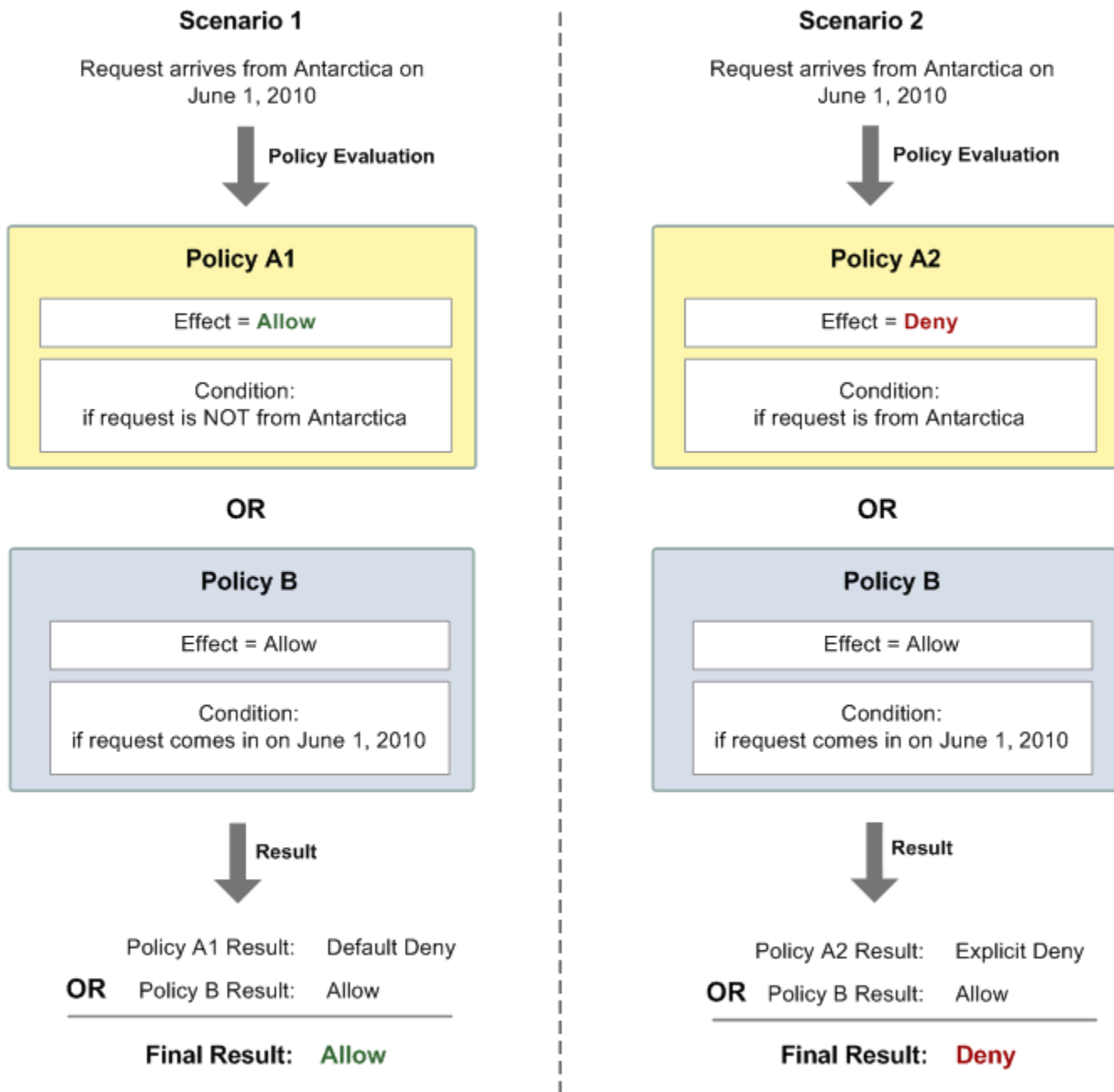
リクエストがアメリカから送られてきた場合、条件を満たしています (リクエストが南極大陸からのものでないため)。従って、そのリクエストは許可されます。しかし、リクエストが南極大陸から送られてきた場合、条件を満たしていないため、ポリシーの結果としてデフォルトで拒否となります。

以下の図のとおり、ポリシー (ポリシー A2 とする) を書き換えることにより、結果を明示的拒否に変えることができます。南極大陸から送られてきた場合、ポリシーによってリクエストが明示的に拒否されます。



リクエストが南極大陸から送られてきた場合、条件を満たしているため、ポリシーの結果として明示的な拒否となります。

デフォルトで拒否と明示的拒否の区別は重要です。許可によってデフォルトで拒否は上書きできますが、明示的拒否は上書きできないためです。例えば、リクエストが 2010 年 6 月 1 日に到着すれば許可するという別のポリシーがあるとしましょう。このポリシーが、南極大陸からのアクセスを制限しているポリシーと併用されている場合、全体の結果にどのような影響を及ぼすでしょうか? 日付ベースのポリシー (ポリシー B とする) が前述のポリシー A1 および A2 と併用されている場合、全体の結果が比較されます。シナリオ 1 は、ポリシー A1 とポリシー B が併用されている場合、シナリオ 2 は、ポリシー A2 とポリシー B が併用されている場合です。以下の図と考察は、2010 年 6 月 1 日に南極大陸からリクエストが来た場合についての結果を示しています。



このセクションの最初に説明したとおり、シナリオ 1 においては、ポリシー A1 はデフォルトで拒否を返します。2010 年 6 月 1 日に到着したリクエストは、当然のことながら許可されるため、ポリシー B は許可を返します。ポリシー B による許可は、ポリシー A1 のデフォルトで拒否に優先するため、結果としてリクエストは許可されます。

このセクションの最初に説明したとおり、シナリオ 2 においては、ポリシー A2 は明示的拒否を返します。再度、ポリシー B は許可を返します。ポリシー A2 による明示的拒否は、ポリシー B の許可に優先するため、結果としてリクエストは拒否されます。

Amazon SNS アクセスコントロールのケース例

このセクションでは、アクセスコントロールの一般的なユースケース例をいくつか紹介します。

トピック AWS アカウント へのアクセスを許可する

Amazon SNS にトピックがあり、メッセージの発行など、そのトピックに対して特定のアクションを実行 AWS アカウント することを 1 つ以上の に許可するとします。これは、Amazon SNS API アクション `AddPermission` を使用して実行できます。

`AddPermission` アクションでは、トピック、AWS アカウント ID のリスト、アクションのリスト、ラベルを指定できます。これによって、Amazon SNS は新しいポリシーステートメントを自動的に生成し、トピックのアクセスコントロールポリシーに追加します。ポリシーステートメントを自分で作成する必要はありません。Amazon SNS が自動的に作成します。後でポリシーを削除する必要がある場合は、`RemovePermission` を呼び出して、アクセス許可の追加時に使用したラベルを指定して削除できます。

たとえば、トピック `arn:aws:sns:us-east-2:444455556666:MyTopic` `AddPermission`で を呼び出す場合、AWS アカウント ID `1111-2222-3333`、`Publish`アクション、ラベル を指定すると `grant-1234-publish`、Amazon SNS は次のポリシーステートメントを生成し、トピックのアクセスコントロールポリシーに挿入します。

```
{
  "Statement": [{
    "Sid": "grant-1234-publish",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": ["sns:Publish"],
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic"
  }]
}
```

このステートメントが追加されると、AWS アカウント `1111-2222-3333` にはトピックにメッセージを発行するアクセス許可が付与されます。

追加情報:

- カスタムポリシー管理: `AddPermission` はアクセス許可の付与には便利ですが、条件の追加や特定の IAM ロールまたはサービスへのアクセス許可の付与など、より複雑なシナリオでは、多く

の場合、トピックのアクセスコントロールポリシーを手動で管理すると便利です。これを行うには、SetTopicAttributes API を使用して、ポリシー属性を直接更新します。

- セキュリティのベストプラクティス: アクセス許可を付与する場合は、信頼できる AWS アカウント またはエンティティのみが Amazon SNS トピックにアクセスできるように注意してください。セキュリティを維持するには、トピックにアタッチされているポリシーを定期的に確認および監査します。
- ポリシーの制限: Amazon SNS ポリシーのサイズと複雑さには制限があることに注意してください。多くのアクセス許可や複雑な条件を追加する必要がある場合は、ポリシーがこれらの制限内に収まるようにしてください。

HTTPS へのサブスクリプションの制限

Amazon SNS トピックの通知配信プロトコルを HTTPS に制限するには、カスタムポリシーを作成する必要があります。Amazon SNS の AddPermission アクションでは、トピックへのアクセスを許可するときにプロトコル制限を指定することはできません。したがって、この制限を適用するポリシーを手動で作成し、SetTopicAttributes アクションを使用してポリシーをトピックに適用する必要があります。

サブスクリプションを HTTPS に制限するポリシーを作成する方法は次のとおりです。

1. ポリシーを作成します。ポリシーは、アクセスを許可する AWS アカウント ID を指定し、HTTPS サブスクリプションのみを許可する条件を適用する必要があります。以下は、使用されるプロトコルが HTTPS である場合にのみ、トピックをサブスクライブするアクセス許可を AWS アカウント ID 1111-2222-3333 に付与するポリシーの例です。

```
{
  "Statement": [{
    "Sid": "Statement1",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": ["sns:Subscribe"],
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
    "Condition": {
      "StringEquals": {
        "sns:Protocol": "https"
      }
    }
  }]
}
```



```
    }  
  }  
}
```

2. ポリシーを適用します。Amazon SNS API の `SetTopicAttributes` アクションを使用して、このポリシーをトピックに適用します。トピックの `Policy` 属性を、作成した JSON ポリシーに設定します。

```
snsClient.setTopicAttributes(SetTopicAttributesRequest.builder()  
    .topicArn("arn:aws:sns:us-east-2:444455556666:MyTopic")  
    .attributeName("Policy")  
    .attributeValue(jsonPolicyString) // The JSON policy as a string  
    .build());
```

追加情報:

- アクセスコントロールのカスタマイズ。このアプローチにより、サブスクリプションプロトコルの制限など、`AddPermission` アクションだけでは実行できない、より詳細なアクセスコントロールを適用できます。カスタムポリシーは、プロトコルの強制や IP アドレスの制限など、特定の条件を必要とするシナリオに柔軟性を提供します。
- セキュリティのベストプラクティス。サブスクリプションを HTTPS に制限すると、転送中のデータが暗号化され、通知のセキュリティが向上します。トピックポリシーを定期的に確認し、セキュリティとコンプライアンスの要件を満たしていることを確認します。
- ポリシーのテスト。ポリシーを本番環境に適用する前に、開発環境でテストし、想定どおりに動作することを確認します。これにより、偶発的なアクセスの問題や意図しない制限を防ぐことができます。

Amazon SQS キューに発行する。

Amazon SNS トピックから Amazon SQS キューにメッセージを発行するには、Amazon SQS キューに適切なアクセス許可を設定する必要があります。Amazon SNS と Amazon SQS はどちらも AWS アクセスコントロールポリシー言語を使用しますが、Amazon SQS Amazon SNS キューにポリシーを明示的に設定する必要があります。

これを実現するには、`SetQueueAttributes` アクションを使用して Amazon SQS キューにカスタムポリシーを適用します。Amazon SNS とは異なり、Amazon SQS は `AddPermission` アクションによる条件付きポリシーステートメントの作成をサポートしていません。したがって、ポリシーを手動で作成する必要があります。

以下は、キューにメッセージを送信するアクセス許可を Amazon SNS に付与する Amazon SQS ポリシーの例です。このポリシーは、Amazon SNS トピックではなく、Amazon SQS キューに関連付けられていることに注意してください。指定されたアクションは Amazon SQS アクションであり、リソースはキューの Amazon リソースネーム (ARN) です。GetQueueAttributes アクションを使用して、キューの ARN を取得できます。

```
{
  "Statement": [{
    "Sid": "Allow-SNS-SendMessage",
    "Effect": "Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action": ["sqs:SendMessage"],
    "Resource": "arn:aws:sqs:us-east-2:444455556666:MyQueue",
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:sns:us-east-2:444455556666:MyTopic"
      }
    }
  }]
}
```

このポリシーは、送信されるメッセージのソースに基づいて、SQS キューへのアクセスを制限する `aws:SourceArn` 条件を使用します。これにより、指定された SNS トピック (この場合は、`arn:aws:sns:us-east-2:444455556666:MyTopic`) から発信されたメッセージのみがキューに配信されます。

追加情報:

- キュー ARN。GetQueueAttributes アクションを使用して、Amazon SQS キューの正しい ARN を取得していることを確認します。この ARN は、正しいアクセス許可を設定するために不可欠です。
- セキュリティのベストプラクティス。ポリシーを設定するときは、常に最小特権の原則に従います。Amazon SQS キューとやり取りするために必要な権限のみを Amazon SNS トピックに付与し、ポリシーを定期的にレビューして、ポリシーが最新かつ安全であることを確認します。
- Amazon SNS のデフォルトポリシー。Amazon SNS は、他の AWS のサービス または アカウントが新しく作成されたトピックにアクセスすることを許可するデフォルトのポリシーを自動的に付与しません。デフォルトでは、Amazon SNS トピックはアクセス許可なしで作成されます。つまり、それらのトピックはプライベートであり、トピックを作成したアカウントのみがアクセスで

きます。他の、アカウント AWS のサービス、またはプリンシパルのアクセスを有効にするには、トピックにアクセスポリシーを明示的に定義してアタッチする必要があります。これは最小特権の原則に従い、意図しないアクセスがデフォルトで付与されないようにするための措置です。

- テストと検証。ポリシーを設定したら、Amazon SNS トピックにメッセージを発行し、それらが Amazon SQS キューに正常に配信されることを確認して、統合をテストします。これにより、ポリシーが正しく設定されていることを確認できます。

Amazon S3 イベント通知がトピックに発行することを許可する

別の Amazon S3 バケット AWS アカウント が Amazon SNS トピックにイベント通知を発行できるようにするには、それに応じてトピックのアクセスポリシーを設定する必要があります。そのためには、特定の AWS アカウント から Amazon S3 サービスにアクセス許可を付与するカスタムポリシーを作成し、そのポリシーを Amazon SNS トピックに適用します。

設定の方法は次のとおりです。

1. ポリシーを作成します。ポリシーは、Amazon SNS トピックに発行するために必要なアクセス許可を Amazon S3 サービス (s3.amazonaws.com) に付与します。SourceAccount 条件を使用して AWS アカウント、Amazon S3 バケットを所有する指定された のみがトピックに通知を発行できるようにします。

次に、ポリシーの例を示します。

```
{
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "s3.amazonaws.com"
    },
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-2:111122223333:MyTopic",
    "Condition": {
      "StringEquals": {
        "AWS:SourceAccount": "444455556666"
      }
    }
  ]
}
```

- トピック所有者 – 111122223333 は、Amazon SNS トピックを所有する AWS アカウント ID です。
 - Amazon S3 バケット所有者 – 444455556666 は、Amazon S3 バケット送信通知を所有する AWS アカウント ID です。Amazon S3
2. ポリシーを適用します。SetTopicAttributes アクションを使用して、Amazon SNS トピックにこのポリシーを設定します。これにより、トピックのアクセスコントロールが更新され、カスタムポリシーで指定されたアクセス許可が含まれます。

```
snsClient.setTopicAttributes(SetTopicAttributesRequest.builder()  
    .topicArn("arn:aws:sns:us-east-2:111122223333:MyTopic")  
    .attributeName("Policy")  
    .attributeValue(jsonPolicyString) // The JSON policy as a string  
    .build());
```

追加情報:

