デベロッパーガイド

Amazon Simple Queue Service

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Simple Queue Service: デベロッパーガイド

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは、Amazon のものではない製品またはサービスにも関連して、お客様に混乱を招いたり Amazon の信用を傷つけたり失わせたりするいかなる形においても使用することはできません。Amazon が所有していない他のすべての商標は、それぞれの所有者の所有物であり、Amazon と提携、接続、または後援されている場合とされていない場合があります。

Table of Contents

Amazon SQSとは? 1
- Amazon SQSを使用する利点 1
- 基本的なアーキテクチャ 1
  - 分散キュー 2
  - メッセージのライフサイクル 2
- Amazon SQS、Amazon MQ、Amazon SNS 間の違い 4

入門 6
- 設定 6
  - ステップ 1: AWS アカウント と IAM ユーザーを作成する 6
  - ステップ 2: プログラム的なアクセス権を付与する 8
  - ステップ 3: コード例を使用する準備を整える 10
  - 次のステップ 11
- Amazon SQS コンソールを理解する 11
- キューのタイプ 12
  - Amazon SQS でのリクエスト/レスポンスシステムの実装 15
- 標準キューの作成 16
  - キューの作成 16
  - 標準キューを使用したメッセージの送信 18
- FIFO キューを作成する 19
  - キューを作成する 19
  - FIFO キューを使用したメッセージの送信 22
- 一般的なタスク 22

キューの管理 24
- キューの編集 24
- メッセージの受信および削除 24
- キューが空であることを確認する 26
- キューの削除 27
- キューのクリア 28

標準キュー 29
- Amazon SQS の少なくとも 1 回の配信 30
- キューとメッセージの識別子 30
  - 標準キューの識別子 30

FIFO キュー 32
- FIFO キューの主要な用語 33

FIFO配信ロジック ... 34
メッセージの送信 ... 34
メッセージの受信 ... 35
複数回の再試行 ... 36
FIFO の動作に関するその他の注意事項 ... 37
理解を深めるための例 ... 37
1 回のみ処理 ... 38
標準キューから FIFO キューへの移行 ... 38
FIFO キューと Lambda の同時実行動作 ... 40
FIFO キューのメッセージのグループ化 ... 40
FIFO キューでの Lambda の同時実行 ... 41
ユースケースの例 ... 41
FIFO キューの高スループット ... 41
ユースケース ... 42
パーティションとデータ分散 ... 43
FIFO キューの高スループットの有効化 ... 45
キューとメッセージの識別子 ... 46
FIFO キューの識別子 ... 30
FIFOキューのその他の識別子 ... 48
クォータ ... 49
FIFO キューのクォータ ... 49
Amazon SQS のクォータ ... 49
標準キューのクォータ ... 50
メッセージのクォータ ... 52
ポリシーのクォータ ... 59
特徴と機能 ... 60
デッドレターキュー ... 60
デッドレターキューのポリシーの使用 ... 61
デッドレターキューのメッセージ保持期間を理解する ... 61
デッドレターキューを設定する ... 61
デッドレターキューを設定します。 ... 62
CloudTrail の更新とアクセス許可の要件 ... 72
Amazon CloudWatch を使用したデッドレターキューのアラームの作成 ... 76
Amazon SQS のメッセージメタデータ ... 76
メッセージ属性 ... 77
メッセージシステム属性 ... 81

メッセージの処理に必要なリソース 81
リストキューのページネーション 82
コスト配分タグ 82
ショートポーリングとロングポーリング 83
ショートポーリングを使用したメッセージの処理 84
ロングポーリングを使用したメッセージの使用 85
ロングポーリングとショートポーリングの違い 85
可視性タイムアウト 86
可視性タイムアウトのユースケース 86
可視性タイムアウトの設定と調整 87
インフライトメッセージとクォータ 87
標準キューと FIFO キューの可視性タイムアウトについて 88
失敗の処理 88
可視性タイムアウトの変更と終了 88
ベストプラクティス 89
公平キュー 89
FIFO キューとの違い 91
公平キューの使用 92
公平キュー CloudWatch メトリクス 92
遅延キュー 93
一時キュー 94
仮想キュー 94
リクエスト–レスポンスメッセージングパターン(仮想キュー) 96
シナリオ例:ログインリクエストの処理 97
キューをクリーンアップする 98
メッセージタイマー 99
EventBridge Pipes へのアクセス 100
大量のメッセージの管理 101
Java 用拡張クライアントライブラリの使用 102
Python 用拡張クライアントライブラリの使用 108
Amazon SQSの設定 112
Amazon SQS 用 ABAC 112
ABAC とは 112
Amazon SQS で ABAC を使用すべき理由は何ですか。 113
アクセス制御のタグ付け 114
IAM ユーザーと Amazon SQS キューの作成 114

属性ベースのアクセス制御のテスト 118
キューパラメータの設定 119
アクセスポリシーの設定 121
キュー対するSSE-SQSを設定 122
キューにSSE-KMSを設定する 123
キューにタグを設定する 124
トピックへキューをサブスクライブする 125
クロスアカウントサブスクリプション 126
クロスリージョンサブスクリプション 127
Lambdaトリガーの設定 128
前提条件 128
EventBridge使用した通知の自動化 130
メッセージ属性 130
ベストプラクティス 132
エラー処理および問題ありのメッセージ 132
Amazon SQS でのリクエストエラーの処理 132
問題ありのメッセージのキャプチャ 133
Amazon SQS でのデッドレターキュー保持の設定 133
メッセージ重複排除とグループ化 133
Amazon SQS での一貫性のないメッセージ処理の回避 134
メッセージ重複排除ID の使用 134
メッセージグループ ID の使用 137
受信リクエスト試行 ID の使用 138
メッセージ処理とタイミング 139
Amazon SQS でのタイムリーな方法でのメッセージ処理 139
Amazon SQS でのロングポーリングの設定 140
Amazon SQS での適切なポーリングモードの使用 141
Java SDKの例 142
サーバーサイドの暗号化の使用 142
既存のキューに SSE を追加 142
キューのSSEの無効化 143
SSEを使用してキューを作成する 144
SSE属性の取得 145
タグを設定する 145
タグを一覧表にする 145
タグの追加または更新するには 146

タグの削除 ... 146
メッセージ属性の送信 ... 147
属性の定義 ... 147
属性を含むメッセージの送信 ... 149
API の使用 ... 150
JSON プロトコルを使用したクエリ API AWS リクエストの実行 ... 151
エンドポイントの構築 ... 151
POST リクエストの作成 ... 152
Amazon SQS JSON API レスポンスの解釈 ... 153
Amazon SQS AWS JSON プロトコルFAQs ... 154
クエリプロトコルを使用した AWS クエリ API リクエストの実行 ... 158
エンドポイントの構築 ... 158
GETリクエストの作成 ... 159
POST リクエストの作成 ... 152
Amazon SQS の XML API レスポンスの解釈 ... 160
リクエストの認証 ... 162
HMAC-SHA による基本的な認証のプロセス ... 162
パート 1: ユーザーからのリクエスト ... 164
パート 2: からのレスポンス AWS ... 165
バッチアクション ... 165
メッセージアクションのバッチ処理 ... 166
Amazon SQS でのクライアント側のバッファリングとリクエストのバッチ処理の有効化 ... 167
Amazon SQS での水平スケーリングとアクションのバッチ処理を使用したスループットの向上 ... 178
AWS SDKs の使用 ... 190
JMS の使用 ... 192
前提条件 ... 192
Java Messaging Library の使用 ... 193
JMS接続の作成 ... 194
Amazon SQSキューを作成する ... 195
同期的なメッセージの送信 ... 196
同期的なメッセージの受信 ... 197
非同期的なメッセージの受信 ... 199
クライアント確認モードの使用 ... 200
順不同確認モードの使用 ... 201
JMS クライアントを他の Amazon SQS クライアントで使用する ... 202

標準キューで JMS を使用するための実用的な Java の例 203
ExampleConfiguration.java 203
TextMessageSender.java 206
SyncMessageReceiver.java 208
AsyncMessageReceiver.java 210
SyncMessageReceiverClientAcknowledge.java 212
SyncMessageReceiverUnorderedAcknowledge.java 215
SpringExampleConfiguration.xml 219
SpringExample.java 220
ExampleCommon.java 223
サポートされているJMS 1.1実装 224
サポートされている共通インターフェース 224
サポートされているメッセージタイプ 225
サポートされているメッセージ確認モード 225
JMS定義ヘッダーと予約プロパティ 225
チュートリアル 227
を使用した Amazon SQS キューの作成 AWS CloudFormation 227
VPCからのメッセージの送信 229
ステップ 1: Amazon EC2 キーペアを作成する 230
ステップ 2: AWS リソースを作成する 230
ステップ 3: EC2 インスタンスがパブリックアクセス可能ではないことを確認する 231
ステップ4: Amazon SQSの Amazon VPC エンドポイントを作成する 233
ステップ5:Amazon SQSキューにメッセージを送信する 234
コードの例 236
基本 250
Hello Amazon SQS 251
アクション 263
シナリオ 428
Slack メッセージングアプリケーションを作成します。 429
メッセンジャーアプリケーションを作成する 430
Amazon Textract エクスプローラーアプリケーションを作成する 431
FIFO トピックを作成して発行する 432
動画内の人物や物体を検出する 444
S3 を使用して大きなメッセージを管理する 446
S3 イベント通知の処理 450
メッセージをキューに発行する 453