- **SourceAccount** 条件の使用。このSourceAccount条件により、この場合は指定された AWS アカウント (444455556666) から発生したイベントのみが Amazon SNS トピックをトリガーできるようになります。これは、不正なアカウントがトピックに通知を送信できないようにするセキュリティ対策です。
- **SourceAccount** をサポートするその他のサービス。SourceAccount 条件は、次のサービスでサポートされています。発信元アカウントに基づいて Amazon SNS トピックへのアクセスを制限する場合は、この条件を使用することが重要です。
 - Amazon API Gateway
 - Amazon CloudWatch
 - Amazon DevOps Guru
 - Amazon EventBridge
 - Amazon GameLift Servers
 - Amazon Pinpoint SMS および音声 API
 - Amazon RDS
 - Amazon Redshift
 - Amazon Glacier
 - Amazon SES

- Amazon Simple Storage Service
- AWS CodeCommit
- Directory Service
- AWS Lambda
- AWS Systems Manager Incident Manager
- テストと検証。ポリシーを適用したら、Amazon S3 バケットでイベントをトリガーしてセットアップをテストし、Amazon SNS トピックに正常に発行されることを確認します。これにより、ポリシーが正しく設定されていることを確認します。
- セキュリティのベストプラクティス。Amazon SNS トピックポリシーを定期的に確認して監査し、セキュリティ要件に準拠していることを確認します。安全なオペレーションを維持するには、信頼できるアカウントとサービスのみにアクセスを制限することが不可欠です。

Amazon SES が別のアカウントが所有するトピックへの発行を許可する

別の が所有するトピック AWS のサービス への発行を別の に許可できます AWS アカウント。111122223333 アカウントにサインインし、Amazon SES を開き、E メールを作成したとします。この E メールに関する通知を、444455556666 アカウントが所有する Amazon SNS トピックに発行するには、以下のようなポリシーを作成します。そのためには、プリンシパル (他のサービス) と各リソースの所有権に関する情報を提供する必要があります。Resource ステートメントは、トピック所有者のアカウント ID 444455556666 を含むトピック ARN を提供します。"aws:SourceOwner": "111122223333" ステートメントは、アカウントがその E メールを所有することを指定します。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "ses.amazonaws.com"
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
```

```
    "Condition": {
      "StringEquals": {
        "aws:SourceOwner": "111122223333"
      }
    }
  ]
}
```

Amazon SNS にイベントを発行する場合、以下のサービスで `aws:SourceOwner` を使用できます。

- Amazon API Gateway
- Amazon CloudWatch
- Amazon DevOps Guru
- Amazon GameLift Servers
- Amazon Pinpoint SMS および音声 API
- Amazon RDS
- Amazon Redshift
- Amazon SES
- AWS CodeCommit
- Directory Service
- AWS Lambda
- AWS Systems Manager Incident Manager

`aws:SourceAccount` と `aws:SourceOwner`

Important

`aws:SourceOwner` は廃止されました。新しいサービスを Amazon SNS と統合できるのは、`aws:SourceArn` と `aws:SourceAccount` を使用した場合のみです。Amazon SNS は、現在 `aws:SourceOwner` をサポートしている既存のサービスとの下位互換性を引き続き維持しています。

aws:SourceAccount 条件キーと aws:SourceOwner 条件キーは、一部の AWS のサービスが Amazon SNS トピックに発行するときに、それぞれ設定されます。サポートされている場合、値は、サービスがデータを発行する の 12 桁の AWS アカウント ID になります。サービスによってサポートするものは変わります。

- Amazon S3 通知の使用[方法](#)aws:SourceAccountと、その条件をサポートする AWS サービスのリスト[Amazon S3 イベント通知がトピックに発行することを許可する](#)については、「」を参照してください。
- Amazon SES が [使用する方法](#)aws:SourceOwnerと、その条件をサポートする AWS サービスのリスト[Amazon SES が別のアカウントが所有するトピックへの発行を許可する](#)については、「」を参照してください。

の組織内のアカウントが別のアカウントのトピックに発行 AWS Organizations することを許可する

この AWS Organizations サービスは、請求の一元管理、アクセスとセキュリティの制御、間でのリソースの共有に役立ちます AWS アカウント。

組織 ID は [Organizations コンソール](#)で確認できます。詳細については、「[管理アカウントからの組織の詳細の表示](#)」を参照してください。

この例では、組織 AWS アカウント 内のすべての がアカウント MyTopicの Amazon SNS トピックに発行myOrgIdできます444455556666。ポリシーでは、aws:PrincipalOrgID グローバル条件キーを使用して組織 ID 値を確認します。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
      "Condition": {
        "StringEquals": {
          "aws:PrincipalOrgID": "myOrgId"
        }
      }
    }
  ]
}
```

```
]
}
```

CloudWatch アラームに別のアカウントのトピックへの発行を許可する

異なる AWS アカウント間で CloudWatch アラームを使用して Amazon SNS トピックを呼び出すには、次の手順を実行します。この例では、次の 2 つのアカウントを使用します。

- アカウント A は CloudWatch アラームの作成に使用されます。
- アカウント B は SNS トピックの作成に使用されます。

アカウント B で SNS トピックを作成する

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションペインで、[Topics (トピック)]、[Create topic (トピックの作成)] の順に選択してください。
3. トピックタイプの [標準] を選択し、トピックの名前を作成します。
4. [トピックの作成] を選択し、トピックの ARN をコピーします。
5. ナビゲーションペインで、[Subscriptions] (サブスクリプション) を選択して、[Create subscription] (サブスクリプションの作成) を選択します。
6. [トピック ARN] セクションにトピックの ARN を追加し、プロトコルに [E メール] を選択して、E メールアドレスを入力します。
7. [サブスクリプションの作成] を選択し、E メールを調べて、サブスクリプションを確認します。

アカウント A で CloudWatch アラームを作成する

1. CloudWatch コンソールの <https://console.aws.amazon.com/cloudwatch/> を開いてください。
2. ナビゲーションペインで、[アラーム] を選択し、[アラームを作成] を選択します。
3. アラームを作成していない場合は、ここで作成します。アラームが作成されている場合は、メトリクスを選択し、しきい値と比較パラメータの詳細を指定します。
4. [アクションの設定] の [通知] で [トピック ARN を使用して他のアカウントに通知] を選択し、アカウント B のトピック ARN を入力します。
5. アラームの名前を作成し、[アラームを作成] を選択します。

アカウント B の SNS トピックのアクセスポリシーを更新する

1. [Amazon SNS コンソール](#)にサインインします。
2. ナビゲーションペインで、[トピック] を選択し、トピックを選びます。
3. [編集] を選択し、ポリシーに以下のコードを追加します。

 Note

以下のポリシーの例に示されている値を、独自の値に置き換えてください。

JSON

```
{
  "Version": "2012-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:GetTopicAttributes",
        "SNS:SetTopicAttributes",
        "SNS:AddPermission",
        "SNS:RemovePermission",
        "SNS:DeleteTopic",
        "SNS:Subscribe",
        "SNS:ListSubscriptionsByTopic",
        "SNS:Publish"
      ],
      "Resource": "arn:aws:cloudwatch:us-west-1:111122223333:alarm:",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:cloudwatch:us-west-1:111122223333:alarm:"
        }
      }
    }
  ]
}
```

```
}
```

アラームをテストする

アラームをテストするには、メトリクスデータポイントに基づいてアラームのしきい値を変更するか、アラームの状態を手動で変更します。アラームのしきい値またはアラームの状態を変更すると、E メール通知が届きます。

ローカルの Amazon SNS トピックを使用してメッセージを転送するための回避策

CloudWatch アラーム用にクロスアカウントの Amazon SNS 通知を有効にするには、次の手順を実行します。

1. CloudWatch アラーム (111122223333) と同じアカウントに [Amazon SNS トピック](#) を作成します。
2. その Amazon SNS トピックに [Lambda 関数](#) または [Amazon EventBridge ルール](#) をサブスクライブします。
3. これにより、Lambda 関数や EventBridge ルールは、ターゲットアカウント (444455556666) の Amazon SNS トピックにメッセージを発行できます。

Amazon SNS トピックの発行を特定の VPC エンドポイントのみからに制限する

この場合、アカウント 444455556666 内のトピックは、ID vpce-1ab2c34d の VPC エンドポイントのみからの発行を許可されます。

```
{
  "Statement": [{
    "Effect": "Deny",
    "Principal": "*",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-2:444455556666:MyTopic",
    "Condition": {
      "StringNotEquals": {
        "aws:sourceVpce": "vpce-1ab2c34d"
      }
    }
  }]
}
```

Amazon SNS と IAM が連携する仕組み

IAM を使用して Amazon SNS へのアクセスを管理する前に、Amazon SNS で使用できる IAM 機能について理解しておく必要があります。

Amazon Simple Notification Service で使用できる IAM の機能

IAM 機能	Amazon SNS のサポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	はい
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACL	なし
ABAC (ポリシー内のタグ)	部分的
一時認証情報	あり
プリンシパルアクセス権限	あり
サービスロール	あり
サービスリンクロール	いいえ

Amazon SNS およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要については、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。

AWS の 管理ポリシー Amazon Simple Notification Service

AWS 管理ポリシーは、によって作成および管理されるスタンドアロンポリシーです AWS。AWS 管理ポリシーは、ユーザー、グループ、ロールにアクセス許可の割り当てを開始できるように、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ように設計されています。

AWS 管理ポリシーは、すべての AWS お客様が使用できるため、特定のユースケースに対して最小特権のアクセス許可を付与しない場合があります。ユースケースに固有の[カスタマー管理ポリシー](#)を定義して、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義されているアクセス許可は変更できません。が AWS マネージドポリシーで定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。AWS は、新しい が起動されるか、新しい API オペレーション AWS のサービス が既存のサービスで使えるようになったときに、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。

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](#)」を参照してください。

AWS マネージドポリシー: AmazonSNSFullAccess

AmazonSNSFullAccess は、を使用して Amazon SNS へのフルアクセスを提供します AWS マネジメントコンソール。このポリシーには、Amazon SNS を使用して呼び出 AWS End User Messaging SMS されたときの の以下の読み取りおよび書き込みアクションも含まれています。Amazon SNS このポリシーをユーザー、グループ、またはロールにアタッチできます。

アクセス許可の詳細

次のアクセス許可は、Amazon SNS API を使用する場合にのみ適用されます。

- `sns:*` – Amazon SNS に関連するアクションを実行するための完全なアクセス許可を許可します。このワイルドカード (*) は、ユーザーが可能なすべての Amazon SNS アクションを実行できることを意味します。
- `sms-voice:DescribeVerifiedDestinationNumbers` – AWS アカウント内で SMS メッセージを送信するために検証された電話番号のリストを取得できます。
- `sms-voice:CreateVerifiedDestinationNumber` – 内の SMS メッセージングサービスで使用する新しい電話番号を検証できます AWS。
- `sms-voice:SendDestinationNumberVerificationCode` – AWS内で SMS メッセージングの検証中の電話番号に検証コードを送信できます。
- `sms-voice:SendTextMessage` – 新しいテキストメッセージを作成し、受信者の電話番号に送信できます。SendTextMessage は、呼び出されるたびに SMS メッセージを 1 人の受信者にのみ送信します。

- `sms-voice:DeleteVerifiedDestinationNumber` – 内の検証済み番号のリストから電話番号を削除できます。AWS アカウント
- `sms-voice:VerifyDestinationNumber` – AWS内の SMS メッセージングサービスに使用する電話番号の検証プロセスを開始および完了できます。
- `sms-voice:DescribeAccountAttributes` – AWS内の SMS メッセージングサービスに関連するアカウントレベルの属性に関する詳細情報を取得できます。
- `sms-voice:DescribeSpendLimits` – AWS アカウント内の SMS メッセージングサービスに関連する支出制限に関する情報を取得できます。
- `sms-voice:DescribePhoneNumbers` – AWS アカウント 内の SMS メッセージングサービスに関連する電話番号に関する詳細情報を取得できます。
- `sms-voice:SetTextMessageSpendLimitOverride` – 内の SMS テキストメッセージの使用制限を設定または上書きできます。AWS アカウント
- `sms-voice:DescribeOptedOutNumbers` – AWS アカウントからの SMS メッセージの受信をオプトアウトした電話番号のリストを取得できます。
- `sms-voice>DeleteOptedOutNumber` – 内のオプトアウトされた番号のリストから電話番号を削除できます。AWS アカウント

AmazonSNSFullAccess ポリシーの例

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SNSFullAccess",
      "Effect": "Allow",
      "Action": "sns:*",
      "Resource": "*"
    },
    {
      "Sid": "SMSAccessViaSNS",
      "Effect": "Allow",
      "Action": [
        "sms-voice:DescribeVerifiedDestinationNumbers",
        "sms-voice:CreateVerifiedDestinationNumber",
        "sms-voice:SendDestinationNumberVerificationCode",
```

```

        "sms-voice:SendTextMessage",
        "sms-voice:DeleteVerifiedDestinationNumber",
        "sms-voice:VerifyDestinationNumber",
        "sms-voice:DescribeAccountAttributes",
        "sms-voice:DescribeSpendLimits",
        "sms-voice:DescribePhoneNumbers",
        "sms-voice:SetTextMessageSpendLimitOverride",
        "sms-voice:DescribeOptedOutNumbers",
        "sms-voice:DeleteOptedOutNumber"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:CalledViaLast": "sns.amazonaws.com"
        }
    }
}
]
}

```

このポリシーのアクセス許可を表示するには、「AWS マネージドポリシーリファレンス」の「[AmazonSNSFullAccess](#)」を参照してください。

AWS マネージドポリシー: AmazonSNSReadOnlyAccess

AmazonSNSReadOnlyAccess は AWS マネジメントコンソールを使用して Amazon SNS への読み取り専用アクセスを提供します。このポリシーには、Amazon SNS を使用して呼び出 AWS End User Messaging SMS されたときの 次の読み取り専用アクションも含まれます。Amazon SNS このポリシーをユーザー、グループ、およびロールにアタッチできます。

アクセス許可の詳細

次のアクセス許可は、Amazon SNS API を使用する場合にのみ適用されます。

- `sns:GetTopicAttributes` – Amazon SNS トピックの属性を取得できます。これには、トピックの ARN (Amazon リソースネーム)、サブスクライバーのリスト、配信ポリシー、アクセスコントロールポリシー、トピックに関連付けられたその他のメタデータなどの情報が含まれます。
- `sns:List*` – Amazon SNS リソースの List で始まるオペレーションを実行できます。これには、Amazon SNS に関連する、次のようなさまざまな要素を一覧表示するアクセス許可が含まれます。

- `sns:ListTopics` – AWS アカウントですべての Amazon SNS トピックのリストを取得できます。
- `sns:ListSubscriptions` – Amazon SNS トピックへのすべてのサブスクリプションのリストを取得できます。
- `sns:ListSubscriptionsByTopic` – 特定の Amazon SNS トピックのすべてのサブスクリプションを一覧表示できます。
- `sns:ListPlatformApplications` – モバイルプッシュ通知用に作成されたすべてのプラットフォームアプリケーションを一覧表示できます。
- `sns:ListEndpointsByPlatformApplication` – プラットフォームアプリケーションに関連付けられたすべてのエンドポイントを一覧表示できます。
- `sns:CheckIfPhoneNumberIsOptedOut` – 特定の電話番号が Amazon SNS を介した SMS メッセージの受信をオプトアウトしたかどうかを確認できます。
- `sns:GetEndpointAttributes` – Amazon SNS プラットフォームアプリケーションに関連付けられたエンドポイントの属性を取得できます。これには、エンドポイントの有効なステータス、カスタムユーザーデータ、エンドポイントに関連付けられたその他のメタデータなどの属性が含まれる場合があります。
- `sns:GetDataProtectionPolicy` – Amazon SNS トピックに関連付けられたデータ保護ポリシーを取得できます。
- `sns:GetPlatformApplicationAttributes` – Amazon SNS プラットフォームアプリケーションの属性を取得できます。プラットフォームアプリケーションは Amazon SNS で使用され、Apple Push Notification Service (APNS) や Firebase Cloud Messaging (FCM) などのサービスを通じて、モバイルデバイスにプッシュ通知を送信します。
- `sns:GetSMSAttributes` – AWS アカウントのデフォルトの SMS 設定を取得できます。
- `sns:GetSMSSandboxAccountStatus` – AWS アカウントの SMS サンドボックスの現在のステータスを取得できます。
- `sns:GetSubscriptionAttributes` – Amazon SNS トピックへの特定のサブスクリプションの属性を取得できます。
- `sms-voice:DescribeVerifiedDestinationNumbers` – 内で SMS メッセージを送信するために検証された電話番号のリストを表示または取得できます。AWS アカウント
- `sms-voice:DescribeAccountAttributes` – AWS内の SMS メッセージングサービスに関連するアカウントレベルの属性に関する情報を表示または取得できます。
- `sms-voice:DescribeSpendLimits` – AWS アカウントの SMS メッセージングサービスに関連する支出制限に関する情報を表示または取得できます。

- `sms-voice:DescribePhoneNumbers` – AWS アカウントの SMS メッセージングサービスに使用されている電話番号に関する情報を表示または取得できます。
- `sms-voice:DescribeOptedOutNumbers` – AWS アカウントからの SMS メッセージの受信をオプトアウトした電話番号のリストを表示または取得できます。

AmazonSNSReadOnlyAccess ポリシーの例

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SNSReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "sns:GetTopicAttributes",
        "sns:List*",
        "sns:CheckIfPhoneNumberIsOptedOut",
        "sns:GetEndpointAttributes",
        "sns:GetDataProtectionPolicy",
        "sns:GetPlatformApplicationAttributes",
        "sns:GetSMSAttributes",
        "sns:GetSMSSandboxAccountStatus",
        "sns:GetSubscriptionAttributes"
      ],
      "Resource": "*"
    },
    {
      "Sid": "SMSAccessViaSNS",
      "Effect": "Allow",
      "Action": [
        "sms-voice:DescribeVerifiedDestinationNumbers",
        "sms-voice:DescribeAccountAttributes",
        "sms-voice:DescribeSpendLimits",
        "sms-voice:DescribePhoneNumbers",
        "sms-voice:DescribeOptedOutNumbers"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
```



```
        "aws:CalledViaLast": "sns.amazonaws.com"
    }
}
}
```

このポリシーのアクセス許可を表示するには、「AWS マネージドポリシーリファレンス」の「[AmazonSNSFullAccess](#)」を参照してください。

AWS マネージドポリシーへの Amazon SNS 更新

このサービスがこれらの変更の追跡を開始してからの Amazon SNS の AWS マネージドポリシーの更新に関する詳細を表示します。このページへの変更に関する自動アラートを受け取るには、Amazon SNS のドキュメント履歴ページで 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
AmazonSNSFullAccess - 既存ポリシーへの更新	Amazon SNS は、AWS マネジメントコンソールを使用して Amazon SNS へのフルアクセスを許可する新しいアクセス許可を追加しました。	09/24/2024
AmazonSNSReadOnlyAccess – 既存のポリシーの更新	Amazon SNS は、AWS マネジメントコンソールを使用して Amazon SNS への読み取り専用アクセスを許可する新しいアクセス許可を追加しました。	09/24/2024
Amazon SNS が変更の追跡を開始しました。	Amazon SNS は、AWS 管理ポリシーの変更の追跡を開始しました。	08/27/2024

Amazon SNS のポリシーアクション

ポリシーアクションのサポート: あり

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。

Amazon SNS アクションの一覧については、「サービス認可リファレンス」の「[Amazon Simple Notification Service で定義されるリソース](#)」を参照してください。

Amazon SNS のポリシーアクションは、アクションの前に次のプレフィックスを使用します。

```
sns
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [  
    "sns:action1",  
    "sns:action2"  
]
```

Amazon SNS のアイデンティティベースポリシーの例を確認するには、「[Amazon Simple Notification Service のアイデンティティベースのポリシーの例](#)」を参照してください。

Amazon SNS のポリシーリソース

ポリシーリソースのサポート: あり

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": "*"
```

Amazon SNS リソースタイプとその ARN の一覧については、「サービス認可リファレンス」の「[Amazon Simple Notification Service で定義されるアクション](#)」を参照してください。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、「[Resources Defined by Amazon Simple Notification Service](#)」(Amazon Simple Notification Service で定義されるリソース) を参照してください。

Amazon SNS のアイデンティティベースポリシーの例を確認するには、「[Amazon Simple Notification Service のアイデンティティベースのポリシーの例](#)」を参照してください。

Amazon SNS のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

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。

Amazon SNS 条件キーの一覧については、「サービス認可リファレンス」の「[Amazon Simple Notification Service の条件キー](#)」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、「[Resources Defined by Amazon Simple Notification Service](#)」(Amazon Simple Notification Service で定義されるリソース) を参照してください。

Amazon SNS のアイデンティティベースポリシーの例を確認するには、「[Amazon Simple Notification Service のアイデンティティベースのポリシーの例](#)」を参照してください。

Amazon SNS での ACL

ACL のサポート: なし

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 はリソーススペ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 SNS での ABAC

ABAC (ポリシー内のタグ) のサポート: 一部

属性ベースのアクセスコントロール (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

タグに基づいてアクセスを管理するには、`aws: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。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「[ABAC 認可でアクセス許可を定義する](#)」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「[属性ベースのアクセスコントロール \(ABAC\) を使用する](#)」を参照してください。

Amazon SNS での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一時的な認証情報は、AWS リソースへの短期アクセスを提供し、フェデレーションまたは切り替えロールを使用する場合に自動的に作成されます。長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成 AWS 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「[IAM の一時的な認証情報](#)」および「[IAM と連携するAWS のサービス](#)」を参照してください。

Amazon SNS のクロスサービスプリンシパル許可

転送アクセスセッション (FAS) のサポート: あり

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。FAS

リクエストを行う際のポリシーの詳細については、「[転送アクセスセッション](#)」を参照してください。

Amazon SNS のサービスロール

サービスロールのサポート: あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける [IAM ロール](#) です。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、IAM ユーザーガイドの [AWS のサービスに許可を委任するロールを作成する](#) を参照してください。

Warning

サービスロールの許可を変更すると、Amazon SNS の機能が破損する可能性があります。Amazon SNS が指示する場合以外は、サービスロールを編集しないでください。

Amazon SNS のサービスにリンクされたロール

サービスにリンクされたロールのサポート: なし

サービスにリンクされたロールは、にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「[IAM と提携するAWS のサービス](#)」を参照してください。表の「サービスリンクロール」列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

Amazon Simple Notification Service の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、ユーザーおよびロールには Amazon SNS リソースを作成または変更する許可はありません。IAM 管理者は、リソースで必要なアクションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの JSON ポリシードキュメントを使用して 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「[IAM ポリシーを作成する \(コンソール\)](#)」を参照してください。

Amazon SNS が定義するアクションとリソースタイプ (リソースタイプごとの ARN の形式を含む) の詳細については、「サービス認可リファレンス」の「[Amazon Simple Notification Service 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。

ポリシーに関するベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内で誰かが Amazon SNS リソースを作成、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 – 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。

Amazon SNS コンソールの使用

Amazon Simple Notification Service コンソールにアクセスするには、許可の最小限のセットが必要です。これらのアクセス許可により、の Amazon SNS 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。

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。

ユーザーとロールが引き続き Amazon SNS 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 Amazon SNS *ConsoleAccess* または *ReadOnly* AWS 管理ポリシーもアタッチします。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。

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最大数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。

自分の権限の表示をユーザーに許可する

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": "*"
}
]
```

Amazon SNS のアイデンティティベースの ポリシー

ID ベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。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。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。JSON ポリシーで使用するすべての要素について学ぶには、「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。

Amazon SNS のアイデンティティベースのポリシー例

Amazon SNS のアイデンティティベースポリシーの例を確認するには、「[Amazon Simple Notification Service のアイデンティティベースのポリシーの例](#)」を参照してください。

Amazon SNS 内のリソースベースのポリシー

リソースベースのポリシーのサポート	はい

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。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。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コントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーで、[プリンシパルを指定する](#)必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、またはを含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。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。

Amazon SNS でのアイデンティティベースのポリシーを使用する

Amazon Simple Notification Service は AWS Identity and Access Management (IAM) と統合されているため、 のユーザーが Amazon SNS リソースで AWS アカウント 実行できる Amazon SNS アクションを指定できます。ポリシーで特定のトピックを指定できます。例えば、Publish アクションを AWS アカウントの特定のトピックで使用するアクセス権限を組織の特定のユーザーに付与する IAM ポリシーを作成するとき、変数を使用できます。詳細については、『IAM の使用』ガイドの「[ポリシー変数](#)」を参照してください。

Important

IAM で Amazon SNS を使用しても、Amazon SNS の使用方法は変わりません。Amazon SNS アクションに変更はなく、ユーザーおよびアクセスコントロールに関連する新しい Amazon SNS アクションはありません。

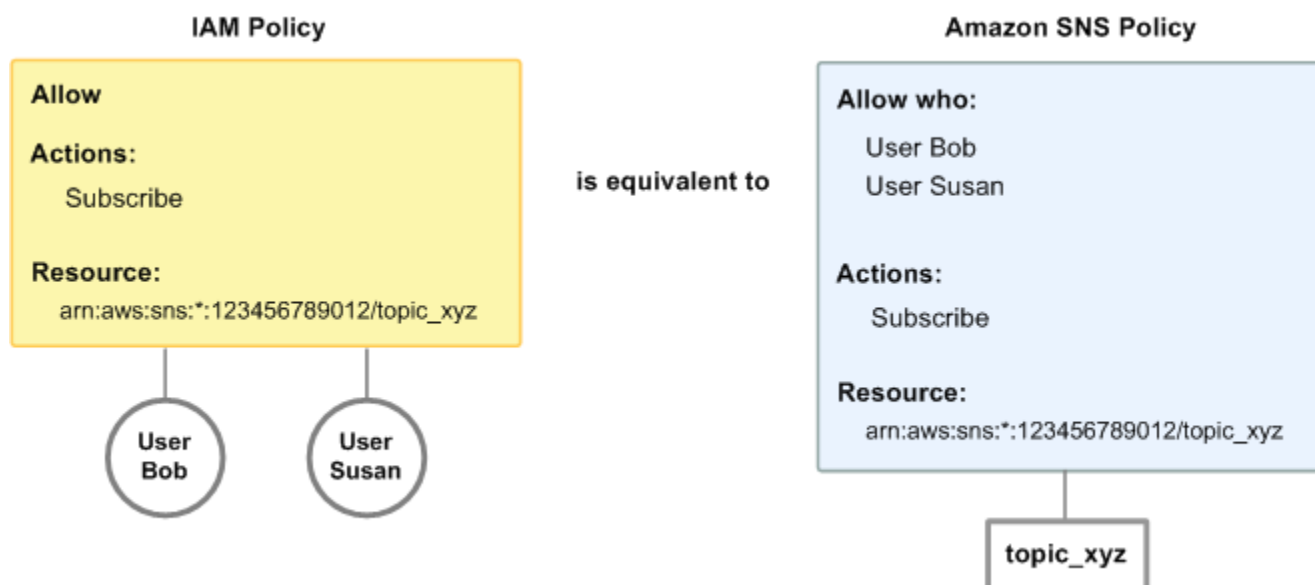
Amazon SNS アクションおよびリソースに対応するポリシーの例については、「[Amazon SNS のポリシーの例](#)」を参照してください。

IAM ポリシーと Amazon SNS ポリシーを一緒に使う

IAM ポリシーを使用して、Amazon SNS アクションおよびトピックへのユーザーのアクセスを制限します。IAM ポリシーは、他ののではなく、AWS アカウント内のユーザーのみにアクセスを制限できます AWS アカウント。

特定のトピックに関連付ける Amazon SNS ポリシーにより、そのトピックを操作できるユーザー (トピックにメッセージを発行できるユーザー、サブスクライブできるユーザーなど) を制限できます。Amazon SNS ポリシーは AWS アカウント、他のユーザーまたは独自のユーザーにアクセスを許可できます AWS アカウント。

Amazon SNS トピックのアクセス権限をユーザーに付与するには、IAM ポリシー、Amazon SNS ポリシー、またはその両方を使用できます。ほとんどの場合、どちらでも同じ結果が得られます。例えば、以下の図は、同じ働きを持つ IAM ポリシーと Amazon SNS ポリシーを示しています。IAM ポリシーは、`topic_xyz` というトピックの Amazon SNS `Subscribe` アクションを許可します。IAM ポリシーは Bob と Susan のユーザーにアタッチされます (つまり、Bob と Susan はポリシーに記載されているアクセス許可を持っていることを意味します)。AWS アカウント 同様に、Amazon SNS ポリシーは `topic_xyz` の `Subscribe` にアクセスする許可を Bob と Susan に付与します。



Note

先の例では、条件のない単純なポリシーを示しました。どちらのポリシーでも特定の条件を指定して、同じ結果を得ることができます。

AWS IAM ポリシーと Amazon SNS ポリシーには 1 つの違いがあります。Amazon SNS ポリシーシステムでは、他のポリシーにアクセス許可を付与できますが AWS アカウント、IAM ポリシーでは付与されません。

両方のシステムを同時に使用してどのようにアクセス許可を管理するかは、ニーズに応じて決めてください。以下の例では、2つのポリシーシステムがどのように連携するかを示しています。

Example 1

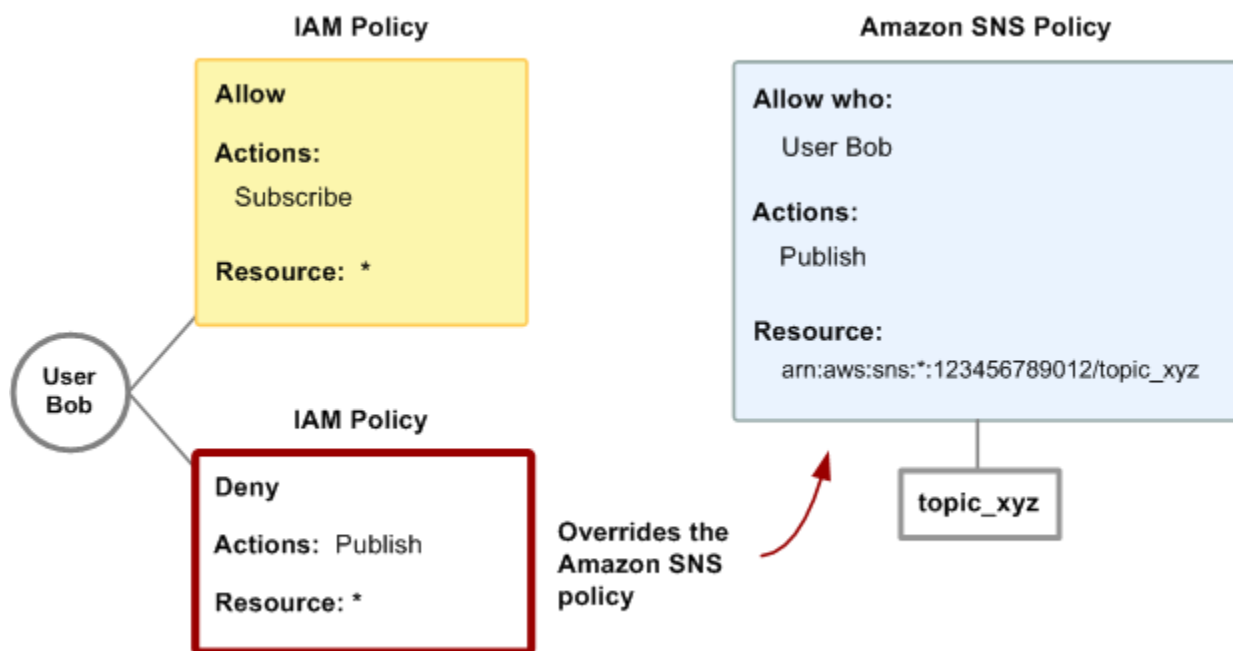
この例では、IAM ポリシーおよび Amazon SNS ポリシーの両方がボブに適用されます。IAM ポリシーは、のいずれかのトピックSubscribeに対する AWS アカウントのアクセス許可をユーザーに付与しますが、Amazon SNS ポリシーは、特定のトピック (topic_xyz) Publish で使用するアクセス許可をユーザーに付与します。以下の図に、そのコンセプトを示します。