バッチメッセージを送受信する ... 569
Amazon SQS で .NET の AWS メッセージ処理フレームワークを使用する ... 600
Amazon SQS Java Messaging Library を使用して JMS インターフェイスを操作する ... 600
キュータグの操作 ... 623
サーバーレスサンプル ... 626
Amazon SQS トリガーから Lambda 関数を呼び出す ... 626
Amazon SQS トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート ... 635
トラブルシューティング ... 646
アクセス拒否エラー ... 646
Amazon SQS キューポリシーと IAM ポリシー ... 647
AWS Key Management Service（AWS KMS) アクセス許可 ... 648
VPC エンドポイントポリシー ... 649
組織のサービスコントロールポリシー ... 650
API エラー ... 650
QueueDoesNotExist エラー ... 650
InvalidAttributeValue エラー ... 651
ReceiptHandle エラー ... 651
DLQ と DLQ 再処理の問題 ... 652
DLQ の問題 ... 652
DLQ 再処理の問題 ... 654
FIFO スロットリングの問題 ... 656
ReceiveMessage API コールで返されないメッセージ ... 657
空のキュー ... 657
インフライト制限に到達 ... 657
メッセージの遅延 ... 657
メッセージはインフライト状態 ... 657
ポーリング方法 ... 658
ネットワークエラー ... 658
ETIMEOUT error ... 658
UnknownHostException error ... 659
X-Rayを使用したキューのトラブルシューティング ... 660
セキュリティ ... 662
データ保護 ... 662
データ暗号化 ... 663
ネットワーク間のトラフィックのプライバシー ... 675
接続にデュアルスタックエンドポイントを使用する ... 677

Identity and Access Management 678
対象者 678
アイデンティティを使用した認証 679
ポリシーを使用したアクセスの管理 682
概要 685
Amazon Simple Queue Service で IAM を使用する方法 692
AWS マネージドポリシー 700
トラブルシューティング 702
ポリシーの使用 704
ログ記録とモニタリング 752
API コールのログ作成 755
キューのモニタリング 759
コンプライアンス検証 777
耐障害性 778
分散キュー 779
インフラストラクチャセキュリティ 779
ベストプラクティス 780
キューがパブリックアクセス可能でないことの確認 780
最小特権アクセスの実装 781
Amazon SQS 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 SQS 781
サーバー側の暗号化を実装する 782
送信時のデータの暗号化を強制する 782
VPC エンドポイントを使用して Amazon SQSにアクセスすることを検討する場合には 782
関連リソース 784
ドキュメント履歴 785
..... dccxciii

Amazon Simple Queue Serviceとは?

Amazon Simple Queue Service (Amazon SQS)は、分散されたソフトウェアシステムとコンポーネントを統合と分離ができる安全性、耐久性があり利用可能なホストキューを提供します。Amazon SQS は、デッドレターキューやコスト配分タグなど、一般的な構造を提供します。 AWS SDK がサポートする任意のプログラミング言語を使用してアクセスできる汎用ウェブサービス API を提供します。

Amazon SQSを使用する利点

- セキュリティ–Amazon SQS キューにメッセージを送信する人、およびメッセージを受信する人を制御します。Amazon SQS 管理のサーバー側の暗号化を使用するか、 AWS Key Management Service (AWS KMS) で管理されるカスタム SSE キーを使用して、キューのメッセージの内容を保護して、機密データを送信できます。
- 耐久性–メッセージの安全性のため、Amazon SQS は複数のサーバーにメッセージを保存します。標準キューはメッセージの At-least once 配信をサポートし、FIFO キューはメッセージの 1 回のみ処理と高スループットモードをサポートします。
- 可用性–Amazon SQS は、冗長なインフラストラクチャを使用して同時実行性が高いメッセージへのアクセスと、メッセージの作成と消費のための高い可用性を提供します。
- スケーラビリティ– Amazon SQSはバッファされたリクエストを独立して個別に処理できるため、プロビジョニング指示を必要とせずに負荷の増大や急増に対応できるよう透過的にスケーリングします。
- 信頼性–Amazon SQSは処理中にメッセージをロックするため、複数のプロデューサーがメッセージを送信し、複数のコンシューマーが同時にメッセージを受信できます。
- カスタマイズ-キューは完全に同じである必要はありません。たとえば、キューでデフォルトの遅延を設定できます。Amazon Simple Storage Service (Amazon S3) または Amazon DynamoDB を使用して、1 MiB を超えるメッセージの内容を保存できます。Amazon SQS は Amazon S3 オブジェクトへのポインタを保持するか、大きなメッセージを小さなメッセージに分割できます。DynamoDB

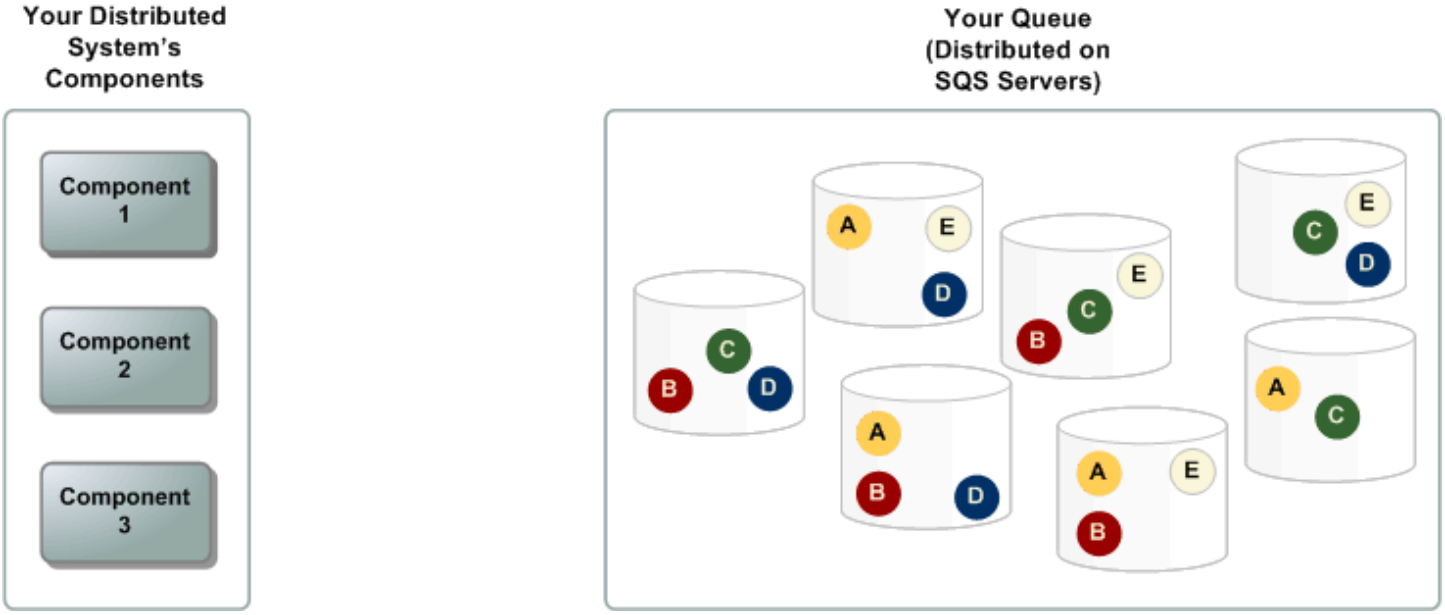
Amazon SQSの基本的なアーキテクチャ

このセクションでは、分散メッセージングシステムのコンポーネントと Amazon SQS メッセージのライフサイクルについて説明します。

分散キュー

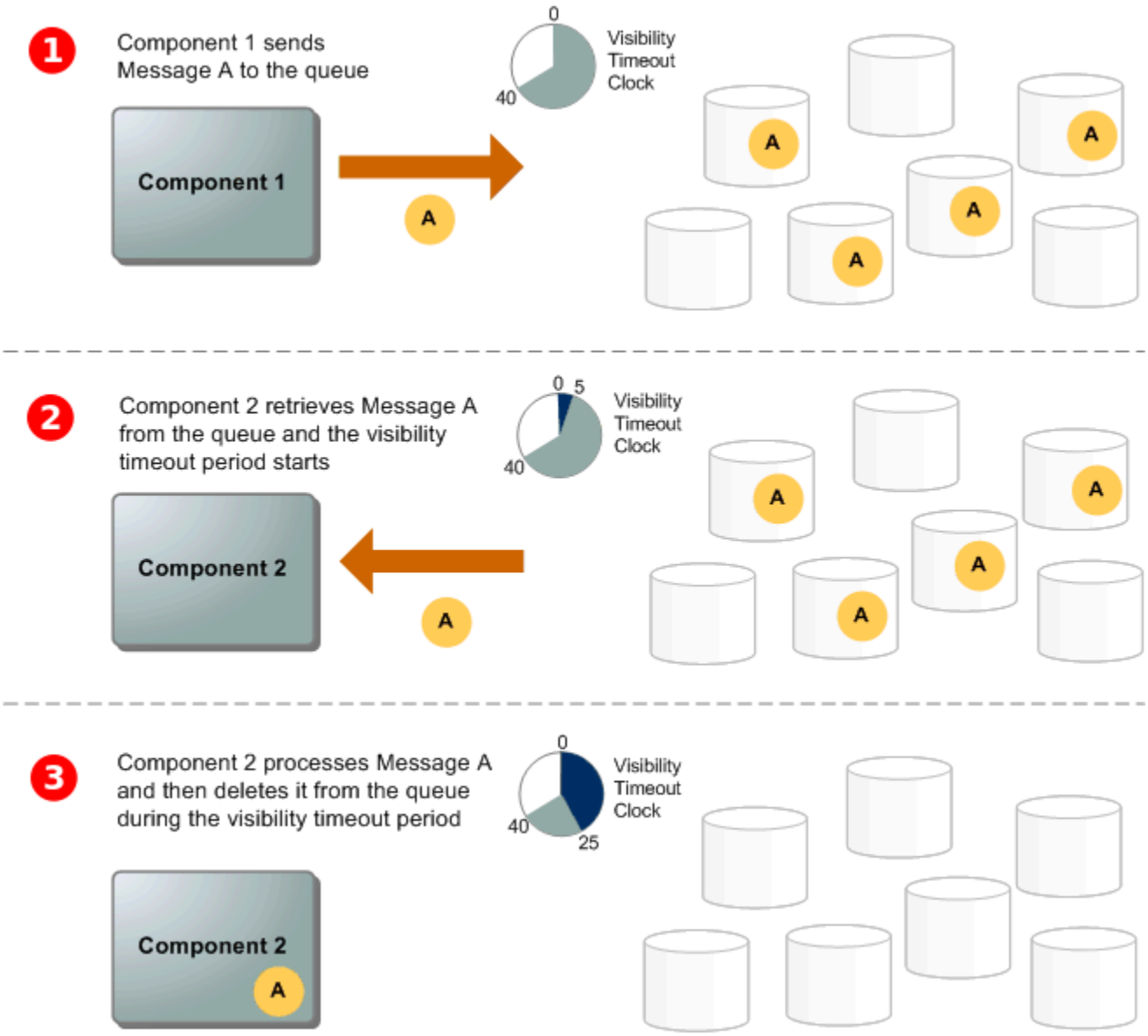
分散メッセージングシステムには、分散システムのコンポーネント、キュー (Amazon SQS サーバーで分散) 、キュー内のメッセージの 3 つの主要部分があります。