Bob が AWS アカウントの任意のトピックにサブスクライブするリクエストを送信する場合、IAM ポリシーは アクションを許可します。ボブが topic_xyz にメッセージを発行するリクエストを送信すると、そのアクションは Amazon SNS ポリシーで許可されます。

Example 2

この例は、例 1 で示した、ボブに 2 つのポリシーが適用されている状態に基づいています。ボブは、必要のない topic_xyz にメッセージを発行しているため、トピックに発行する機能を完全に削除するとします。最も簡単な方法は、すべてのトピックに対するボブの Publish アクションを拒否するような IAM ポリシーを追加することです。この 3 番目のポリシーは、topic_xyz に発行するアクセス権限を最初にボブに与えた Amazon SNS ポリシーより優先されます。明示的拒否は、常に許可よりも優先されるためです (ポリシー評価のロジックの詳細については、「[評価論理](#)」を参照)。以下の図に、そのコンセプトを示します。



Amazon SNS アクションおよびリソースに対応するポリシーの例については、「[Amazon SNS のポリシーの例](#)」を参照してください。

Amazon SNS リソース ARN 形式

Amazon SNS では、トピックはポリシーで指定できる唯一のリソースタイプです。トピックの Amazon リソースネーム (ARN) 形式を以下に示します。

```
arn:aws:sns:region:account_ID:topic_name
```

ARN の詳細については、『IAM ユーザーガイド』の「[ARN](#)」を参照してください。

Example

以下は、AWS アカウント 123456789012 に属する us-east-2 リージョンの MyTopic という名前のトピックの ARN です。

```
arn:aws:sns:us-east-2:123456789012:MyTopic
```

Example

Amazon SNS によりサポートされる各リージョンに MyTopic という名前のトピックがある場合、次の ARN を使用してトピックを指定できます。

```
arn:aws:sns:*:123456789012:MyTopic
```

トピック名にはワイルドカード * および ? を使用できます。例えば、以下では、ボブが bob_ をプレフィックスとして付けて作成したすべてのトピックを参照します。

```
arn:aws:sns:*:123456789012:bob_*
```

利便性を高めるため、トピックを作成すると、Amazon SNS は応答でトピックの ARN を返します。

Amazon SNS API アクション

IAM ポリシーでは、Amazon SNS で提供されている任意のアクションを指定できます。ただし、ConfirmSubscription および Unsubscribe アクションでは認証を必要としません。つまり、ポリシーでこれらのアクションを指定しても、IAM はこれらのアクションへのユーザーのアクセスを制限しません。

ポリシーで指定する各アクションには、小文字の文字列 sns: をプレフィックスとして付ける必要があります。例えば、すべての Amazon SNS アクションを指定するには、sns:* を使用します。アクションのリストについては、「[Amazon Simple Notification Service API リファレンス](#)」にアクセスしてください。

Amazon SNS ポリシーキー

Amazon SNS は、以下の AWS ワイドポリシーキーと、いくつかのサービス固有のキーを実装します。

各でサポートされている条件キーのリストについては AWS のサービス、IAM ユーザーガイドの「[アクション、リソース、および条件キー AWS のサービス](#)」を参照してください。複数ので利用できる条件キーのリストについては AWS のサービス、「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。

Amazon SNS では以下のサービス固有のキーを使用します。Subscribe リクエストへのアクセスを制限するポリシーで、これらのキーを使用します。

- sns:Endpoint - Subscribe リクエストからの URL、E メールアドレス、または ARN、あるいは以前に確認されたサブスクリプション。文字列条件とともに使用 (「[Amazon SNS のポリシーの例](#)」を参照) して特定のエンドポイント (*@yourcompany.com など) へのアクセスを制限します。

- `sns:Protocol - Subscribe` リクエストまたは以前に確認されたサブスクリプションからの `protocol` 値。文字列条件とともに使用 (「[Amazon SNS のポリシーの例](#)」を参照) して、特定の配信プロトコル (`https` など) への発行を制限します。

Amazon SNS のポリシーの例

このセクションでは、Amazon SNS へのユーザーアクセスをコントロールするための簡単なポリシーをいくつか紹介します。

Note

将来的には、Amazon SNS に新しいアクションが追加される可能性があります。これらは、以下のポリシーのいずれかに論理的に含まれ、ポリシーに記載された目的に基づいている必要があります。

Example 1: グループでトピックの作成と管理を許可する

この例では、`CreateTopic`、`ListTopics`、`SetTopicAttributes`、および `DeleteTopic` へのアクセスを付与するポリシーを作成します。

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": ["sns:CreateTopic", "sns:ListTopics", "sns:SetTopicAttributes",
"sns>DeleteTopic"],
    "Resource": "*"
  }]
}
```

Example 2: IT グループが特定のトピックにメッセージを発行することを許可する

この例では、IT のグループを作成し、対象の特定のトピックで `Publish` へのアクセスを付与するポリシーを割り当てます。

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:*:123456789012:MyTopic"
  }]
}
```

```
    }]  
  }  
}
```

Example 3: ユーザーがトピックをサブスクライブ AWS アカウント できるようにする

この例では、`sns:Protocol` および `sns:Endpoint` ポリシーキーの文字列一致条件を使って `Subscribe` アクションにアクセスを付与するポリシーを作成します。

```
{  
  "Statement": [{  
    "Effect": "Allow",  
    "Action": ["sns:Subscribe"],  
    "Resource": "*",  
    "Condition": {  
      "StringLike": {  
        "sns:Endpoint": "*@example.com"  
      },  
      "StringEquals": {  
        "sns:Protocol": "email"  
      }  
    }  
  }]  
}
```

Example 4: パートナーが特定のトピックにメッセージを発行することを許可する

Amazon SNS ポリシーまたは IAM ポリシーを使用して、パートナーが特定のトピックに発行することを許可できます。パートナーに がある場合は AWS アカウント、Amazon SNS ポリシーを使用する方が簡単な場合があります。ただし、AWS セキュリティ認証情報を所有するパートナーの会社では、誰でもトピックにメッセージを発行できます。この例では、特定のユーザー (またはアプリケーション) へのアクセスを制限するとします。そのためには、パートナーを自社内のユーザーのように扱い、Amazon SNS ポリシーの代わりに IAM ポリシーを使用する必要があります。

この例では、パートナー企業を表す `WidgetCo` というグループを作成し、パートナー企業でアクセス許可が必要な特定の人 (またはアプリケーション) に対してユーザーを作成し、そのユーザーをグループに入れます。

次に、`Publish` グループに、`WidgetPartnerTopic` という名前の特定のトピックへのアクセスを付与するポリシーをアタッチします。

また、`WidgetCo` グループがトピックに対して他の操作を実行できないようにするため、`WidgetPartnerTopic` 以外のトピックで `Publish` 以外の Amazon SNS アクションへのアクセス

許可を拒否するステートメントを追加します。これは、システム内の他の場所に、Amazon SNS への幅広いアクセスをユーザーに付与する広範なポリシーが存在する場合にのみ必要です。

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:*:123456789012:WidgetPartnerTopic"
  },
  {
    "Effect": "Deny",
    "NotAction": "sns:Publish",
    "NotResource": "arn:aws:sns:*:123456789012:WidgetPartnerTopic"
  }
]
}
```

E メールエンドポイントの Amazon SNS ポリシー

Amazon SNS は、先頭と末尾の空白を削除し、すべての文字を小文字に変換し、特殊文字とエスケープシーケンスを正規化することで、エンドポイントアクセスポリシーの E メールアドレスを正規化します。そのため、サブスクリプションリクエストを正しく照合するには、ポリシー定義で小文字の E メールアドレスを使用する必要があります。

ポリシーの例:

```
{
  "Condition": {
    "StringEquals": {
      "sns:Endpoint": "user.name@example.com"
    }
  }
}
```

次の E メールアドレスは上記の条件に一致します。

- user.name@example.com
- USER.NAME@EXAMPLE.COM
- User.N\ame@Example.com

HTTP/HTTPS エンドポイントの Amazon SNS ポリシー

Amazon SNS は、ポート、パス、クエリパラメータ、フラグメントを正確に保持しながら、スキームとホスト名を小文字に変換することで、エンドポイントアクセスポリシーの HTTP/HTTPS エンドポイントを正規化します。そのため、サブスクリプションリクエストを正しく照合するには、ポリシー定義で小文字のスキームとホスト名を使用する必要があります。

```
{
  "Condition": {
    "StringEquals": {
      "sns:Endpoint": "https://example.com:443/path?A=B"
    }
  }
}
```

次の HTTP/HTTPS は上記の条件に一致します。

- `HTTPS://EXAMPLE.COM:443/path?A=B`
- `HTTPS://example.com:443/path?A=B`
- `HTTPS://ExAmPlE.cOm:443/path?A=B`

カスタム Amazon SNS IAM ポリシーの管理

カスタム IAM ポリシーを使用すると、個々の IAM ユーザー、グループ、またはロールのアクセス許可を指定し、特定の AWS リソースやアクションへのアクセスを許可または制限できます。Amazon SNS リソースを管理する場合、カスタム IAM ポリシーを使用すると、組織のセキュリティおよび運用要件に応じてアクセス許可を調整できます。

Amazon SNS のカスタム IAM ポリシーを管理するには、次の手順に従います。

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/iam/> で IAM コンソールを開きます。
2. ナビゲーションペインで [ポリシー] を選択します。
3. 新しいカスタム IAM ポリシーを作成するには、[ポリシーを成] を選択し、SNS を選択します。既存のポリシーを編集するには、リストからポリシーを選択し、[ポリシーを編集] を選択します。
4. ポリシーエディタで、Amazon SNS リソースにアクセスするためのアクセス許可を定義します。特定の要件に基づいて、アクション、リソース、および条件を指定できます。

5. Amazon SNS アクションのアクセス許可を付与するには、IAM ポリシーに関連する Amazon SNS アクション (sns:Publish、sns:Subscribe、sns>DeleteTopic など) を含めます。アクセス許可が適用される Amazon SNS トピックの ARN (Amazon リソースネーム) を定義します。
6. ポリシーをアタッチする IAM ユーザー、グループ、またはロールを指定します。ポリシーは IAM ユーザーまたはグループに直接アタッチすることも、AWS のサービス またはアプリケーションで使用される IAM ロールに関連付けることもできます。
7. IAM ポリシー設定を確認して、アクセスコントロールの要件と一致していることを確認します。確認したら、ポリシーの変更を保存します。
8. カスタム IAM ポリシーを、AWS アカウント内の関連する IAM ユーザー、グループ、またはロールにアタッチします。これにより、ポリシーで定義された Amazon SNS リソースを管理するためのアクセス許可が付与されます。

Amazon SNS で一時的なセキュリティ認証情報を使用する

AWS Identity and Access Management (IAM) では、AWS リソースへのアクセスを必要とするユーザーとアプリケーションに一時的なセキュリティ認証情報を付与できます。これらの一時的なセキュリティ認証情報は、主に IAM ロールと、SAML や OpenID Connect (OIDC) などの業界標準プロトコルを介したフェデレーティッドアクセスに使用されます。

AWS リソースへのアクセスを効果的に管理するには、以下の主要な概念を理解することが重要です。

- IAM ロール – ロールはリソースへのアクセスを委任するために使用されます AWS。ロールは、Amazon EC2 インスタンス、Lambda 関数、または他の AWS アカウントのユーザーなどのエンティティで引き受けることができます。
- フェデレーションユーザー – これらは、SAML または OIDC を使用して外部 ID プロバイダー (IdP) を介して認証されたユーザーです。フェデレーティッドアクセスは人間のユーザーに推奨され、IAM ロールはソフトウェアアプリケーションに使用する必要があります。
- Roles Anywhere – AWS アクセスを必要とする外部アプリケーションの場合、IAM Roles Anywhere を使用して、長期的な認証情報を作成せずにアクセスを安全に管理できます。

Amazon SNS に対するリクエストを作成するときに、これらの一時的なセキュリティ認証情報を使用できます。SDK ライブラリおよび API ライブラリによって、これらの認証情報を使用して必要な署名値が計算されて、リクエストが認証されます。認証情報の有効期限が切れたリクエストは、Amazon SNS によって拒否されます。

一時的なセキュリティ認証情報の詳細については、「IAM ユーザーガイド」の「[IAM ロールの使用](#)」および「[外部で認証されたユーザー \(ID フェデレーション\) へのアクセス許可](#)」を参照してください。

Example HTTPS リクエストの例

次の例は、AWS Security Token Service (STS) から取得した一時的なセキュリティ認証情報を使用して Amazon SNS リクエストを認証する方法を示しています。

```
https://sns.us-east-2.amazonaws.com/  
?Action=CreateTopic  
&Name=My-Topic  
&SignatureVersion=4  
&SignatureMethod=AWS4-HMAC-SHA256  
&Timestamp=2023-07-05T12:00:00Z  
&X-Amz-Security-Token=SecurityTokenValue  
&X-Amz-Date=20230705T120000Z  
&X-Amz-Credential=<your-access-key-id>/20230705/us-east-2/sns/aws4_request  
&X-Amz-SignedHeaders=host  
&X-Amz-Signature=<signature-value>
```

リクエストを認証する手順

1. 一時的なセキュリティ認証情報の取得 - AWS STS を使用して、ロールを引き受けたり、フェデレーションユーザーの認証情報を取得したりします。これにより、アクセスキー ID、シークレットアクセスキー、およびセキュリティトークンが提供されます。
2. リクエストの作成 - Amazon SNS アクションに必要なパラメータ (CreateTopic など) を含め、安全な通信に HTTPS を使用していることを確認します。
3. リクエストへの署名 - AWS 署名バージョン 4 プロセスを使用して、リクエストに署名します。これには、正規リクエストの作成、署名対象の文字列の作成、署名の計算が含まれます。AWS 署名バージョン 4 の詳細については、「Amazon EBS [ユーザーガイド](#)」の「[署名バージョン 4 署名](#)を使用する」を参照してください。
4. リクエストの送信 - リクエストヘッダーに X-Amz-Security-Token を含め、一時的なセキュリティ認証情報を Amazon SNS に渡します。

Amazon SNS API のアクセス許可: アクションとリソースのリファレンス

次のリストは、Amazon SNS でのアクセスコントロールの実装に固有の情報を示しています。

- 各ポリシーは、1 つのトピックだけを対象とする必要があります (ポリシーを作成する際は、異なるトピックを対象とするステートメントを含めないでください)。
- 各ポリシーには固有のポリシー (Id) が必要です。
- ポリシーを構成する各ステートメントには固有のステートメント (sid) が必要です。

ポリシーのクォータ

以下の表では、ポリシーステートメントの最大クォータを示しています。

名前	最大クォータ
バイト	30 KB
ステートメント	100
プリンシパル	1 ~ 200 (0 は無効)
[リソース]	1 (0 は無効です。値はポリシーのトピックの ARN と一致する必要があります)

有効な Amazon SNS ポリシーアクション

Amazon SNS は、次の表に示すアクションをサポートします。

アクション	説明
sns:AddPermission	トピックポリシーへのアクセス権限の追加を許可します。
sns>DeleteTopic	トピックを削除する許可を与えます。
sns:GetDataProtectionPolicy	トピックのデータ保護ポリシーを返すアクセス許可を付与
sns:GetTopicAttributes	すべてのトピック属性を受け取る許可を与えます。
sns:ListSubscriptionsByTopic	特定のトピックへのサブスクリプションをすべて取得することを許可します。

アクション	説明
sns:ListTagsForResource	指定したトピックに追加されたすべてのタグを一覧表示する許可を与えます。
sns:Publish	トピックやエンドポイントへの公開とバッチの公開の両方に対する許可を付与します。詳細については、「Amazon Simple Notification Service API リファレンス」の「 Publish 」および「 PublishBatch 」を参照してください。
sns:PutDataProtectionPolicy	トピックのデータ保護ポリシーを設定するアクセス許可を付与
sns:RemovePermission	トピックポリシーのアクセス権限を削除する許可を与えます。
sns:SetTopicAttributes	トピックの属性を設定する許可を与えます。
sns:Subscribe	トピックにサブスクライブすることを許可します。

サービス固有のキー

Amazon SNS では以下のサービス固有のキーを使用します。これらは、Subscribe リクエストへのアクセスを制限するポリシーに使用できます。

- sns:endpoint - Subscribe リクエストからの URL、E メールアドレス、または ARN、あるいは以前に確認されたサブスクリプション。文字列条件とともに使用 (「[Amazon SNS のポリシーの例](#)」を参照) して、特定のエンドポイント (*@example.com など) へのアクセスを制限します。
- sns:Protocol - Subscribe リクエストまたは以前に確認されたサブスクリプションからの protocol 値。文字列条件とともに使用 (「[Amazon SNS のポリシーの例](#)」を参照) して、特定の配信プロトコル (https など) への発行を制限します。

Important

sns:Endpoint によるアクセスの制御にポリシーを使用するときは、将来、DNS の問題がエンドポイントの名前解決に影響する可能性があるため、注意が必用です。

Amazon Simple Notification Service アイデンティティとアクセスのトラブルシューティング

Amazon SNS と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復には、次の情報を利用してください。

Amazon SNS でアクションを実行する認可がありません

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。

以下のエラー例は、mateojackson ユーザーがコンソールを使用して架空の *my-example-widget* リソースに関する詳細情報を表示しようとしているが、架空の `sns:GetWidget` 許可がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
sns:GetWidget on resource: my-example-widget
```