次のシナリオでは、システムには複数のプロデューサー (キューにメッセージを送信するコンポーネント) とコンシューマー (キューからメッセージを受信するコンポーネント) があります。キュー (A から E までのメッセージを保持) は、複数のAmazon SQS サーバー全体にまたがって冗長的にメッセージを保存します。



メッセージのライフサイクル

次のシナリオは、キューのAmazon SQSメッセージのライフサイクルを作成から削除までを説明しています。



1

プロデューサー (コンポーネント 1) はメッセージ A をキューに送信し、メッセージは Amazon SQS サーバー全体に冗長的に分散されます。

2

コンシューマー (コンポーネント 2) がメッセージを処理する準備ができたら、キューからのメッセージを消費し、メッセージ A が返されます。メッセージ A は処理されている間キューに残り、可視性タイムアウトの間は次の受信リクエストに返されることはありません。

3

コンシューマー (コンポーネント 2) はキューからメッセージ A を削除して、可視性タイムアウトの期限が切れたときにメッセージを受信して再度処理しないようにします。

> **Note**
>
> Amazon SQSは最大メッセージ保持期間を超えてキューに保存されたメッセージを自動的に削除します。デフォルトのメッセージ保持期間は 4 日間です。ただし、`SetQueueAttributes` アクションを使うと、メッセージ保持期間の値を 60 秒から 1,209,600 秒 (14 日間) までの範囲で設定できます。

Amazon SQS、Amazon MQ、Amazon SNS 間の違い

Amazon SQS、Amazon SNS、Amazon MQ は、高度にスケーラブルで使いやすいマネージドメッセージングサービスを提供します。各サービスは、分散システム内の特定の役割向けに設計されています。これらのサービス間の違いについて、以下に詳しく説明します。

Amazon SQS は、分散ソフトウェアシステムとコンポーネントをキューサービスとしてデカップリングしてスケーリングします。通常、単一のサブスクライバーを通じてメッセージを処理します。順序と損失の防止が重要なワークフローに最適です。より広範に分散する場合は、Amazon SQS を Amazon SNS と統合すると、ファンアウトメッセージングパターンが可能になり、複数のサブスクライバーにメッセージを一度に効果的にプッシュできます。

Amazon SNS を使用すると、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を通じて複数のサブスクライバーにメッセージを送信できます。サブスクライバーは、Amazon Data Firehose、Amazon SQS、Lambda、HTTP、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) などのサポートされているエンドポイントタイプを使用して、発行されたメッセージを受け取ります。このサービスは、リアルタイムのユーザーエンゲージメントやアラームシステムなど、即時の通知を必要とするシナリオに最適です。サブスクライバーがオフラインのときにメッセージの損失を防ぐため、Amazon SNS を Amazon SQS キューメッセージと統合することで、一貫性のある配信を確保します。

Amazon MQ は、従来のメッセージブローカーからの移行を検討している企業に最適であり、AMQP や MQTT などの標準メッセージングプロトコル、Apache ActiveMQ、RabbitMQ をサポートしています。大幅な再設定なしで、安定した信頼性の高いメッセージングを必要とするレガシーシステムとの互換性を提供します。

次の表は、各サービスのリソースタイプの概要を示しています。

リソースタイプ	Amazon SNS	Amazon SQS	Amazon MQ
同期	なし	なし	あり
非同期	はい	あり	あり
キュー	なし	あり	あり
パブリッシャー/サブスクライバーメッセージング	あり	なし	あり
メッセージブローカー	なし	なし	あり

新規のアプリケーションには、Amazon SQS および Amazon SNS をお勧めします。ほぼ無制限のスケーラビリティとシンプルな API が利点です。通常、従量制料金で、大量のアプリケーションに対してコスト効率の高いソリューションを提供します。JMS などの API や、アドバンストメッセージキューイングプロトコル (AMQP)、MQTT、OpenWire、STOMP (Simple Text Oriented Message Protocol) などのプロトコルとの互換性に依存する既存のメッセージブローカーからのアプリケーション移行にはAmazon MQ をお勧めします。

Amazon SQSの開始方法

このトピックでは、Amazon SQS コンソールを使用して標準キューと FIFO キューを作成および管理する方法を説明します。コンソールを操作し、キュー属性を表示し、キュータイプを区別する方法について説明します。主なタスクには、メッセージの送信、受信、設定、可視性タイムアウトやメッセージ保持などのパラメータの調整、ポリシーによるキューアクセスの管理などがあります。

トピック

- Amazon SQSのセットアップ
- Amazon SQS コンソールを理解する
- Amazon SQS キュータイプ
- Amazon SQS 標準キューの作成とメッセージの送信
- Amazon SQS FIFO キューを作成してメッセージを送信する
- Amazon SQSの使用開始の汎用タスク

Amazon SQSのセットアップ

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

ステップ 1: AWS アカウント と IAM ユーザーを作成する

サービスにアクセスするには AWS 、まずAWS アカウント、 AWS 製品を使用できる Amazon.com アカウントである を作成する必要があります。を使用して 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。

Amazon SQS アクションに AWS アカウント ルートユーザーを使用しないようにするには、Amazon SQS への管理アクセスを必要とするユーザーごとに IAM ユーザーを作成することがベストプラクティスです。

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

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

管理アクセスを持つユーザーを作成する

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

を保護する AWS アカウントのルートユーザー

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

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

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

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

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

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

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

管理アクセス権を持つユーザーとしてサインインする

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

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

追加のユーザーにアクセス権を割り当てる

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

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。

2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

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

ステップ 2: プログラム的なアクセス権を付与する

Amazon SQS アクション (Java の使用や 経由など AWS Command Line Interface) を使用するには、アクセスキー ID とシークレットアクセスキーが必要です。

> **Note**
>
> アクセスキー ID とシークレットアクセスキーは に固有です AWS Identity and Access Management。Amazon EC2 キーペアなどの他の AWS サービスの認証情報と混同しないでください。

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 がアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM アイデンティティセンターで管理されているユーザー)	一時的な認証情報を使用して、 AWS CLI、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「 を使用する AWS CLI ように AWS IAM Identity Centerを設定する」を参照してください。 • AWS SDKs、ツール、API については、 AWS APIs「 SDK およびツールリファレンスガイド」の「IAM アイデンティティセンター認証」を参照してください。 AWS SDKs
IAM	一時的な認証情報を使用して、 AWS CLI、 AWS SDKs、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、 AWS CLI、 AWS SDKs、または AWS APIs。	使用するインターフェイスの指示に従ってください。 • については AWS CLI、「 AWS Command Line

プログラマチックアクセス権を必要とするユーザー	目的	方法
		Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 AWS SDKs • API AWS APIs「IAM ユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

ステップ 3: コード例を使用する準備を整える

このガイドには、 AWS SDK for Java を使用する例が含まれています。サンプルコードを実行するには、{AWS SDK for Java 2.0の使用開始 } セットアップ手順に従います。

Go、JavaScript、Python、Ruby など、他のプログラミング言語で AWS アプリケーションを開発できます。詳細については、「構築するツール AWS」を参照してください。

> **Note**
>
> (AWS CLI) や Windows PowerShellなどのツールを使用すると、コードを記述せずに AWS Command Line Interface Amazon SQS を試すことができます。コマンドAWS CLI リファレンスの Amazon SQS セクションに AWS CLI 例があります。Windows PowerShellの例については、AWS Tools for PowerShell コマンドレットリファレンスの「Amazon Simple Queue Service」セクションにあります。