この場合、Mateo のポリシーでは、*my-example-widget* アクションを使用して `sns:GetWidget` リソースへのアクセスを許可するように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。サインイン認証情報を提供した担当者が管理者です。

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 Amazon SNS にロールを渡せ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アクセス許可が必要です。

次の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して Amazon SNS 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アクセス許可が必要です。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 Amazon SNS 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、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:

- Amazon SNS がこれらの機能をサポートしているかどうかを確認するには、「[Amazon SNS と IAM が連携する仕組み](#)」を参照してください。
- 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 AWS アカウント が所有する へのアクセスを提供する](#)」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。

Amazon SNS でのログ記録とモニタリング

Amazon SNS では、CloudTrail で API コールをログ記録し、CloudWatch でトピックをモニタリングして、メッセージングアクティビティを追跡およびモニタリングできます。これらのツールは、メッセージ配信に関するインサイトの取得、問題のトラブルシューティング、メッセージングワークフローの正常性の確保に役立ちます。このトピックでは次の内容について説明します。

- [を使用した AWS SNS API コールのログ記録 AWS CloudTrail](#)。このログ記録によって、トピックの作成、サブスクリプションの管理、メッセージの発行など、Amazon SNS トピックで実行され

たアクションを追跡できます。CloudTrail ログを分析すると、特定の API リクエストを実行したユーザーとそのリクエストの実行日時を特定できるため、Amazon SNS の使用状況の監査とトラブルシューティングに役立ちます。

- [CloudWatch を使用した Amazon SNS のモニタリング](#)。CloudWatch が提供するメトリクスを利用すると、Amazon SNS トピックのパフォーマンスと状態をリアルタイムで観察できます。これらのメトリクスに基づいてアラームを設定すると、配信障害やメッセージレイテンシーの増加などの異常に迅速に対応できます。このモニタリング機能により、潜在的な問題にプロアクティブに対処して、SNS ベースのメッセージングシステムの信頼性を維持できます。

を使用した AWS SNS API コールのログ記録 AWS CloudTrail

AWS SNS は、ユーザー[AWS CloudTrail](#)、ロール、または によって実行されたアクションを記録するサービスであると統合されています AWS のサービス。CloudTrail は、SNS のすべての API コールをイベントとしてキャプチャします。キャプチャされたコールには、SNS コンソールからのコールと、SNS API オペレーションへのコードのコールが含まれます。CloudTrail で収集された情報を使用して、SNS に対するリクエスト、リクエスト元の IP アドレス、リクエストの作成日時、その他の詳細を確認できます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます。

- ルートユーザーまたはユーザー認証情報のどちらを使用してリクエストが送信されたか。
- リクエストが IAM Identity Center ユーザーに代わって行われたかどうか。
- 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テンポラリ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 のサービスによって送信されたかどうか。

CloudTrail は、アカウントを作成する AWS アカウント と でアクティブになり、CloudTrail イベント履歴に自動的にアクセスできます。CloudTrail の [イベント履歴] では、AWS リージョンで過去 90 日間に記録された管理イベントの表示、検索、およびダウンロードが可能で、変更不可能な記録を確認できます。詳細については、「AWS CloudTrail ユーザーガイド」の「[CloudTrail イベント履歴の使用](#)」を参照してください。[イベント履歴] の閲覧には CloudTrail の料金はかかりません。

AWS アカウント 過去 90 日間のイベントの継続的な記録については、証跡または [CloudTrail Lake](#) イベントデータストアを作成します。

CloudTrail 証跡

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。を使用して作成されたすべての証跡 AWS マネジメントコンソール はマルチリージョンです。AWS CLIを使用する際は、単一リージョンまたは複数リージョンの証跡を作成できます。アカウント AWS リージョン 内のすべての アクティビティをキャプチャするため、マルチリージョン証跡を作成することをお勧めします。単一リージョンの証跡を作成する場合、証跡の AWS リージョンに記録されたイベントのみを表示できます。証跡の詳細については、「AWS CloudTrail ユーザーガイド」の「[AWS アカウントの証跡の作成](#)」および「[組織の証跡の作成](#)」を参照してください。

証跡を作成すると、進行中の管理イベントのコピーを 1 つ無料で CloudTrail から Amazon S3 バケットに配信できますが、Amazon S3 ストレージには料金がかかります。CloudTrail の料金の詳細については、「[AWS CloudTrail の料金](#)」を参照してください。Amazon S3 の料金に関する詳細については、「[Amazon S3 の料金](#)」を参照してください。

CloudTrail Lake イベントデータストア

[CloudTrail Lake] を使用すると、イベントに対して SQL ベースのクエリを実行できます。CloudTrail Lake は、行ベースの JSON 形式の既存のイベントを [Apache ORC](#) 形式に変換します。ORC は、データを高速に取得するために最適化された単票ストレージ形式です。イベントは、イベントデータストアに集約されます。イベントデータストアは、[高度なイベントセレクト](#)を適用することによって選択する条件に基づいた、イベントのイミュータブルなコレクションです。どのイベントが存続し、クエリに使用できるかは、イベントデータストアに適用するセレクトが制御します。CloudTrail Lake の詳細については、AWS CloudTrail ユーザーガイドの[AWS CloudTrail 「Lake の使用」](#)を参照してください。

CloudTrail Lake のイベントデータストアとクエリにはコストがかかります。イベントデータストアを作成する際に、イベントデータストアに使用する[料金オプション](#)を選択します。料金オプションによって、イベントの取り込みと保存にかかる料金、および、そのイベントデータストアのデフォルトと最長の保持期間が決まります。CloudTrail の料金の詳細については、「[AWS CloudTrail の料金](#)」を参照してください。

CloudTrail の SNS データイベント

[データイベント](#)では、リソース上またはリソース内で実行されるリソースオペレーション (Amazon S3 オブジェクトの読み取りまたは書き込みなど) についての情報が得られます。これらのイベントは、データプレーンオペレーションとも呼ばれます。データイベントは、多くの場合、高ボリュームのアクティビティです。デフォルトでは、CloudTrail はデータイベントをログ記録しません。CloudTrail [イベント履歴] にはデータイベントは記録されません。

追加の変更がイベントデータに適用されます。CloudTrail の料金の詳細については、「[AWS CloudTrail の料金](#)」を参照してください。

CloudTrail コンソール、または CloudTrail CloudTrail API オペレーションを使用して AWS CLI、SNS リソースタイプのデータイベントを記録できます。データイベントをログに記録する方法の詳細については、「AWS CloudTrail ユーザーガイド」の「[AWS マネジメントコンソールを使用したデータイベントのログ記録](#)」および「[AWS Command Line Interfaceを使用したデータイベントのログ記録](#)」を参照してください。

次の表に、データイベントをログ記録できる SNS リソースタイプを示します。データイベントタイプ (コンソール) 列には、CloudTrail コンソールの[データイベントタイプ]リストから選択する値が表示されます。resources.type 値列には、AWS CLI または CloudTrail APIs を使用して高度なイベントセレクトタを設定するときに指定する resources.type 値が表示されます。CloudTrail に記録されたデータ API 列には、リソースタイプの CloudTrail にログ記録された API コールが表示されます。

データイベントタイプ (コンソール)	resources.type 値	CloudTrail にログ記録されたデータ API
SNS トピック	AWS::SNS::Topic	• Publish • PublishBatch
SNS プラットフォームエンドポイント	AWS::SNS::PlatformEndpoint	• Publish 詳細については、AWS CloudTrail API リファレンス AdvancedEventSelector の「」を参照してください。

Note

SNS リソースタイプ `AWS::SNS::PhoneNumber` は、CloudTrail によってログ記録されません。

eventName、readOnly、および resources.ARN フィールドでフィルタリングして、自分にとって重要なイベントのみをログに記録するように高度なイベントセレクトタを設定できます。オブジェク

トの詳細については、「AWS CloudTrail API リファレンス」の「[AdvancedFieldSelector](#)」を参照してください。

データイベントのログ記録の詳細については、「CloudTrail ユーザーガイド」の「[データイベントのログ記録](#) AWS マネジメントコンソール」および AWS CLI「[データイベントのログ記録](#)」を参照してください。

CloudTrail の SNS 管理イベント

[管理イベント](#)は、のリソースで実行される管理オペレーションに関する情報を提供します AWS アカウント。これらのイベントは、コントロールプレーンオペレーションとも呼ばれます。CloudTrail は、デフォルトで管理イベントをログ記録します。

AWS SNS は、次の SNS コントロールプレーンオペレーションを管理イベントとして CloudTrail に記録します。

- [AddPermission](#)
- [CheckIfPhoneNumberIsOptedOut](#)
- [ConfirmSubscription](#)
- [CreatePlatformApplication](#)
- [CreatePlatformEndpoint](#)
- [CreateSMSSandboxPhoneNumber](#)
- [CreateTopic](#)
- [DeleteEndpoint](#)
- [DeletePlatformApplication](#)
- [DeleteSMSSandboxPhoneNumber](#)
- [DeleteTopic](#)
- [GetDataProtectionPolicy](#)
- [GetEndpointAttributes](#)
- [GetPlatformApplicationAttributes](#)
- [GetSMSAttributes](#)
- [GetSMSSandboxAccountStatus](#)
- [GetSubscriptionAttributes](#)
- [GetTopicAttributes](#)

- [ListEndpointsByPlatformApplication](#)
- [ListOriginationNumbers](#)
- [ListPhoneNumbersOptedOut](#)
- [ListPlatformApplications](#)
- [ListSMSSandboxPhoneNumbers](#)
- [ListSubscriptions](#)
- [ListSubscriptionsByTopic](#)
- [ListTagsForResource](#)
- [ListTopics](#)
- [OptInPhoneNumber](#)
- [PutDataProtectionPolicy](#)
- [RemovePermission](#)
- [SetEndpointAttributes](#)
- [SetPlatformApplicationAttributes](#)
- [SetSMSAttributes](#)
- [SetSubscriptionAttributes](#)
- [SetTopicAttributes](#)
- [Subscribe](#)
- [TagResource](#)
- [Unsubscribe](#)
- [UntagResource](#)
- [VerifySMSSandboxPhoneNumber](#)

 Note

Amazon Web Services にログインしていない状態 (非認証モード)

で、[ConfirmSubscription](#) または [Unsubscribe](#) アクションのいずれかが呼び出された場合、そのアクションは CloudTrail に記録されません。例えば、E メール通知内のリンクを選択して、あるトピックへの保留中のサブスクリプションを確認したとき、ConfirmSubscription アクションが非認証モードで呼び出されたとします。この場合、ConfirmSubscription アクションは CloudTrail に記録されません。

SNS イベントの例

各イベントは任意の送信元からの単一のリクエストを表し、リクエストされた API オペレーション、オペレーションの日時、リクエストパラメータなどに関する情報を含みます。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、イベントは特定の順序で表示されません。

次は、**ListTopics**、**CreateTopic**、および **DeleteTopic** の各アクションを示す CloudTrail イベントの例です。

```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "userName": "Bob",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Bob",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
      },
      "eventTime": "2014-09-30T00:00:00Z",
      "eventSource": "sns.amazonaws.com",
      "eventName": "ListTopics",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "aws-sdk-java/unknown-version",
      "requestParameters": {
        "nextToken": "ABCDEF1234567890EXAMPLE=="
      },
      "responseElements": null,
      "requestID": "example1-b9bb-50fa-abdb-80f274981d60",
      "eventID": "example0-09a3-47d6-a810-c5f9fd2534fe",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "userName": "Bob",
        "principalId": "EX_PRINCIPAL_ID",
```

```

    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
  },
  "eventTime": "2014-09-30T00:00:00Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "CreateTopic",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-java/unknown-version",
  "requestParameters": {
    "name": "hello"
  },
  "responseElements": {
    "topicArn": "arn:aws:sns:us-west-2:123456789012:hello-topic"
  },
  "requestID": "example7-5cd3-5323-8a00-f1889011fee9",
  "eventID": "examplec-4f2f-4625-8378-130ac89660b1",
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
},
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "userName": "Bob",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
  },
  "eventTime": "2014-09-30T00:00:00Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "DeleteTopic",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-java/unknown-version",
  "requestParameters": {
    "topicArn": "arn:aws:sns:us-west-2:123456789012:hello-topic"
  },
  "responseElements": null,
  "requestID": "example5-4faa-51d5-aab2-803a8294388d",
  "eventID": "example8-6443-4b4d-abfd-1b867280d964",
  "eventType": "AwsApiCall",

```

```

    "recipientAccountId": "123456789012"
  }
]
}

```

以下の例は、**Publish** アクションを示す CloudTrail イベントエントリです。

```

{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/Admin",
        "accountId": "123456789012",
        "userName": "ExampleUser"
      },
      "attributes": {
        "creationDate": "2023-08-21T16:44:05Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2023-08-21T16:48:37Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "Publish",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "topicArn": "arn:aws:sns:us-east-1:123456789012:ExampleSNSTopic",
    "message": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "subject": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "messageStructure": "json",
    "messageAttributes": "HIDDEN_DUE_TO_SECURITY_REASONS"
  }
}

```



```

},
"responseElements": {
  "messageId": "0787cd1e-d92b-521c-a8b4-90434e8ef840"
},
"requestID": "0a8ab208-11bf-5e01-bd2d-ef55861b545d",
"eventID": "bb3496d4-5252-4660-9c28-3c6aebdb21c0",
"readOnly": false,
"resources": [
  {
    "accountId": "123456789012",
    "type": "AWS::SNS::Topic",
    "ARN": "arn:aws:sns:us-east-1:123456789012:ExampleSNSTopic"
  }
],
"eventType": "AwsApiCall",
"managementEvent": false,
"recipientAccountId": "123456789012",
"eventCategory": "Data",
"tlsDetails": {
  "tlsVersion": "TLSv1.2",
  "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
  "clientProvidedHostHeader": "sns.us-east-1.amazonaws.com"
}
}