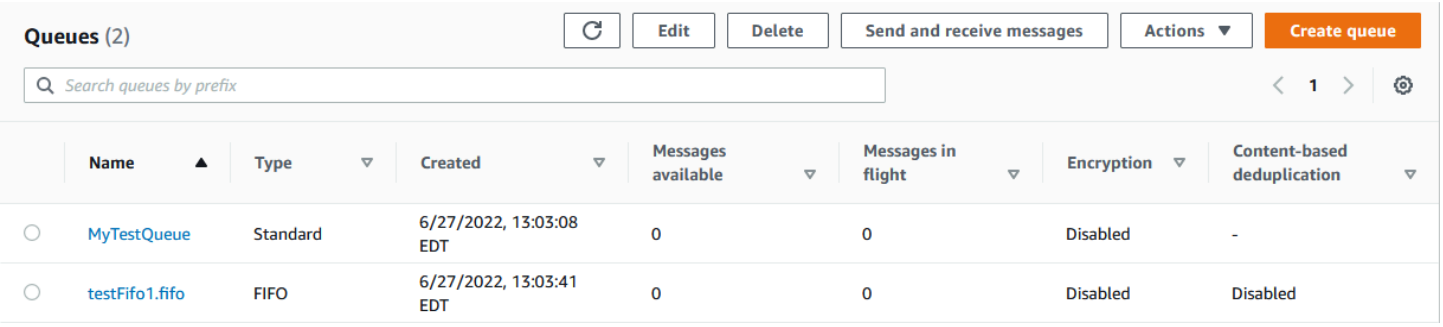
次のステップ

これで、「 AWS Management Consoleを使用してAmazon SQS キューとメッセージを管理する」を開始 する準備ができました。

Amazon SQS コンソールを理解する

Amazon SQS コンソールを開いて、ナビゲーションペインの [キュー] を選択します。[キュー] ページには、アクティブなリージョン内のすべてのキューに関する情報が表示されます。

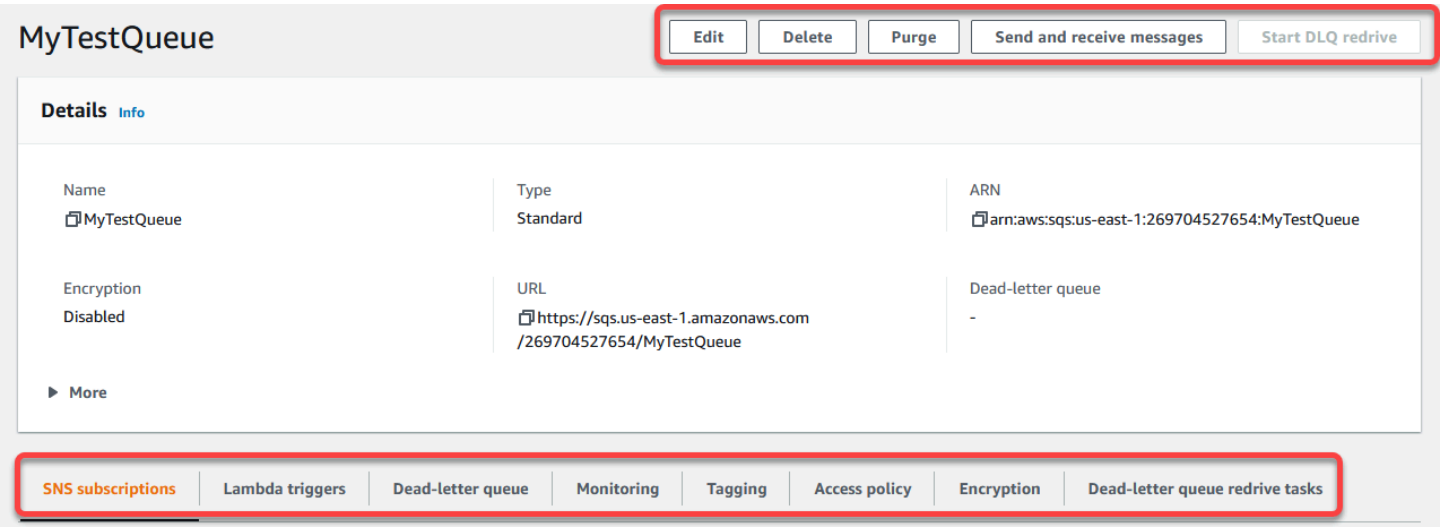
各キューエントリは、キューのタイプや主要な属性など、キューに関する重要な情報を提供します。標準キューは、最大のスループットとベストエフォートのメッセージ順序付けのために最適化されます。一方、FIFO (先入れ先出し) キューは、メッセージの厳密な順序付けを必要とするアプリケーションでメッセージの順序と一意性を優先します。両者は区別されます。



インタラクティブな要素とアクション

[キュー] ページには、キューを管理するための複数のオプションがあります。

1. クイックアクション – 各キュー名の横にあるドロップダウンメニューを使用すると、メッセージの送信、メッセージの表示または削除、トリガーの設定、キュー自体の削除など、一般的なアクションにすばやくアクセスできます。
2. 詳細ビューと設定 – キュー名をクリックすると [詳細] ページが開き、キューの設定や構成を詳しく参照できます。ここでは、メッセージ保持期間、可視性タイムアウト、最大メッセージサイズなどのパラメータを調整し、アプリケーションの要件に合わせてキューをカスタマイズできます。



リージョンの選択とリソースタグ

キューに効果的にアクセスして管理 AWS リージョン するために、正しい にいることを確認します。さらに、リソースタグを使用してキューを整理および分類し、 AWS 共有環境内のリソース管理、コスト配分、アクセスコントロールを向上させることを検討してください。

Amazon SQS コンソールで提供される特徴と機能を活用することで、アプリケーションのメッセージングインフラストラクチャの効率的な管理、キューのパフォーマンスの最適化、信頼性の高いメッセージ配信を実現できます。

Amazon SQS キュータイプ

Amazon SQS は、標準キューと FIFO キューの 2 種類のキューをサポートしています。次の表を使用して、ニーズに最適なキューを決定します。

標準キュー	FIFO キュー
無制限のスループット – 標準キューは、アクション (`SendMessage` 、`ReceiveMessage` 、または `DeleteMessage`) ごとに 1 秒あたり非常に多くの、ほぼ無制限の API コール数をサポートします。この高スループットにより、リアルタイムデータストリーミングや大規模なアプリケーションなど、大量のメッセージをすばやく処理する必要があるユースケース	高スループット — バッチ処理を使用すると、FIFO キューは API メソッド (`SendMessageBatch` 、`ReceiveMessage` 、または`DeleteMessageBatch`) ごとに 1 秒あたり最大 3,000 件のメッセージを処理します。このスループットでは 1 秒あたりの API コール数を 300 とし、各 API コールで 10 件のメッセージをバッチで処理します。高スルー

標準キュー	FIFO キュー
に最適です。標準キューは需要に応じて自動的にスケールされますが、特にワークロードの高いリージョンでは、最適なパフォーマンスを確保するために使用パターンをモニタリングすることが不可欠です。 At-least once 配信 — 少なくとも 1 回の配信が保証されます。つまり、すべてのメッセージは少なくとも 1 回配信されますが、再試行やネットワーク遅延が原因で 2 回以上配信される場合もあります。同じメッセージを複数回処理してもシステムの状態に影響が及ばないように、べき等性オペレーションを使用して重複メッセージに対応できるようにアプリケーションを設計する必要があります。 ベストエフォート型の順序 – ベストエフォート型の順序を提供します。つまり、Amazon SQS はメッセージを送信された順に配信しようとしますが、これは保証されません。場合によっては、特に高スループットまたは障害復旧の条件下では、メッセージが順序どおりに到着しないことがあります。メッセージ処理の順序が重要であるアプリケーションの場合、アプリケーション内で順序変更ロジックを処理するか、FIFO キューを使用して順序を厳密に保証する必要があります。 耐久性と冗長性 – 標準キューは、各メッセージの複数のコピーを複数の AWS アベイラビリティーゾーンに格納することで、高い耐久性を確保します。これにより、インフラストラクチャに障害が発生しても、メッセージは失われません。	プットモードを有効にすると、メッセージグループ内の順序範囲を広げて 1 秒あたりのトランザクション数 (TPS) を最大 30,000 までスケールアップできます。バッチ処理を使用しない場合、FIFO キューは、API メソッド (`SendMessage` 、`ReceiveMessage` 、または `DeleteMessage`) ごとに 1 秒あたり最大 300 件の API コールをサポートします。さらにスループットが必要な場合は、AWS サポートセンターにクォータの引き上げをリクエストできます。高スループットモードを有効にするには、「Amazon SQS における FIFO キューの高スループットの有効化」を参照してください。 1 回のみ処理 — FIFO キューは各メッセージを 1 回配信し、処理して削除するまで、そのメッセージを利用可能にします。`MessageDeduplicationId` やコンテンツベースの重複排除などの機能を使用することで、ネットワークの問題やタイムアウトが原因で再試行した場合でも、メッセージの重複を防ぐことができます。 FIFO 配信 – FIFO キューでは、各メッセージグループ内のメッセージを送信した順序どおりに取り出します。メッセージを複数のグループに分散することで、各グループ内の順序を維持しながら、メッセージを並列処理できます。

標準キュー	FIFO キュー
可視性タイムアウト – Amazon SQS では、可視性タイムアウトを設定してメッセージを受信後に非表示にしておく期間を制御できます。これにより、メッセージを完全に処理するか、タイムアウトが期限切れになるまで、他のコンシューマーがメッセージを処理することはありません。	

標準キュー	FIFO キュー
標準キューは、スループットが重要であるアプリケーション間でのデータ送信に使用します。以下に例を示します。 • ライブユーザーのリクエストを集中的なバックグラウンド作業から切り離します。サイズ変更やエンコードなどのタスクをバックグラウンドで処理している間に、ユーザーがメディアをすばやくアップロードできるようにすることで、システムに過負荷をかけずにレスポンス時間を短縮できます。 • タスクを複数のワーカーノードに割り当てます。多数のクレジットカード検証リクエストを複数のワーカーノードに分散し、べき等性オペレーションを使用して重複メッセージを処理することで処理エラーを回避します。 • 後で処理するためにメッセージをバッチ処理します。複数のエントリをキューに入れてバッチとしてデータベースに追加します。メッセージの順序は保証されないため、必要に応じて順不同の処理に対応できるようシステムを設計します。	FIFO キューは、イベントの順序が重要であるアプリケーション間でのデータ送信に使用します。以下に例を示します。 • ユーザーが入力したコマンドが正しい順に実行されていることを確認します。これは、コマンドの順序が重要である FIFO キューの主要なユースケースです。例えば、ユーザーがアプリケーションで一連のアクションを実行する場合、FIFO キューは、アクションが入力された順序どおりに処理されるようにします。 • 価格変更を正しい順序で送信して、正しい製品価格を表示します。FIFO キューでは、製品価格の複数の更新が到着した順序どおりに処理されます。FIFO を使用しないと、値上げ後に値下げが処理され、誤ったデータが表示される可能性があります。 • 受講者がアカウントに登録する前にコースに登録することを防ぎます。FIFO キューを使用すると、登録プロセスが正しい順序で実行されるようになります。システムは最初にアカウント登録を処理し、次にコース登録を処理するため、コース登録リクエストが先に実行されることはありません。

Amazon SQS でのリクエスト/レスポンスシステムの実装

リクエストと応答またはリモートプロシージャ呼び出し (RPC) システムを実行するときは、次のベストプラクティスに留意してください:

- 起動時に応答キューを作成する — メッセージごとに応答キューを作成するのではなく、プロデューサーごとに起動時に作成します。相関 ID メッセージ属性を使用して、応答をリクエストに効率的にマッピングします。

- プロデューサー間で応答キューを共有しない — プロデューサーごとに独自の応答キューを持っていることを確認します。応答キューを共有すると、プロデューサーが別のプロデューサー向けのレスポンスメッセージを受信する可能性があります。

Temporary Queue Client を使用したリクエスト/応答パターンの実行の詳細については、「リクエスト–レスポンスメッセージングパターン(仮想キュー)」を参照してください。

Amazon SQS 標準キューの作成とメッセージの送信

標準キューを作成し、Amazon SQS コンソールを使用してメッセージを送信できます。このトピックでは、キュー名での機密情報の回避やマネージドサーバー側の暗号化の活用など、ベストプラクティスについても強調します。

Amazon SQS コンソールを使用した標準キューの作成

> **Important**
>
> 2022 年 8 月 17 日に、Amazon SQS キューにデフォルトのサーバー側の暗号化が適用されました。
> 個人を特定できる情報 (PII) やその他の機密情報をキュー名に追加しないでください。キュー名で支払いや CloudWatch Logs を含む多数の Amazon Web Services にアクセスできます。キュー名はプライベートまたは機密データに使用することを意図していません。

Amazon SQS の標準キューを作成するには

1. Amazon SQSコンソールを開きます https://console.aws.amazon.com/sqs/。
2. [キューの作成]を選択します。
3. [タイプ] では、[標準] キュータイプがデフォルトで設定されています。

   > **Note**
   >
   > キューの作成後は、キューのタイプを変更することはできません。

4. キュー名を入力します。
5. （オプション）コンソールはキュー設定パラメータにデフォルト値を設定します。設定では、以下のパラメータに新しい値を設定することができます。

a. 可視性タイムアウトを使用する場合、期間と単位を入力します。指定できる範囲は0秒から12時間です。デフォルト値は30秒です。

b. メッセージの保持期間を使用する場合、期間と単位を入力します。指定できる範囲は1分から14日です。デフォルト値は4日です。

c. 配信の遅れを使用する場合、期間と単位を入力します。指定できる範囲は0から15分です。デフォルト値は0秒です。

d. 最大メッセージサイズを使用する場合、値を入力します。範囲は 1 KiB ~ 1024 KiB です。デフォルト値は 1024 KiB です。

e. メッセージの受信待ち時間を使用する場合、値を入力します。指定できる範囲は0から20秒です。デフォルト値は0秒で、ショートポーリングを設定します。0以外の値を指定すると、ロングポーリングが設定されます。

6. (オプション)アクセスポリシーの定義 アクセスポリシーキューにアクセスできるアカウント、ユーザー、ロールを定義します。アクセスポリシーでは、ユーザーがアクセスできるアクション (`SendMessage`、`ReceiveMessage`、`DeleteMessage` など) も定義します。デフォルトポリシーでは、キューの所有者のみがメッセージの送受信を許可されます。

   アクセスポリシーを定義するには、次のいずれかの操作を行います。

   - キューにメッセージを送信できるユーザーと、キューからメッセージを受信できるユーザーを設定するにはベーシックを選択します。コンソールは、選択に基づいてポリシーを作成し、結果のアクセスポリシーを読み取り専用 JSON パネルに表示します。
   - JSONアクセスポリシーを直接変更する場合は、アドバンストを選択します。これにより、各プリンシパル (アカウント、ユーザー、またはロール) が実行できるアクションのカスタムセットを指定できます。

7. [許可ポリシーの再実行] で、[有効] を選択します。[すべて許可]、[キュー別]、または [すべて拒否] のいずれかを選択します。キュー別選択する場合、Amazon リソースネーム (ARN) で、最大10個のソースキューのリストを指定します。

8. Amazon SQS では、デフォルトでサーバー側の暗号化が使用されます。暗号化キータイプを選択するか、Amazon SQS が管理するサーバー側の暗号化を無効にするには、[暗号化] を展開します。暗号化キータイプの詳細については、「SQS マネージド暗号化キーを使用したキューに対するサーバー側の暗号化の設定」および「Amazon SQS を使用したキューに対するサーバー側の暗号化の設定」を参照してください。

> **Note**
>
> SSE を有効にすると、暗号化されたキューへの匿名の `SendMessage` リクエストと `ReceiveMessage` リクエストは拒否されます。Amazon SQS のセキュリティベストプラクティスでは、匿名リクエストを使用しないことを推奨しています。Amazon SQS キューに匿名リクエストを送信する場合は、必ず SSE を無効にしてください。

9. (オプション) 配信不能メッセージを受信するデッドレターキューを設定するためには、デッドレターキューを拡大します。
10. (オプション)キューのタグを追加するには、タグを拡大します。
11. [キューの作成]を選択します。Amazon SQSはキューを作成し、キューの詳細ページが表示されます。

Amazon SQSは、新しいキューに関する情報をシステム全体に伝えます。Amazon SQS は分散システムであるため、コンソールがキューページにキューを表示するまでに多少の遅れが生じることがあります。

標準キューを使用したメッセージの送信

キューを作成したら、キューにメッセージを送信できます。

1. 左のナビゲーションペインで {キュー} を選択します。キューのリストから作成したキューを選択します。
2. アクションからメッセージの送信と受信を選択します。

   コンソールにメッセージの送信と受信ページが表示されます。
3. {メッセージ本文}に[メッセージテキスト] を (入力) します。
4. 標準キューの場合、[配信の遅延] の値を入力し、単位を選択します。例えば、`60` と入力し、[秒]を選択します。詳細については、「Amazon SQSメッセージタイマー」を参照してください。
5. [メッセージの送信] を選択します。

   メッセージが送信されると、コンソールは成功メッセージを表示します。詳細を表示を選択して、送信されたメッセージに関する情報を表示します。

Amazon SQS FIFO キューを作成してメッセージを送信する

Amazon SQS FIFOキューを作成し、 コンソールを使用してメッセージを送信できます。このトピックでは、可視性タイムアウト、メッセージ保持、重複排除などのキューパラメータをセットアップし、キュー名内の機密情報の回避やサーバー側の暗号化の有効化などのセキュリティのベストプラクティスに従う方法について説明します。また、アクセスポリシーの定義、デッドレターキューの設定、メッセージグループ ID や重複排除 ID などの FIFO 固有の属性を使用したメッセージの送信についても説明します。

Amazon SQS コンソールを使用した FIFO キューの作成

Amazon SQS コンソールを使用して、標準キューおよび FIFO キューを作成できます。コンソールには、キュー名以外のすべての設定にデフォルト値が表示されます。

> **Important**
>
> 2022 年 8 月 17 日に、Amazon SQS キューにデフォルトのサーバー側の暗号化が適用されました。
> 個人を特定できる情報 (PII) やその他の機密情報をキュー名に追加しないでください。キュー名で支払いや CloudWatch Logs を含む多数の Amazon Web Services にアクセスできます。キュー名はプライベートまたは機密データに使用することを意図していません。

Amazon SQS FIFO キュー を作成する方法

1. Amazon SQSコンソールを開きます https://console.aws.amazon.com/sqs/。
2. [キューの作成]を選択します。
3. [タイプ] では、[標準] キュータイプがデフォルトで設定されています。FIFO キューを作成するには、FIFOを選択します。

   > **Note**
   >
   > キューの作成後は、キューのタイプを変更することはできません。

4. キュー名を入力します。

FIFOキューの名前は`.fifo`のサフィックスで終わる必要があります。サフィックスは80文字のキュー名クォータにカウントされます。キューがであるかどうかを確認するにはFIFOでは、キュー名の末尾がサフィックスで終わるかどうかでチェックすることがきます。

5. （オプション）コンソールはキュー設定パラメータにデフォルト値を設定します。設定では、以下のパラメータに新しい値を設定することができます。

   a. 可視性タイムアウトを使用する場合、期間と単位を入力します。指定できる範囲は0秒から12時間です。デフォルト値は30秒です。

   b. メッセージの保持期間を使用する場合、期間と単位を入力します。指定できる範囲は1分から14日です。デフォルト値は4日です。

   c. 配信の遅れを使用する場合、期間と単位を入力します。指定できる範囲は0から15分です。デフォルト値は0秒です。

   d. 最大メッセージサイズを使用する場合、値を入力します。範囲は 1 KiB ~ 1024 KiB です。デフォルト値は 1024 KiB です。

   e. メッセージの受信待ち時間を使用する場合、値を入力します。指定できる範囲は0から20秒です。デフォルト値は0秒で、ショートポーリングを設定します。0以外の値を指定すると、ロングポーリングが設定されます。

   f. FIFO キューの場合は、コンテンツベースの重複除外を有効にするために [コンテンツベースの重複除外] を選択します。デフォルト設定は無効です。

   g. (オプション) FIFO キューの場合、キュー内のメッセージの送受信でより高スループットを有効にするには、[高スループット FIFO を有効にする] を選択します。