```

以下の例は、**PublishBatch** アクションを示す CloudTrail イベントエントリです。

```

{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::123456789012:user/Bob",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/Admin",
        "accountId": "123456789012",
        "userName": "ExampleUser"
      }
    }
  },

```

```

    "attributes": {
      "creationDate": "2023-08-21T19:20:49Z",
      "mfaAuthenticated": "false"
    }
  },
  "eventTime": "2023-08-21T19:22:01Z",
  "eventSource": "sns.amazonaws.com",
  "eventName": "PublishBatch",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "aws-cli/1.29.16 md/Botocore#1.31.16 ua/2.0 os/
linux#5.4.250-173.369.amzn2int.x86_64 md/arch#x86_64 lang/python#3.8.17 md/
pyimpl#CPython cfg/retry-mode#legacy botocore/1.31.16",
  "requestParameters": {
    "topicArn": "arn:aws:sns:us-east-1:123456789012:ExampleSNSTopic",
    "publishBatchRequestEntries": [
      {
        "id": "1",
        "message": "HIDDEN_DUE_TO_SECURITY_REASONS"
      },
      {
        "id": "2",
        "message": "HIDDEN_DUE_TO_SECURITY_REASONS"
      }
    ]
  },
  "responseElements": {
    "successful": [
      {
        "id": "1",
        "messageId": "30d68101-a64a-5573-9e10-dc5c1dd3af2f"
      },
      {
        "id": "2",
        "messageId": "c0aa0c5c-561d-5455-b6c4-5101ed84de09"
      }
    ],
    "failed": []
  },
  "requestID": "e2cdf7f3-1b35-58ad-ac9e-aaaaea0ace2f1",
  "eventID": "10da9a14-0154-4ab6-b3a5-1825b229a7ed",
  "readOnly": false,
  "resources": [

```

```
{
  "accountId": "123456789012",
  "type": "AWS::SNS::Topic",
  "ARN": "arn:aws:sns:us-east-1:123456789012:ExampleSNSTopic"
},
{
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "sns.us-east-1.amazonaws.com"
  }
}
```

CloudTrail レコードの内容については、「AWS CloudTrail ユーザーガイド」の「[CloudTrail record contents](#)」を参照してください。

CloudWatch を使用した Amazon SNS のモニタリング

Amazon SNS と Amazon CloudWatch は統合されているため、個々のアクティブな Amazon SNS 通知についてメトリクスを収集、表示、分析できます。Amazon SNS に対して CloudWatch を設定すると、トピック、プッシュ通知、および SMS 配信のパフォーマンスをより正確に把握できます。例えば、NumberOfNotificationsFailed など、Amazon SNS メトリクスの指定のしきい値に達した場合に E メール通知が送信されるよう、アラームを設定することができます。Amazon SNS が CloudWatch に送信するすべてのメトリクスのリストについては、「[Amazon SNS のメトリクス](#)」を参照してください。Amazon SNS プッシュ通知の詳細については、「[Amazon SNS でモバイルプッシュ通知を送信する](#)」を参照してください。

Note

CloudWatch を使用して Amazon SNS トピック用に設定するメトリクスは、自動的に収集され、1 分間隔で CloudWatch にプッシュ通知されます。これらのメトリクスは、CloudWatch ガイドラインを満たすすべてのトピックで収集され、アクティブになります。CloudWatch は、トピックの最後のアクティビティ (つまり、任意の API コール) から最大 6 時間、そのトピックをアクティブと見なします。

CloudWatch でレポートされた Amazon SNS メトリクスに対して料金は発生しません。それらは Amazon SNS サービスの一部として提供されます。

Amazon SNS 用の CloudWatch メトリクス

Amazon SNS のメトリクスは CloudWatch の独自のコマンドインターフェイス (CLI) を使用して、あるいはプログラムによって CloudWatch API を使用してモニタリングできます。次の手順は、AWS マネジメントコンソールを使用してメトリクスにアクセスする方法を示しています。

CloudWatch コンソールを使用してメトリクスを表示するには

1. [CloudWatch コンソール](#)にサインインします。
2. ナビゲーションパネルで [Metrics] を選択します。
3. [All metrics] タブで [SNS] を選択し、次のいずれかのディメンションを選択します。
 - 国/地域、SMS タイプ
 - 電話番号
 - トピックのメトリクス
 - ディメンションの定められていないメトリクス
4. 詳細を表示するには、特定の項目を選択します。例えば、[Topic Metrics] を選択し、[NumberOfMessagesPublished] を選択すると、6 時間の時間範囲における 1 分間の公開済み Amazon SNS メッセージの平均数が表示されます。
5. Amazon SNS の使用状況メトリクスを表示するには、[All metrics] (すべてのメトリクス) タブで [Usage] (使用率) を選択し、対象の Amazon SNS 使用率メトリクス (例: NumberOfMessagesPublishedPerAccount) を選択します。

Amazon SNS メトリクス用の CloudWatch メトリクスを設定する

CloudWatch では、メトリクスのしきい値に到達したときのアラームを設定することもできます。例えば、サンプリング期間内に指定されたしきい値に到達した場合、イベントについて知らせる E メール通知が送信されるように、メトリクス NumberOfNotificationsFailed のアラームを設定できます。

CloudWatch コンソールを使用してアラームを設定するには

1. にサインイン AWS マネジメントコンソール し、<https://console.aws.amazon.com/cloudwatch/>で CloudWatch コンソールを開きます。
2. [Alarms]、[Create Alarm] の順に選択します。これにより、[Create Alarm] ウィザードが起動します。

3. Amazon SNS メトリクスをスクロールして、アラームを設定するメトリクスを見つけます。アラームを設定するメトリクスを選択してから、[Continue] を選択します。
4. [Name]、[Description]、[Threshold]、[Time] のそれぞれにメトリクスの値を入力し、[Continue] を選択します。
5. アラーム状態として [Alarm] を選択します。アラーム状態になったときに CloudWatch から E メールが届くようにするには、既存の Amazon SNS トピックを選択するか、[新しいメールトピックの作成] を選択します。[新しいメールトピックの作成] を選択した場合は、新しいトピックの名前と E メールアドレスを設定できます。このリストは、今後のアラーム用に保存され、ドロップダウンボックスに表示されます。[続行] をクリックしてください。

 Note

[Create New Email Topic] を使用して新しい Amazon SNS トピックを作成する場合、メールアドレスを検証しなければ、そのアドレスで通知を受け取ることができません。E メールは、アラームがアラーム状態になったときにのみ送信されます。アラーム状態になったときに、メールアドレスの検証がまだ完了していない場合は、そのアドレスで通知を受け取ることはできません。

6. この時点で、[Create Alarm] ウィザードで、作成するアラームを確認できます。何らかの変更を行う必要がある場合は、右側にある [Edit] リンクを使用します。希望どおりの設定になったら、[Create Alarm] を選択します。

CloudWatch とアラームの使用方法的詳細については、「[CloudWatch のドキュメント](#)」を参照してください。

Amazon SNS のメトリクス

Amazon SNS は、次のメトリクスを CloudWatch に送信します。

名前空間	メトリクス	説明
AWS/SNS	NumberOfMessagesPublished	Amazon SNS トピックに対して発行されたメッセージ数。 単位: カウント

名前空間	メトリクス	説明
		有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum
AWS/SNS	NumberOfNotificationsDelivered	Amazon SNS トピックからそのトピックにサブスクライブしているエンドポイントに正常に配信されたメッセージ数。 配信の試行が成功するには、エンドポイントのサブスクリプションでメッセージが許可されている必要があります。サブスクリプションでメッセージが許可されるのは、a.) フィルターポリシーがないか、b.) フィルターポリシーに、メッセージに割り当てられた属性と一致する属性が含まれる場合です。サブスクリプションでメッセージが拒否された場合、配信の試行はこのメトリクスに対してカウントされません。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum

名前空間	メトリクス	説明
AWS/SNS	NumberOfNotificationsFailed	Amazon SNS が配信に失敗したメッセージの数。 Amazon SQS、E メール、SMS、またはモバイルプッシュエンドポイントの場合、Amazon SNS がメッセージ配信の試行を停止すると、このメトリクスが 1 増分されます。HTTP または HTTPS エンドポイントの場合、失敗した配信の試行がすべて、初回試行に続く再試行を含め、このメトリクスに対してカウントされます。その他のすべてのエンドポイントの場合、メッセージが配信されないと、カウントが 1 増加します (試行回数には関係ありません)。 サブスクリプションフィルターポリシーによって拒否されたメッセージは、このメトリクスに対してカウントされません。 HTTP エンドポイントの再試行回数をコントロールできます。詳細については、「Amazon SNS メッセージ配信の再試行」を参照してください。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average

名前空間	メトリクス	説明
AWS/SNS	NumberOfNotificationsFilteredOut	サブスクリプションフィルターポリシーによって拒否されたメッセージの数。メッセージの属性がポリシーの属性と一致しない場合、フィルターポリシーによってメッセージは拒否されます。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average
AWS/SNS	NumberOfNotificationsFilteredOut-MessageAttributes	属性ベースのフィルタリングのサブスクリプションフィルターポリシーによって拒否されたメッセージの数。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average

名前空間	メトリクス	説明
AWS/SNS	NumberOfNotificationsFilteredOut-MessageBody	ペイロードベースのフィルタリングのサブスクリプションフィルターポリシーによって拒否されたメッセージの数。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average
AWS/SNS	NumberOfNotificationsFilteredOut-InvalidAttributes	メッセージの属性が無効であるため (例えば属性の JSON 形式が正しくないため)、サブスクリプションフィルターポリシーによって拒否されたメッセージの数。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average

名前空間	メトリクス	説明
AWS/SNS	NumberOfNotificationsFilteredOut-NoMessageAttributes	メッセージに属性がないため、サブスクリプションフィルターポリシーによって拒否されたメッセージの数。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average
AWS/SNS	NumberOfNotificationsFilteredOut-InvalidMessageBody	メッセージ本文がフィルタリングに対して無効であるため (JSON メッセージ本文が無効であるなど)、サブスクリプションフィルターポリシーによって拒否されたメッセージの数。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average
AWS/SNS	NumberOfNotificationsRedrivenToDlq	デッドレターキューに移動されたメッセージの数。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average

名前空間	メトリクス	説明
AWS/SNS	NumberOfNotificationsFailedToRedriveToDlq	デッドレターキューに移動できなかったメッセージの数。 単位: カウント 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Sum、Average
AWS/SNS	PublishSize	発行されたメッセージのサイズ。 単位: バイト 有効なディメンション: アプリケーション、PhoneNumber、プラットフォーム、TopicName 有効な統計: Minimum、Maximum、Average、Count

名前空間	メトリクス	説明
AWS/SNS	SMSThMonthToDateSpentUSD	今月の始めから今日までの SMS メッセージの送信料金。 今月の始めから今日までの料金がアカウントの毎月の SMS 使用限度に近付いたことがわかるように、このメトリクスにアラームを設定できます。Amazon SNS が、SMS メッセージを送信するとこの限度を超えるコストが発生すると判断する場合、数分以内に SMS メッセージの発行を停止します。 SMS の毎月の使用料限度の設定の詳細、または AWS に対して限度の引き上げをリクエストする方法については、「Amazon SNS の SMS メッセージプリファレンスを設定する」を参照してください。 単位: USD 有効なディメンション: なし 有効な統計: Sum
AWS/SNS	SMSSuccessRate	正常な SMS メッセージ配信のレート 単位: カウント 有効なディメンション: PhoneNumber 有効な統計: Sum、Average、Data Samples

Amazon SNS メトリクスのディメンション

Amazon Simple Notification Service は、以下のディメンションを CloudWatch に送信します。

ディメンション	説明
Application	アプリケーションオブジェクトのフィルター。APN や FCM など、サポートされるプッシュ通知サービスの 1 つに登録されているアプリケーションやデバイスを表します。
Application,Platform	アプリケーションおよびプラットフォームオブジェクトのフィルター。プラットフォームオブジェクトは、APN や FCM など、サポートされるプッシュ通知サービスで使用されます。
Country	SMS メッセージの送信先の国またはリージョンのフィルター。国またはリージョンは、ISO 3166-1 alpha-2 コードで表されます。
PhoneNumber	SMS を電話番号に直接公開するときに、電話番号にフィルターをかけます (トピックなし)。
Platform	APN や FCM などのプッシュ通知サービスで使用されるプラットフォームオブジェクトのフィルター。
TopicName	Amazon SNS トピック名のフィルター。
SMSType	SMS メッセージのメッセージタイプのフィルター。プロモーションまたはトランザクションがあります。

Amazon SNS 使用率メトリクス

Amazon Simple Notification Service は、以下の使用状況メトリクスを CloudWatch に送信します。

名前空間	サービス	メトリクス	[リソース]	タイプ	説明
AWS/使用	SNS	ResourceCount	NumberOfMessagesPu	[リソース]	• AWS アカウント全体で Amazon

名前空間	サービス	メトリクス	[リソース]	タイプ	説明
			PublishedPerAccount		SNS トピックに発行されたメッセージの数。 - 単位: なし - 有効な統計: Sum
AWS/使用	SNS	ResourceCount	ApproximateNumberOfTopics	[リソース]	- AWS アカウント全体のトピックのおおよその数。 - 単位: なし - 有効な統計: Average、Minimum、Maximum、Sum
AWS/使用	SNS	ResourceCount	ApproximateNumberOfFilterPolicies	[リソース]	- AWS アカウント全体のフィルターポリシーのおおよその数。 - 単位: なし - 有効な統計: Average、Minimum、Maximum、Sum

名前空間	サービス	メトリクス	[リソース]	タイプ	説明
AWS/使用	SNS	ResourceCount	ApproximateNumberOfPendingSubscriptions	[リソース]	• AWS アカウント全体で保留中のサブスクリプションのおおよその数。 • 単位: なし • 有効な統計: Average、Minimum、Maximum、Sum

名前空間	サービス	メトリクス	[リソース]	タイプ	説明
AWS/使用	SNS	CallCount	- AddPermission - CheckIfPhoneNumberIsOptedOut - CreatePlatformApplication - CreatePlatformEndpoint - ConfirmSubscription - CreateSMSSandboxPhoneNumber - CreateTopic - DeleteEndpoint - DeletePlatformApplication - DeleteSMSSandboxPhoneNumber - DeleteTopic	API	- AWS アカウント全体で選択した Amazon SNS API の API コールの数。 - 末尾のセクションではコンテンツは許可されません。 - 単位: なし - 有効な統計: Sum

名前空間	サービス	メトリクス	[リソース]	タイプ	説明
			• <code>GetEndpointAttributes</code> • <code>GetPlatformApplicationAttributes</code> • <code>GetSMSAttributes</code> • <code>GetSMSSandboxAccountStatus</code> • <code>GetSubscriptionAttributes</code> • <code>GetTopicAttributes</code> • <code>ListEndpointsByPlatformApplication</code> • <code>ListOriginNumbers</code> • <code>ListPhoneNumbersOptedOut</code> • <code>ListPlatformApplications</code>		

名前空間	サービス	メトリクス	[リソース]	タイプ	説明
			- ListSMSSandboxPhoneNumbers - ListSubscriptions - ListSubscriptionsByTopic - ListTagsForResource - ListTopics - OptInPhoneNumber - RemovePermission - SetEndpointAttributes - SetPlatformApplicationAttributes - SetSMSAttributes - SetSubscriptionAttributes - SetTopicAttributes		

名前空間	サービス	メトリクス	[リソース]	タイプ	説明
			- Subscribe - Unsubscribe - UntagResource - VerifySMSSandboxPhoneNumber		

Amazon SNS 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、「コンプライアンス [AWS のサービス 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。