      このオプションを選択すると、関連するオプション (重複除外のスコープおよびFIFO スループットの制限) を使用して、FIFO キューの高スループットを有効にするために必要な設定に変更されます。高スループット FIFO の使用に必要な設定のいずれかを変更すると、キューに対して通常のスループットが有効になり、指定されたとおりに重複除外が実行されます。詳細については、「Amazon SQS の FIFO キューの高スループット」および「Amazon SQS のメッセージキュー」を参照してください。

6. (オプション)アクセスポリシーの定義 アクセスポリシーキューにアクセスできるアカウント、ユーザー、ロールを定義します。アクセスポリシーでは、ユーザーがアクセスできるアクション (`SendMessage`、`ReceiveMessage`、`DeleteMessage` など) も定義します。デフォルトポリシーでは、キューの所有者のみがメッセージの送受信を許可されます。

   アクセスポリシーを定義するには、次のいずれかの操作を行います。

- キューにメッセージを送信できるユーザーと、キューからメッセージを受信できるユーザーを設定するにはベーシックを選択します。コンソールは、選択に基づいてポリシーを作成し、結果のアクセスポリシーを読み取り専用 JSON パネルに表示します。
- JSONアクセスポリシーを直接変更する場合は、アドバンストを選択します。これにより、各プリンシパル (アカウント、ユーザー、またはロール) が実行できるアクションのカスタムセットを指定できます。

7. [許可ポリシーの再実行] で、[有効] を選択します。[すべて許可]、[キュー別]、または [すべて拒否] のいずれかを選択します。キュー別選択する場合、Amazon リソースネーム (ARN) で、最大10個のソースキューのリストを指定します。
8. Amazon SQS では、デフォルトでサーバー側の暗号化が使用されます。暗号化キータイプを選択するか、Amazon SQS が管理するサーバー側の暗号化を無効にするには、[暗号化] を展開します。暗号化キータイプの詳細については、「SQS マネージド暗号化キーを使用したキューに対するサーバー側の暗号化の設定」および「Amazon SQS を使用したキューに対するサーバー側の暗号化の設定」を参照してください。

   > **Note**
   >
   > SSE を有効にすると、暗号化されたキューへの匿名の `SendMessage` リクエストと `ReceiveMessage` リクエストは拒否されます。Amazon SQS のセキュリティベストプラクティスでは、匿名リクエストを使用しないことを推奨しています。Amazon SQS キューに匿名リクエストを送信する場合は、必ず SSE を無効にしてください。

9. (オプション) 配信不能メッセージを受信するデッドレターキューを設定するためには、デッドレターキューを拡大します。
10. (オプション)キューのタグを追加するには、タグを拡大します。
11. [キューの作成]を選択します。Amazon SQSはキューを作成し、キューの詳細ページが表示されます。

Amazon SQSは、新しいキューに関する情報をシステム全体に伝えます。Amazon SQS は分散システムであるため、コンソールがキューページにキューを表示するまでに多少の遅れが生じることがあります。

キューを作成したら、メッセージの送信およびメッセージの受信と削除ができます。また、キュータイプ以外のキュー構成の設定も編集することができます。

FIFO キューを使用したメッセージの送信

キューを作成したら、そのキューにメッセージを送信できます。

1. 左のナビゲーションペインで {キュー} を選択します。キューのリストから作成したキューを選択します。
2. アクションからメッセージの送信と受信を選択します。

   コンソールにメッセージの送信と受信ページが表示されます。
3. {メッセージ本文}に[メッセージテキスト] を (入力) します。
4. 先入れ先出し (FIFO) キューの場合、 { メッセージグループ ID } を (入力) します。詳細については、「Amazon SQS の FIFO キュー配信ロジック」を参照してください。
5. (オプション) FIFO キューの場合、メッセージ重複除外IDを入力できます。キューでコンテンツベースの重複除外を有効にした場合、メッセージ重複除外IDは必要ありません。詳細については、「Amazon SQS の FIFO キュー配信ロジック」を参照してください。
6. FIFO キューは、個々のメッセージのタイマーをサポートしていません。詳細については、「Amazon SQSメッセージタイマー」を参照してください。
7. [メッセージの送信] を選択します。

   メッセージが送信されると、コンソールは成功メッセージを表示します。詳細を表示を選択して、送信されたメッセージに関する情報を表示します。

Amazon SQSの使用開始の汎用タスク

キューを作成し、メッセージの送信、受信、削除方法を学習したら、以下を試すことができます。

- Lambda 関数をトリガーして受信メッセージを自動的に処理し、継続的なポーリングを必要とせずにイベント駆動型のワークフローを有効にします。
- SSEおよびその他の機能を含むキューの設定。
- 属性を含むメッセージを送信します。
- VPC からメッセージを送信します。
- Amazon SQS の機能とアーキテクチャについて説明します。
- Amazon SQSを最大限に活用するのに役立つガイドラインと注意事項について説明します。
- AWS SDK for Java 2.x デベロッパーガイドなど、 AWS SDK の Amazon SQS の例をご覧ください。

- Amazon SQS AWS CLI コマンドについて説明します。
- Amazon SQS API アクションについて説明します。
- Amazon SQS をプログラムで操作する方法について説明します。APIs の使用」およびAWS 「 開発センター」を参照してください。
  - Java
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Windows & .NET

- コストとリソースをモニタリングする方法について説明します。
- データを保護する方法について説明します。
- Amazon SQS ワークフローの詳細について説明します。

Amazon SQS キューの管理

コンソールを使用して Amazon SQS キューを管理する方法について説明します。これには、キュー設定の編集、メッセージの受信と削除、キューの空の確認、キューの削除と消去が含まれます。ロングポーリングの使用、可視性タイムアウトの管理、モニタリングダッシュボードまたは によるメトリクスの検証など、効率的なメッセージ処理のベストプラクティスを理解します AWS CLI。実際のステップに従ってキューを維持し、中断を最小限に抑えながらメッセージを効果的に処理します。

コンソールを使用した Amazon SQS キューの編集

Amazon SQS コンソールを使用して、キュー設定パラメータ (キュータイプを除く) を編集し、必要に応じて機能を変更または削除できます。

Amazon SQSキューを編集するには(コンソール)

1. Amazon SQSコンソールの キュー ページを開きます。
2. キューを選択し、[編集] を選択します。
3. (オプション)設定で、キュー設定パラメータを更新する。
4. (オプション) アクセスポリシーを更新するには、アクセスポリシーで JSON ポリシーを変更します。
5. (オプション) デッドレターキューの再実行の許可ポリシーを更新するには、[許可ポリシーの再実行] を展開します。
6. (オプション) 暗号化を更新または削除するには、[暗号化] を展開します。
7. (オプション)デッドレターキュー (配信不能メッセージを受信できる) を追加、更新、または削除するには、デッドレターキュー拡大します。
8. (オプション)タグキューを追加、更新、または削除するには、タグを拡大します。
9. [Save]を選択します。

   - コンソールはキューの詳細ページを表示します。

Amazon SQS でのメッセージの受信と削除

Amazon SQS キューにメッセージを送信した後、それらを取得および削除してアプリケーションワークフローを処理できます。このプロセスにより、安全で信頼性の高いメッセージ処理が保証され

ます。このトピックでは、Amazon SQS コンソールを使用してメッセージを取得および削除する手順と、このオペレーションを最適化するための主要な設定について説明します。以下は、メッセージを受信および削除するための主要な概念です。

1. メッセージの受信
   - Amazon SQS キューからメッセージを取得する場合、特定のメッセージをターゲットにすることはできません。代わりに、1 回のリクエストで取得するメッセージの最大数 (最大 10) を指定します。
   - Amazon SQS の分散性により、メッセージが少ないキューから取得すると、空のレスポンスが返される可能性があります。これを軽減するには：
     - ロングポーリングを使用すると、メッセージが使用可能になるか、ポーリングがタイムアウトするまで待機します。このアプローチにより、不要なポーリングコストが削減され、効率が向上します。
     - 必要に応じてリクエストを再発行します。
2. メッセージの可視性と削除
   - メッセージは取得後に自動的に削除されません。この機能を使用すると、アプリケーションの障害やネットワークの中断が発生した場合にメッセージを再処理できます。
   - 処理後、メッセージを完全に削除するには、明示的に削除リクエストを送信する必要があります。このアクションは、正常に処理されたことを確認します。
   - Amazon SQS コンソールを使用して取得したメッセージは、再取得のために表示され続けます。自動環境の可視性タイムアウト設定を調整して、処理中に他のコンシューマーからのメッセージを一時的に非表示にします。
3. 可視性タイムアウト
   - この設定は、取得後にメッセージが非表示になる期間を決定します。適切なタイムアウトを設定して、メッセージが 1 回だけ処理されるようにし、分散処理中の重複を防ぎます。

コンソールを使用してメッセージを受信および削除するには

1. Amazon SQSコンソール (https://console.aws.amazon.com/sqs/) を開きます。
2. ナビゲーションペインで [キュー] を選択します。
3. キューページで、メッセージを受信するキューを選択し、メッセージの送受信を選択します。
4. メッセージの送受信ページで、メッセージのポーリングを選択します。