Amazon SNS の耐障害性

Amazon SNS の耐障害性は、AWS リージョン およびアベイラビリティゾーンを中心に展開する AWS グローバルインフラストラクチャを活用することで保証されます。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アベイラビリティゾーン AWS リージョン を提供します。このアーキテクチャにより、アベイラビリティゾーン間でのシームレスで中断のないフェイルオーバーが可能になります。従来のデータセンターインフラストラクチャと比べ、アプリケーションとデータベースの本質的な耐障害性とス

ケーラビリティが向上します。アベイラビリティゾーンを使用すると、Amazon SNS サブスクライバーは可用性と信頼性の向上によるメリットを活用でき、潜在的な中断に対するメッセージ配信のレジリエンスを確保できます。AWS リージョン およびアベイラビリティゾーンの詳細については、[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。

さらに、Amazon SNS トピックのサブスクリプションを、配信再試行とデッドレターキューを使用して設定すると、一時的な障害を自動的に処理でき、メッセージを確実に目的の送信先に配信できます。

Amazon SNS はメッセージフィルタリングとメッセージ属性もサポートしており、特定のユースケースに合わせてレジリエンス戦略を調整して、アプリケーションの全体的な堅牢性を向上できます。

Amazon SNS のインフラストラクチャセキュリティ

マネージドサービスである Amazon SNS は、[「セキュリティ、アイデンティティ、コンプライアンスのベストプラクティス」](#)ドキュメントに記載されている 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 API アクションを使用して、ネットワーク経由で Amazon SNS 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。

IAM プリンシパルに関連付けられているアクセスキー ID とシークレットアクセスキーの両方を使用してリクエストに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。

これらの API アクションは任意のネットワークの場所から呼び出すことができますが、Amazon SNS ではリソースベースのアクセスポリシーがサポートされています。これには送信元 IP アドレスに基づく制限を含めることができます。また、Amazon SNS ポリシーを使用して、特定の Amazon VPC エンドポイントまたは特定の VPC からのアクセスを制御することもできます。これにより効果的に、AWS ネットワーク内の特定の VPC から特定の Amazon SNS トピックへのネットワークアクセスのみが分離されます。詳細については、「[Amazon SNS トピックの発行を特定の VPC エンドポイントのみから制限する](#)」を参照してください。

Amazon SNS のセキュリティベストプラクティス

AWS は、Amazon SNS に多くのセキュリティ機能を提供します。独自のセキュリティポリシーのコンテキストで、これらのセキュリティ機能を確認します。

Note

これらのセキュリティ機能のガイダンスは、一般的ユースケースと実装に適用されます。特定のユースケース、アーキテクチャ、脅威モデルのコンテキストで、これらのベストプラクティスを確認することをお勧めします。

予防的ベストプラクティス

以下に、Amazon SNS の予防的なセキュリティに関するベストプラクティスを示します。

トピック

- [トピックがパブリックアクセス可能でないようにする](#)
- [最小特権アクセスの実装](#)
- [Amazon SNS アクセスを必要とするアプリケーションと AWS サービスに IAM ロールを使用する Amazon SNS](#)
- [サーバー側の暗号化を実装する](#)
- [送信時のデータの暗号化を強制する](#)
- [VPC エンドポイントを使用して Amazon SNS にアクセスすることを検討する](#)
- [サブスクリプションが raw http エンドポイントに配信するように設定されていないことを確認する](#)
- [サブスクリプション解除時に認証を適用する](#)

トピックがパブリックアクセス可能でないようにする

インターネット上の誰かが Amazon SNS トピックを読み書きできるように明示的に要求しない限り、トピックにパブリックにアクセスできないようにする必要があります (世界中のすべてのユーザーまたは認証された AWS ユーザーがアクセス可能)。

- Principal を "" に設定してポリシーを作成しないでください。
- ワイルドカード (*) を使用しないでください。代わりに、特定のユーザーに名前を付けます。

最小特権アクセスの実装

アクセス権限を付与する場合、アクセス権限を受け取るユーザー、アクセス許可の対象となるトピック、およびこれらのトピックに対して許可する特定の API アクションを決定します。最小権限の原則を実装することは、セキュリティリスクを軽減するために重要です。また、エラーや悪意のある意図による悪影響を減らすのにも役立ちます。

最小特権を付与するスタンダードのセキュリティアドバイスに従ってください。つまり、特定のタスクの実行に必要なアクセス権限のみを付与します。ユーザーアクセスに関連するセキュリティポリシーを組み合わせて使用することで、最小権限を実装できます。

Amazon SNS では、発行者と受信者のモデルが使用され、次の 3 種類のユーザーアカウントアクセスが必要です。

- 管理者 - トピックの作成、変更、削除にアクセスします。管理者は、トピックポリシーも制御します。
- 発行者 - トピックへのメッセージ送信のアクセス権限を持ちます。
- 受信者 - トピックへの登録のアクセス権限を持ちます。

詳細については、次のセクションを参照してください。

- [Amazon SNS での Identity and Access Management](#)
- [Amazon SNS API のアクセス許可: アクションとリソースのリファレンス](#)

Amazon SNS アクセスを必要とするアプリケーションと AWS サービスに IAM ロールを使用する Amazon SNS

Amazon EC2 などのアプリケーションまたは AWS サービスが Amazon SNS トピックにアクセスするには、AWS API リクエストで有効な AWS 認証情報を使用する必要があります。これらの認証情報は自動的にローテーションされないため、AWS 認証情報をアプリケーションまたは EC2 インスタンスに直接保存しないでください。

代わりに、IAM ロールを使用して、Amazon SNS にアクセスする必要があるアプリケーションまたはサービスの一時的な認証情報を管理することをおすすめします。ロールを使用するとき、EC2 インスタンスまたは AWS のサービス (AWS Lambda など) に長期の認証情報 (ユーザー名、パスワード、アクセスキーなど) を配布する必要はありません。代わりに、ロールは、アプリケーションが他の AWS リソースを呼び出すときに使用できる一時的なアクセス許可を提供します。

詳細については、「IAM ユーザーガイド」の「[IAM ロール](#)」および「[ロールの一般的なシナリオ: ユーザー、アプリケーション、およびサービス](#)」を参照してください。

サーバー側の暗号化を実装する

データ漏洩の問題を軽減するには、保存時の暗号化を使用して、メッセージを保存する場所とは別の場所に保存されているキーを使用してメッセージを暗号化します。サーバー側の暗号化 (SSE) は、保存時のデータ暗号化を提供します。Amazon SNS は、データを保存するときにメッセージレベルで暗号化し、アクセスするとメッセージを復号します。SSE は、管理されるキーを使用します。AWS Key Management Service。リクエストが認証され、お客様がアクセス権限を持っていれば、トピックが暗号化されているかどうかに関係なく同じ方法でアクセスできます。

詳細については、「[サーバー側の暗号化を使用した Amazon SNS データの保護](#)」および「[Amazon SNS 暗号化キーとコストの管理](#)」を参照してください。

送信時のデータの暗号化を強制する

HTTP を使用して送信中に暗号化されていないメッセージを公開することは可能ですが、お勧めできません。ただし、を使用してトピックを保管中に暗号化する場合は AWS KMS、メッセージの発行に HTTPS を使用して、保管中と転送中の両方の暗号化を確保する必要があります。トピックは HTTP メッセージを自動的に拒否しませんが、セキュリティ標準を維持するには HTTPS を使用する必要があります。

AWS では、HTTP の代わりに HTTPS を使用することをお勧めします。HTTPS を使用すると、SNS トピック自体が暗号化されていなくても、メッセージは送信中に自動的に暗号化されます。HTTPS を使用しない場合、ネットワークベースの攻撃者は、中間者などの攻撃を使用して、ネットワークトラフィックを傍受したり操作することができます。

HTTPS 経由の暗号化された接続のみを実行するには、[aws:SecureTransport](#) 条件を、暗号化されていない SNS トピックに添付されている IAM ポリシーに追加します。これにより、メッセージ発行者は HTTP ではなく HTTPS を使用することになります。次の例のポリシーをガイドとして使用できます。

JSON

```
{
  "Id": "ExamplePolicy",
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
    "Sid": "AllowPublishThroughSSLOnly",
    "Action": "SNS:Publish",
    "Effect": "Deny",
    "Resource": [
        "arn:aws:sns:us-east-1:111122223333:test-topic"
    ],
    "Condition": {
        "Bool": {
            "aws:SecureTransport": "false"
        }
    },
    "Principal": "*"
}
]
```

VPC エンドポイントを使用して Amazon SNS にアクセスすることを検討する

操作できる必要があるが、インターネットに絶対に公開してはならないトピックがある場合は、VPC エンドポイントを使用して、特定の VPC 内のホストのみにトピックアクセスを制限します。トピックポリシーを使用して、特定の VPC エンドポイントまたは特定の VPC からのトピックへのアクセスを制御できます。

Amazon SNS の VPC エンドポイントには、メッセージへのアクセスを制御するために、2 通りの方法が用意されています。

- 特定の VPC エンドポイントを通じて許可されるリクエスト、ユーザー、またはグループを管理できます。
- トピックポリシーを使用して、どの VPC または VPC エンドポイントがトピックにアクセスできるかを制御できます。

詳細については、「[エンドポイントの作成](#)」および「[Amazon SNS 用の VPC エンドポイントポリシーを作成する](#)」を参照してください。

サブスクリプションが raw http エンドポイントに配信するように設定されていないことを確認する

サブスクリプションは raw http エンドポイントに配信する設定をしないでください。エンドポイントドメイン名に配信するサブスクリプションが必ずなければいけません。例えば、エンドポイント

に配信するように構成されたサブスクリプションである `http://1.2.3.4/my-path` は `http://my.domain.name/my-path` に変更する必要があります。

サブスクリプション解除時に認証を適用する

E メールや SMS のサブスクリプション解除を簡単に行う場合など、認証されていないサブスクリプション解除を許可する必要がある場合を除き、トピックからのサブスクリプション解除には認証を適用する必要があります。これは、[最小特権アクセスコントロールの推奨事項](#)と一致しています。

サブスクリプションの確認中に `AuthenticateOnUnsubscribe` を `True` に設定できます。Amazon SNS サブスクリプションを確認するときに `AuthenticateOnUnsubscribe` フラグを `True` に設定しないと、サブスクリプション解除のリクエストが認証されていない場合でも成功してしまう可能性があります。詳細については、「Amazon SNS API リファレンス」の「[ConfirmSubscription](#)」または [Amazon Q Detector Library の Python の例](#) を参照してください。

たとえば、を使用して E メールサブスクリプションを確認するには AWS CLI、E メール通知の「サブスクリプションの確認」テキストからリンクをコピーします。この URL は、次の AWS CLI コマンドを呼び出すために必要な情報を提供します。

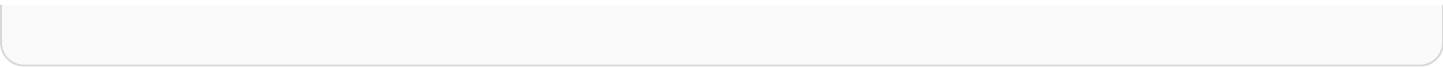
```
aws sns confirm-subscription --region us-west-2 \
    --topic-arn sns-topic-arn \
    --token token-from-subscribe-notification \
    --authenticate-on-unsubscribe true
```

コードの説明は以下のとおりです。

- `aws-region` は、トピックがある AWS リージョンです。これは、トピック ARN でも使用できます。
- `sns-topic-arn` はトピックの ARN です。これは、サブスクリプション確認 URL 内の「TopicArn=」と「&Token」の間にあるテキストです。
- `token-from-subscribe-notification` は、サブスクリプション確認 URL 内の「Token=」と「&Endpoint」の間にある UUID 文字列です。

URL の例を次に示します。

```
https://sns.us-west-2.amazonaws.com/confirmation.html?TopicArn=arn:aws:sns:us-west-2:123456789012:sns-topic&Token=a1b2c3d4e5f6789012345678901234567890abcdef1234567890abcdef1234567890abcdef1234567890
```



AWS X-Ray を使用した Amazon SNS トピックのトラブルシューティング

AWS X-Ray は、アプリケーションが処理するリクエストに関するデータを収集し、データの表示とフィルター処理を可能にして、潜在的な問題や最適化の機会を識別できるようにします。アプリケーションに対するトレース対象のリクエストの場合、リクエスト、レスポンス、およびアプリケーションがダウンストリーム AWS リソース、マイクロサービス、データベース、HTTP ウェブ API に対して行う呼び出しの詳細な情報を表示できます。

Amazon SNS で X-Ray を使用して、アプリケーションを通過するメッセージをトレースおよび分析できます。AWS マネジメントコンソールを使用して、Amazon SNS およびアプリケーションが使用する他のサービス間の接続マップを表示できます。コンソールを使用して、平均レイテンシーや障害発生率などのメトリクスを表示することもできます。詳細については、『AWS X-Ray デベロッパーガイド』の「[Amazon SNS と AWS X-Ray](#)」を参照してください。

Amazon SNS でのアクティブトレース

を使用して AWS X-Ray、Amazon SNS トピックを [Amazon Data Firehose](#)、[AWS Lambda](#)、[Amazon SQS](#)、および HTTP/S エンドポイントサブスクリプションに渡すユーザーリクエストを追跡および分析します。

X-Ray では、各リクエストのエンドツーエンドのビューを取得でき、次のことが可能になります。

- Amazon SNS トピックを呼び出しているものは何か、およびサブスクリプションのダウンストリームとなるサービスを特定します。
- 次のようなレイテンシーを分析します。
 - 処理前に Amazon SNS トピックで費やされた時間。
 - サブスクライブされた各エンドポイントの配信時間。

Important

多数のサブスクリプションがある Amazon SNS トピックはサイズ制限に達し、完全にトレースされない場合があります。トレースドキュメントのサイズ制限の詳細については、AWS「全般のリファレンス」の「[X 線サービスクォータ](#)」を参照してください。

すでにトレースされているサービスから Amazon SNS API を呼び出すと、Amazon SNS は、API で X-Ray トレースが有効になっていなくてもトレースをパススルーします。

Amazon SNS は、標準トピックと FIFO トピックの両方で X-Ray トレースをサポートしています。Amazon SNS トピックの X-Ray を有効にするには、[Amazon SNS コンソール](#)、[Amazon SNS SetTopicAttributes API](#)、[Amazon Simple 通知サービス CLI リファレンス](#)、または [CloudFormation](#) を使用します。

Amazon SNS で X-Ray を使用方法の詳細については、「AWS X-Ray デベロッパーガイド」の「[Amazon SNS と AWS X-Ray](#)」を参照してください。

アクティブトレースのアクセス許可

Amazon SNS コンソールを使用する場合、Amazon SNS は Amazon SNS トピックが X-Ray を呼び出すために必要なアクセス許可を作成しようとします。Amazon SNS コンソールを使用するための十分なアクセス許可がない場合、この試行は拒否される可能性があります。詳細については、「[Amazon SNS での Identity and Access Management](#)」および「[Amazon SNS アクセスコントロールのケース例](#)」を参照してください。

CLI を使用するときには、アクセス許可を手動で設定する必要があります。これらのアクセス許可は、リソースポリシーを使用して設定されます。X-Ray で必要なアクセス許可を使用する方法の詳細については、「[Amazon SNS と AWS X-Ray](#)」を参照してください。

AWS コンソールを使用して Amazon SNS トピックでアクティブなトレースを有効にする

Amazon SNS トピックでアクティブトレースが有効になると、トレース ID を読み取り、トレース ID に基づいてデータをカスタマーに送信し、そのトレース ID をダウンストリームのサービスに伝播します。

1. [Amazon SNS コンソール](#)にサインインします。
2. トピックを選択するか、新しいトピックを作成できます。トピックの作成の詳細については、「[Amazon SNS のトピックの作成](#)」を参照してください。
3. [トピックの作成] ページの [詳細] セクションで、[FIFO] または [標準] を選択します。
 - a. トピックの名前を入力します。
 - b. (オプション) トピックの表示名を入力します。
4. [Active tracing] (アクティブトレース) を展開し、[Use active tracing] (アクティブトレースを使用する) を選択します。

Amazon SNS トピックで X-Ray を有効にすると、[X-Ray サービスマップ](#)を使用して、トピックのエンドツーエンドのトレースとサービス マップを表示できます。

AWS SDK を使用して Amazon SNS トピックでアクティブトレースを有効にする

次のコード例は、AWS SDK for Java を使用して Amazon SNS トピックでアクティブトレースを有効にする方法を示しています。

```
public static void enableActiveTracing(SnsClient snsClient, String topicArn) {

    try {

        SetTopicAttributesRequest request = SetTopicAttributesRequest.builder()
            .attributeName("TracingConfig")
            .attributeValue("Active")
            .topicArn(topicArn)
            .build();

        SetTopicAttributesResponse result = snsClient.setTopicAttributes(request);
        System.out.println("\n\nStatus was " +
            result.sdkHttpResponse().statusCode() + "\n\nTopic " + request.topicArn()
            + " updated " + request.attributeName() + " to " +
            request.attributeValue());

    } catch (SnsException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
    }
}
```

AWS CLI を使用して Amazon SNS トピックでアクティブなトレースを有効にする

次のコード例は、AWS CLI を使用して Amazon SNS トピックでアクティブトレースを有効にする方法を示しています。

```
aws sns set-topic-attributes \
  --topic-arn arn:aws:sns:us-west-2:123456789012:MyTopic \
  --attribute-name TracingConfig \
  --attribute-value Active
```

を使用して Amazon SNS トピックでアクティブトレースを有効にする CloudFormation

次の CloudFormation スタックは、Amazon SNS トピックでアクティブトレースを有効にする方法を示しています。

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  MyTopicResource:
    Type: 'AWS::SNS::Topic'
    Properties:
      TopicName: 'MyTopic'
      TracingConfig: 'Active'
```

トピックでアクティブトレースが有効になっていることを確認する

Amazon SNS コンソールを使用して、トピックでアクティブトレースが有効になっているかどうか、またはリソースポリシーの追加に失敗したかどうかを確認できます。

1. [Amazon SNS コンソール](#)にサインインします。
2. 左のナビゲーションペインで、[トピック] を選択します。
3. [Topics] (トピック) ページで、トピックを選択します。
4. [Integrations] (統合) タブを開きます。

アクティブトレースが有効になっている場合、[Active] (有効化) アイコンが表示されます。

5. アクティブトレースを有効にしているのにリソースポリシーが追加されていない場合は、[Create policy] (ポリシーの作成) を選択して必要なアクセス許可を追加します。

[Amazon SNS](#) > [Topics](#) > SampleTopic

SampleTopic

[Edit](#)[Delete](#)[Publish message](#)

Details

Name	Display name
SampleTopic	-
ARN	Topic owner
arn:aws:sns:us-east-1:242420583777:DeliveryRequest	123456789123
Type	
Standard	

[Policy](#) | [Delivery retry policy \(HTTP/S\)](#) | [Delivery status logging](#) | [Encryption](#) | [Integrations](#) >

AWS X-Ray active tracing

**Active tracing may require additional permission.**

We couldn't find an AWS X-Ray resource policy that allows Amazon SNS to send trace data. To create that policy now, choose "Create policy".

[Create policy](#)

Active tracing

Active

Resource policy

Not found

アクティブトレースのテスト

1. [Amazon SNS コンソール](#)にサインインします。
2. Amazon SNS トピックを作成します。この操作の詳細については、「[を使用してトピックを作成するには AWS マネジメントコンソール](#)」を参照してください。
3. [Active tracing] (アクティブトレース) を展開し、[Use active tracing] (アクティブトレースを使用する) を選択します。
4. Amazon SNS トピックにメッセージを発行します。この操作の詳細については、「[を使用して Amazon SNS トピックにメッセージを発行するには AWS マネジメントコンソール](#)」を参照してください。
5. [X-Ray サービスマップ](#)を使用して、トピックのエンドツーエンドトレースとサービスマップを表示します。



Amazon SNS のドキュメント履歴

以下の表は、「Amazon Simple Notification Service デベロッパーガイド」の最近の変更点をまとめています。

サービス機能は、サービスが利用可能な AWS リージョンに段階的にロールアウトされることがあります。このドキュメントは、最初のリリースのためにのみ更新されています。リージョンの可用性に関する情報を提供したり、その後のリージョンのロールアウトを発表したりすることはありません。サービス機能のリージョンの可用性と更新に関する通知のサブスクライブについては、[「の最新情報 AWS」](#)を参照してください。

変更	説明	日付
デュアルスタック (IPv4 と IPv6) エンドポイントのサポートを追加	Amazon SNS は API リクエストで IPv6 をサポートするようになりました。これにより、クライアントは IPv4、IPv6、またはデュアルスタックを使用してパブリックエンドポイントに接続できます。	2025 年 4 月 3 日
Amazon SNS で、Amazon SNS FIFO トピック用に FifoThroughputScope 属性を追加	Amazon SNS は、FIFO トピックに適用するスループットのクォータと重複排除動作を指定する FifoThroughputScope 属性をサポートしています。有効な値は Topic または MessageGroup です。	2025 年 1 月 21 日
AmazonSNSFullAccess および AmazonSNSReadOnlyAccess のマネージドポリシーの更新	Amazon SNS では、Amazon SNSFullAccess および AmazonSNSReadOnlyAccess のマネージドポリシーに新しいアクセス許可が追加されました。これにより、AWS マネジメントコンソールから	2024 年 9 月 24 日

Amazon SNS に追加でアクセスできるようになります。

[SMS メッセージの配信のための Amazon SNS との統合 AWS End User Messaging SMS](#)

Amazon SNS は、設定や Amazon SNS で使用されるグローバル SMS ネットワークを変更することなく、SMS リソース管理、双方向メッセージング、詳細なリソースアクセス許可、国ブロックルール、すべての AWS SMS AWS メッセージングの一括請求などの新機能をサポートしています Amazon SNS。

2024 年 9 月 24 日

[FIFO トピックのカナダ西部 \(カルガリー\) サポート](#)

Amazon SNS は、カナダ西部 (カルガリー) で FIFO トピックをサポートします。

2024 年 3 月 28 日

[5 つの新しいリージョンでの Amazon SNS SMS サポート](#)

アジアパシフィック (ハイデラバード)、アジアパシフィック (メルボルン)、中東 (アラブ首長国連邦)、欧州 (チューリッヒ)、欧州 (スペイン) における Amazon SNS の SMS サポートが追加されました。

2024 年 2 月 8 日

[Firebase Cloud Messaging \(FCM\) HTTP v1 のサポート](#)

Amazon SNS は FCM v1 認証情報をサポートします。

2024 年 1 月 18 日

[アジアパシフィック \(ジャカルタ\) では、Amazon SNS は SMS をサポート](#)

アジアパシフィック (ジャカルタ) では、Amazon SNS は SMS メッセージングをサポートしています。

2023 年 12 月 14 日

[CloudFormation Amazon SNS トピック DeliveryStatusLogging の設定のサポート](#)

CloudFormation サポートは、Amazon SNS トピックの作成または更新 DeliveryStatusLogging 中の の設定に使用できます。

2023 年 12 月 7 日

[新しいメッセージフィルタリング演算子が追加されました](#)

Amazon SNS メッセージをフィルタリングするときに、サフィックスマッチング、equals-ignore case、および OR 演算子を使用できるようになりました。

2023 年 11 月 16 日

[メッセージのアーカイブとリプレイに関するサポートが追加されました](#)

トピック所有者はメッセージを最大 365 日間トピックにアーカイブできます。トピックサブスクライバーは、アーカイブされたメッセージをサブスクライブされたエンドポイントにリプレイして、ダウンストリームアプリケーションの障害によるメッセージを回復したり、既存のアプリケーションの状態を複製したりできます。

2023 年 10 月 26 日

[標準キューを FIFO トピックにサブスクライブするためのサポートを追加](#)

Amazon SQS FIFO キューまたは標準キューを Amazon SNS FIFO トピックにサブスクライブできます。Amazon SQS FIFO キューに限り、メッセージが順番に受信され、重複がないことが保証されます。

2023 年 9 月 14 日

[イスラエル \(テルアビブ\) 向けの SMS サポートが追加されました](#)

Amazon SNS SMS がイスラエル (テルアビブ) リージョンでサポートされるようになりました。

2023 年 8 月 28 日

[FIFO トピックに追加された X-Ray アクティブトレースのサポート](#)

以前は Amazon SNS 標準トピックでのみサポートされていましたが、FIFO トピックを介して Amazon Data Firehose、AWS Lambda Amazon SQS、および HTTP/S エンドポイントサブスクリプションに移動するユーザーリクエストをトレースして分析する AWS X-Ray になりました。

2023 年 5 月 31 日

[Content-Type ヘッダーのサポート強化](#)

リクエストポリシーに Content-Type ヘッダーを設定して、通知のメディアタイプを指定できます。

2023 年 3 月 23 日

[アクティブトレースのサポートが追加されました](#)

AWS X-Ray は、Amazon SNS 標準トピックを介して Amazon Data Firehose、Amazon SQS AWS Lambda、HTTP/S エンドポイントサブスクリプションに移動するユーザーリクエストをトレースして分析します。

2023 年 2 月 8 日

[シンガポールの送信者 ID 登録](#)

シンガポールで送信者 ID を登録するための手順が追加されました。

2023 年 1 月 10 日

[ペイロードベースのメッセージフィルタリング](#)

ペイロードベースのフィルタリングにより、メッセージペイロードに基づいてメッセージをフィルタリングできるため、不要なデータの処理に伴うコストを回避できます。

2022 年 11 月 22 日

[Amazon SNS メッセージ署名用に SHA256 ハッシュアルゴリズムが追加されました](#)

Amazon SNS メッセージ署名を使用する際の SHA256 ハッシュアルゴリズムのサポートが追加されました。

2022 年 9 月 15 日

[SMS メッセージングにリージョンが追加されました](#)

Amazon SNS は、アフリカ (ケープタウン)、アジアパシフィック (大阪)、欧州 (ミラノ)、AWS GovCloud (米国東部) の各リージョンで SMS メッセージングをサポートしています。

2022 年 9 月 9 日

[メッセージデータ保護サポートが追加されました](#)

メッセージデータ保護は、データ保護ポリシーを使用してアプリケーションまたは AWS サービス間を移動する機密情報を監査およびブロックすることで、Amazon SNS トピックに発行されるデータを保護します。

2022 年 9 月 8 日

[通話料無料番号の新規登録プロセス](#)

通話料無料の電話番号 (TFN) を使用して Amazon SNS メッセージを米国の受信者に送信するためのサポートを追加しました。

2022 年 8 月 1 日

[属性ベースのアクセスコントロール \(ABAC\) のサポート](#)

Publish および PublishBatch を含め、API アクションの属性ベースのアクセスコントロール (ABAC) のサポートが追加されました。ABAC は、IAM ユーザーやロールなどの IAM リソース、および Amazon SNS トピックなどの AWS リソースにアタッチできるタグに基づいてアクセス許可を定義する認可戦略であり、アクセス許可の管理を簡素化します。

2022 年 1 月 10 日

[プッシュ通知用の Apple トークンベース認証のサポート](#)

アプリの開発者であることを識別する情報を提供することで、Amazon SNS に iOS または macOS アプリへのプッシュ通知の送信を許可できます。

2021 年 10 月 28 日

[SMS メッセージの新しい送信者は SMS サンドボックスに配置される](#)

SMS サンドボックスは、不正使用や悪用を防止し、送信者としての評判を保つのに役立ちます。AWS アカウントが SMS サンドボックスにある間は、検証済みの送信先電話番号にのみ SMS メッセージを送信できます。

2021 年 6 月 1 日

[SMS メッセージの新しい送信者は SMS サンドボックスに配置される](#)

SMS サンドボックスは、不正使用や悪用を防止し、送信者としての評判を保つのに役立ちます。AWS アカウントが SMS サンドボックスにある間は、検証済みの送信先電話番号にのみ SMS メッセージを送信できます。

2021 年 6 月 1 日

[インドの受信者に SMS メッセージを送信する新しい属性](#)

2 つの新しい属性、エンティティ ID およびテンプレート ID が、インドの受信者に SMS メッセージを送信するために必要になりました。

2021 年 4 月 22 日

[メッセージフィルタリング演算子の更新](#)

新しい演算子、cidr は、メッセージ送信元 IP アドレスとサブネットのマッチングに使用できます。また、属性キーがないかどうかをチェックし、属性文字列のマッチングのための anything-but 演算子でプレフィックスを使用します。

2021 年 4 月 7 日

[米国の送信先の P2P ロングコードのサポート終了](#)

2021 年 6 月 1 日より、米国のテレコムプロバイダーは、米国の送信先への Application-to-Person (A2P) 通信に Person-to-Person (P2P) ロングコードの使用をサポートしていません。代わりに、短いコード、通話料無料の番号、または 10DLC と呼ばれる新しいタイプの発信番号を使用できます。

2021 年 2 月 16 日

[1 分間 Amazon CloudWatch メトリクスのサポート](#)

Amazon SNS の 1 分間の CloudWatch メトリクスがすべての AWS リージョンで利用可能になりました。Amazon SNS

2021 年 1 月 28 日

[エンドポイントのサポート](#)

SNS トピックへ Firehose 配信ストリームをサブスクライブできます。これにより、Amazon Simple Storage Service (Amazon S3) バケット、Amazon Redshift テーブル、Amazon OpenSearch Service (OpenSearch Service) などのアーカイブと分析エンドポイントへ通知を送信することができます。

2021 年 1 月 12 日

[送信元番号が利用可能](#)

テキストメッセージ (SMS) の送信時に送信元番号を使用できます。

2020 年 10 月 23 日

[Amazon SNS FIFO トピック のサポート](#)

データの一貫性をほぼリアルタイムで必要とする分散アプリケーションを統合するには、Amazon SQS FIFO キューで Amazon SNS 先入れ先出し (FIFO) トピックを使用できます。

2020 年 10 月 22 日

[Amazon SNS Extended Client Library for Java が利用可能](#)

このライブラリを使用して、容量の大きい Amazon SNS メッセージを発行できます。

2020 年 8 月 25 日

[SSE は、中国リージョンで使 用可能](#)

Amazon SNS のサーバー側の暗号化 (SSE) が、中国リージョンで利用可能です。

2020 年 1 月 20 日

DLQ を使用した配信不能メッセージのキャプチャのサポート	Amazon SQS デッドレターキュー (DLQ) を使用して、Amazon SNS サブスクリプションで配信不能メッセージをキャプチャできます。	2019 年 11 月 14 日
カスタム APN ヘッダーの値の指定のサポート	APN ヘッダーのカスタム値を指定できます。	2019 年 10 月 18 日
APN の 'apns-push-type' ヘッダーフィールドのサポート	APN を介して送信されるモバイル通知に、apns-push-type ヘッダーフィールドを使用できます。	2019 年 9 月 10 日
を使用したトピックのトラブルシューティングのサポート AWS X-Ray	X-Ray を使用して、SNS トピックを通過するメッセージのトラブルシューティングを行うことができます。	2019 年 7 月 24 日
'exists' 演算子を使用した属性キーマッチングのサポート	フィルターポリシーにキーがリストされている属性が受信メッセージにあるかどうかをチェックするには、exists 演算子を使用できます。	2019 年 7 月 5 日
複数の数値の「以外」のマッチングのサポート	複数の文字列に加えて、Amazon SNS では複数の数値の「以外」のマッチングが可能です。	2019 年 7 月 5 日
Amazon SNS リリースノートが、RSS フィードとして利用できるようになりました	このページのタイトルに続いて ([ドキュメント履歴]) を選択し、[RSS] を選択します。	2019 年 6 月 22 日

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。