   Amazon SQS には、ポーリング期間を示す進行状況バーが表示されます。取得したメッセージはメッセージセクションに表示され、以下が表示されます。

- メッセージ ID
- 送信日
- サイズ
- 受信数

5. メッセージを削除するには、削除するメッセージを選択し、削除を選択します。

   Delete を選択して、Delete Messages ダイアログボックスで削除を確認します。

API ベースのメッセージの取得や削除など、高度なオペレーションの詳細については、Amazon SQS リファレンスガイド」を参照してください。

Amazon SQS キューが空であることを確認する

ほとんどの場合、キューが空かどうかを判断するためにロングポーリングが使用できます。まれに、キューにまだメッセージが含まれている場合でも、空の返信が受信されることがあります。特に作成したキューにメッセージの受信待ち時間を低い値を指定した場合にこのようなことがある場合があります。このセクションでは、キューが空であることを確認する方法について説明します。

キューが空であることを確認するには(コンソール)

1. すべてのプロデューサーのメッセージの送信を停止する。
2. Amazon SQSコンソールを開きます https://console.aws.amazon.com/sqs/ 。
3. ナビゲーションペインで [Queues(キュー)]を選択します。
4. キュー ページで、キュー を選択します。
5. モニタリングタブを選択します。
6. モニタリングダッシュボードの右上にある、{更新}マークの横にある下向き矢印を選択します。ドロップダウンメニューから[自動更新]を選択します。更新間隔は1分のままにしてください。
7. 次のダッシュボードを確認してください:

   - 遅延したメッセージの概数
   - 表示されないメッセージの概数
   - 表示されるメッセージの概数

   数分間、全ての0値が表示されるとキューは空になります。

キューが空であることを確認するには (AWS CLI， AWS API)

1. すべてのプロデューサーにメッセージの送信を停止。
2. 以下のいずれかのコマンドを繰り返し実行します。
   - AWS CLI: `get-queue-attributes`
   - AWS API: `GetQueueAttributes`
3. 以下の属性のメトリックスを確認してください。
   - `ApproximateNumberOfMessagesDelayed`
   - `ApproximateNumberOfMessagesNotVisible`
   - `ApproximateNumberOfMessagesVisible`

   それら全てが0数分間表示されると、キューは空になります。

Amazon CloudWatch メトリックスを利用している場合は、キューが空であると考える前に、連続する複数の0データポイントが表示されることを確認してください。CloudWatchのメトリクスの詳細については、「Amazon SQS で利用可能な CloudWatchメトリクス」を参照してください。

Amazon SQSキューの削除

Amazon SQS キューを使用しなくなり、近い将来使用する予定がない場合は、キューを削除します。

> **Tip**
> キューを削除する前にキューが空であることを確認する場合は、Amazon SQS キューが空であることを確認するを参照してください

空でない場合でも、キューを削除できます。キュー内のメッセージを削除し、キュー自体を削除しない場合は、キューを消去することができます。

キューを削除するには(コンソール)

1. Amazon SQSコンソールを開きますhttps://console.aws.amazon.com/sqs/。
2. ナビゲーションペインで [Queues(キュー)]を選択します。

3. キューのページで{キューの削除}を選択します。
4. [削除] を選択します。
5. キューの削除ダイアログボックスで**delete**を入力して削除を確定します。
6. [削除] を選択します。

キューを削除するには (AWS CLI および API)

必要に応じてキューを削除する適切な方法を選択します。

- AWS CLI: `aws sqs delete-queue`
- AWS API: `DeleteQueue`

Amazon SQS を使用したキューからのメッセージのクリア

Amazon SQS キューを保持し、そのすべてのメッセージを削除するには、キューを消去します。これにより、現在表示されていないメッセージ (転送中) を含むすべてのメッセージが削除されます。パージプロセスには最大 60 秒かかる場合があるため、キューのサイズに関係なく 60 秒間待機します。

> **Important**
> キューをクリアすると、削除されたメッセージを取得できなくなります。

キューをクリアするには (コンソール)

1. Amazon SQSコンソール (https://console.aws.amazon.com/sqs/) を開きます。
2. ナビゲーションペインで [Queues (キュー)]を選択します。
3. [キュー] ページで、クリアするキューを選択します。
4. [アクション] で、[クリア] を選択します。
5. [キューをクリア] ダイアログボックスで、「**purge**」と入力して [クリア] を選択し、クリアを確定します。
   - すべてのメッセージがキューからクリアされます。コンソールに確認バナーが表示されます。

Amazon SQS の標準キュー

Amazon SQS は、デフォルトのキュータイプとして標準キューを提供し、`SendMessage`、`ReceiveMessage`、`DeleteMessage` などのアクションに対して 1 秒あたりほぼ無制限の数の API コールをサポートしています。標準キューでは、少なくとも 1 回のメッセージ配信が保証されますが、分散性の高いアーキテクチャにより、メッセージの複数のコピーが配信されたり、メッセージが順不同で着信したりすることがあります。それにもかかわらず、標準キューは、メッセージが送信された順序を維持しようと最善を尽くします。

`SendMessage` を使用してメッセージを送信すると、Amazon SQS はメッセージを確認する前に、メッセージを複数のアベイラビリティーゾーン (AZ) に冗長的に保存します。この冗長性により、単一のコンピュータ、ネットワーク、または AZ の障害によってメッセージがアクセス不能になることがなくなります。

キューは、Amazon SQS コンソールを使用して作成および設定できます。詳細な手順については、「Amazon SQS コンソールを使用した標準キューの作成」を参照してください。Java 固有の例については、「Amazon SQS Java SDKの例」を参照してください。

標準キューのユースケース

標準メッセージキューは、メッセージの複数の着信や順不同の着信をアプリケーションで処理できる限り、さまざまなシナリオに適しています。以下に例を示します。

- ライブユーザーリクエストを集中的なバックグラウンド作業から分離する – システムがメディアのサイズ変更やエンコーディングをバックグラウンドで行っている間に、ユーザーはメディアをアップロードできます。
- タスクを複数のワーカーノードに割り当てる – 大量のクレジットカード検証リクエストを処理する場合などが該当します。
- 後で処理するためにメッセージをバッチ処理する - 複数のエントリを後でデータベースに追加するようスケジュールします。

標準キューに関連するクォータについては、「Amazon SQS 標準キューのクォータ」を参照してください。

標準キューを使用した作業のベストプラクティスについては、「Amazon SQS のベストプラクティス」を参照してください。

Amazon SQS の少なくとも 1 回の配信

Amazon SQSでは、冗長性と高可用性を確保するため、メッセージのコピーが複数のサーバーに保存されます。まれではありますが、メッセージを受信または削除するときに、メッセージのコピーが保存されているサーバーの1台が使用できない場合があります。

このような場合、使用できないサーバーではメッセージのコピーが削除されないため、メッセージの受信時に同じメッセージのコピーをもう一度受け取る可能性があります。アプリケーションがべき等になるよう設計する必要があります (同じメッセージを繰り返し処理した場合にも悪影響が発生しないように設計する必要があります)。

Amazon SQSキューとメッセージの識別子

このトピックでは、標準キューと FIFO キューの識別子について説明します。これらの識別子は、特定のキューとメッセージを見つけて操作するうえで役立ちます。

Amazon SQS の標準キューの識別子

次の識別子の詳細については、Amazon Simpleキューサービス APIリファレンス。を参照してください

キュー名およびURL

新しいキューを作成する際は、 AWS アカウントおよびリージョンに一意なキュー名を指定する必要があります。Amazon SQS は、作成したキューごとにキュー URL と呼ばれる識別子を割り当てます。これには、キュー名と他の Amazon SQS コンポーネントが含まれます。キューでアクションを実行するときは必ず、そのキュー URLを指定します。

次に示すのは、AWSアカウント番号`MyQueue`を持つユーザーにより所有される`123456789012`という名前のキューのキュー URL です。

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
```

キューを一覧表示し、アカウント番号に続く文字列を解析することで、キューのURLをプログラムで取得できます。詳細については、「`ListQueues`」を参照してください。

メッセージ ID

各メッセージは、システム割り当てのメッセージ ID を受け取ります。この ID は、Amazon SQS から `SendMessage` レスポンスで返されます。この識別子は、メッセージを特定する場合に役立ちます。メッセージ IDの最大長は100文字です。

受信ハンドル

キューからメッセージを受信するたびに、そのメッセージの受信ハンドルを受け取ります。このハンドルは、メッセージ自体ではなくメッセージ受信のアクションと関連付けられます。メッセージを削除したり、メッセージ可視性を変更したりするには、受信ハンドル (メッセージ ID ではなく) を指定する必要があります。つまり、メッセージを削除する前にメッセージを受信する必要があります (メッセージをキューにおいてから回収することはできません)。受信ハンドルの最大長は1,024文字です。

> **Important**
>
> メッセージを複数回受信した場合、受信するたびに異なる受信ハンドルを受け取ります。メッセージの削除をリクエストするときは、最後に受け取った受信ハンドルを指定してください (そうしないと、メッセージが削除されない可能性があります)。

以下は、3 行に分割された受信ハンドルの例です。

```
MbZj6wDWli+JvwwJaBV+3dcjk2YW2vA3+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Amazon SQS FIFO キュー

FIFO (First-In-First-Outキューには標準キューのすべての機能がありますが、オペレーションとイベントの順序が重要である場合、または重複を許容できない場合に、アプリケーション間のメッセージングを強化するように設計されています。

FIFO キューの最も重要な機能は、 FIFO (先入れ先出し) 配信と 1 回のみ処理です。

- メッセージを送信および受信する順序を厳密に保持します。メッセージは、一度配信されると、コンシューマーが処理して削除するまで利用できなくなります。
- キューには重複が導入されません。

また、FIFO キューはメッセージグループをサポートしており、単一のキュー内で複数のメッセージグループを順序付けて使用できます。FIFOキュー内のメッセージグループの数にクォータはありません。

FIFO キューを使用する可能性のある状況の例を次に示します。

1. 順序が重要な e コマース注文管理システム
2. イベントを順番に処理する必要があるサードパーティシステムとの統合
3. ユーザーが入力した入力を入力順に処理する
4. 通信とネットワーク — データと情報を同じ順序で送受信する
5. コンピュータシステム - ユーザーが入力したコマンドが正しい順序で実行されるようにする
6. 教育機関 - アカウントに登録前に受講者がコースに登録するのを防ぐ
7. オンラインチケットシステム — チケットを先着順で配布する

> **Note**
>
> また、FIFO キューはただ 1 回のみの処理を行いますが、1 秒あたりのトランザクション (TPS) 数は制限されます。FIFO キューで Amazon SQS 高スループットモードを使用すると、トランザクションの上限を引き上げることができます。高スループットモードの使用方法の詳細については、「Amazon SQS の FIFO キューの高スループット」を参照してください。スループットクォータの詳細については、「the section called “メッセージのクォータ”」を参照してください。

Amazon SQS FIFO キューは、Amazon SQSが利用可能なすべてのリージョンで利用可能です。

複雑な順序で FIFO キューを使用する方法の詳細については、「Amazon SQS FIFO キューによる複雑な順序付けの課題の解決」を参照してください。

Amazon SQS コンソールを使用してキューを作成および設定する方法については、{Amazon SQS コンソールを使用した標準キューの作成}を参照してください。Javaの例については、{Amazon SQS Java SDKの例}を参照してください。

FIFO キューを使用するためのベストプラクティスについては、「Amazon SQS のベストプラクティス」を参照してください。

Amazon SQS の FIFO キューの主要な用語

以下の重要な用語は、FIFOキューの機能をよりよく理解するうえで役立ちます。詳細については、「Amazon Simple キューService APIリファレンス」を参照してください。

クライアント

Amazon SQSバッファリング非同期クライアントは現在 FIFOキューをサポートしていません。

メッセージ重複排除ID

メッセージを一意に識別して重複を防ぐために Amazon SQS の FIFO キューで使用するトークン。同じ重複排除 ID を持つ複数のメッセージを 5 分間の重複排除間隔内に送信すると、これらは重複として扱われ、1 つのコピーのみが配信されます。重複排除 ID を指定しない場合、コンテンツベースの重複排除が有効になっていると、Amazon SQS はメッセージ本文をハッシュして重複排除 ID を生成します。このメカニズムは、指定した時間枠内の重複するメッセージを削除することで、正確に 1 回だけ配信されるようにします。

> **Note**
> Amazon SQS は、メッセージの受信と削除後も重複排除 ID の追跡を続けます。

メッセージグループ ID

FIFO (First-In-First-Out) キューでは、 `MessageGroupId`はメッセージを個別のグループに整理する属性です。同じメッセージグループ内のメッセージは常に 1 つずつ厳密な順序で処理されるため、同じグループからの 2 つのメッセージが同時に処理されることはありません。標準キューで

は、 を使用すると公平キューMessageGroupIdが有効になります。 ???厳密な順序付けが必要な場合は、FIFO キューを使用します。

受信リクエスト試行ID

受信リクエスト試行 ID は、Amazon SQS でのReceiveMessage呼び出しの重複排除に使用される一意のトークンです。

シーケンス番号

Amazon SQS が各メッセージに割り当てる、連続しないラージ番号。

サービス

アプリケーションが複数のサービスを使用している場合 AWS 、または AWS と外部サービスの組み合わせを使用している場合は、どのサービス機能が FIFO キューをサポートしていないかを理解することが重要です。

Amazon SQS に通知を送信する一部の AWS または外部サービスは、FIFO キューをターゲットとして設定できるにもかかわらず、FIFO キューと互換性がない場合があります。

現在、 AWS サービスの以下の機能は FIFO キューと互換性がありません。

- Amazon S3イベント通知
- Auto Scalingライフサイクルフック
- AWS IoT ルールアクション
- AWS Lambda デッドレターキュー

FIFOキューと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 SQS の FIFO キュー配信ロジック

以下の概念では、Amazon SQS FIFO キューがメッセージの送受信を処理する方法、特にメッセージの順序付けとメッセージグループ IDs を処理する方法について説明します。

メッセージの送信

Amazon SQS FIFO キューは、一意の重複排除 IDsとメッセージグループ IDs を使用してメッセージの順序を維持します。このトピックでは、グループ内で厳密な順序を維持するためのメッセージグループ IDs の重要性と、複数のプロデューサー間で信頼性が高く順序付けられたメッセージ配信を確保するためのベストプラクティスについて説明します。

1. 注文の保存
   - 一意のメッセージ重複排除 IDs を持つ FIFO キューに複数のメッセージが連続して送信されると、Amazon SQS はそれらを保存して送信を承認します。これらのメッセージは、送信された正確な順序で受信および処理されます。
2. メッセージグループ ID
   - FIFO キューでは、メッセージはメッセージグループ ID に基づいて順序付けられます。複数のプロデューサーまたはスレッドが同じメッセージグループ ID でメッセージを送信する場合、Amazon SQS はそれらが到着した順序で保存および処理されるようにします。
   - ベストプラクティス: 複数のプロデューサー間で厳密なメッセージ順序を保証するには、各プロデューサーからのすべてのメッセージに一意のメッセージグループ ID を割り当てます。
3. グループごとの順序付け
   - FIFO キューロジックは、メッセージグループ ID ごとに適用されます。
     - 各メッセージグループ ID は、順序付けられた個別のメッセージグループを表します。
     - メッセージグループ ID 内では、すべてのメッセージが厳密な順序で送受信されます。
     - メッセージグループ IDsが異なるメッセージは、相互に関連して受信または処理される場合があります。
   - 要件 - メッセージグループ ID を各メッセージに関連付ける必要があります。グループ ID なしでメッセージを送信すると、アクションは失敗します。
   - 単一グループシナリオ - すべてのメッセージを厳密な順序で処理する必要がある場合は、すべてのメッセージに同じメッセージグループ ID を使用します。

メッセージの受信

Amazon SQS FIFOキューは、バッチ処理、FIFO 注文の保証、特定のメッセージグループ IDs。このトピックでは、Amazon SQS が厳密な順序付けと可視性ルールを維持しながら、メッセージグループ IDs 内およびメッセージグループ ID 間でメッセージを取得する方法について説明します。

1. バッチ取得
   - 複数のメッセージグループ IDs、Amazon SQS は次のようになります。
     - 1 回の呼び出しで、同じメッセージグループ ID を持つメッセージをできるだけ多く返そうとします。
     - 他のコンシューマーが異なるメッセージグループ IDsからのメッセージを同時に処理できるようにします。
   - 重要な明確化

- 同じメッセージグループ ID から複数のメッセージを 1 つのバッチで受信できます (`MaxNumberOfMessages`パラメータを使用して 1 回の呼び出しで最大 10 個のメッセージ)。
- ただし、次のリクエストで同じメッセージグループ ID から追加のメッセージを受信することはできません。
  - 現在受信されているメッセージが削除される、または
  - それらは再び表示されます (可視性タイムアウトの有効期限が切れた後など)。

2. FIFO 注文の保証
   - バッチで取得されたメッセージは、FIFO 順序をグループ内に保持します。
   - 同じメッセージグループ ID で使用できるメッセージが 10 件未満の場合、Amazon SQS は同じバッチ内の他のメッセージグループ IDs からのメッセージを含めることができますが、各グループは FIFO 順序を保持します。
3. コンシューマーの制限
   - 特定のメッセージグループ ID からのメッセージの受信を明示的にリクエストすることはできません。

複数回の再試行

プロデューサーとコンシューマーは、メッセージの順序を中断したり、重複させたりすることなく、Amazon SQS FIFO キューで失敗したアクションを安全に再試行できます。このトピックでは、重複IDs と可視性タイムアウトが再試行中のメッセージの整合性を確保する方法について説明します。

1. プロデューサーの再試行
   - `SendMessage` アクションが失敗した場合、プロデューサーは同じメッセージ重複排除 ID を使用してメッセージの送信を複数回再試行できます。
   - 重複排除間隔が切れる前にプロデューサーが少なくとも 1 つの確認を受け取る限り、 は再試行します。
     - 重複したメッセージは導入しないでください。
     - メッセージの順序を中断しないでください。
2. コンシューマーの再試行
   - `ReceiveMessage` アクションが失敗した場合、コンシューマーは同じ受信リクエスト試行 ID を使用して必要な回数だけ再試行できます。

- 可視性タイムアウトの有効期限が切れる前にコンシューマーが少なくとも 1 つの確認を受け取る限り、 は再試行します。
  - メッセージの順序を中断しないでください。

FIFO の動作に関するその他の注意事項

可視性タイムアウトの処理、複数のメッセージグループ IDs による並列処理の有効化、単一グループシナリオでの厳密なシーケンシャル処理について説明します。

1. 可視性タイムアウトの処理
   - メッセージが取得されても削除されない場合、可視性タイムアウトの有効期限が切れるまで表示されません。
   - 最初のメッセージが削除されるか、再び表示されるまで、同じメッセージグループ ID からの追加のメッセージは返されません。
2. 同時実行と並列処理
   - FIFO キューを使用すると、さまざまなメッセージグループ IDs。
   - 同時実行を最大化するには、独立したワークフロー用に複数のメッセージグループ IDsを使用してシステムを設計します。
3. 単一グループのシナリオ
   - FIFO キュー内のすべてのメッセージを厳密に順次処理するには、キュー内のすべてのメッセージに単一のメッセージグループ ID を使用します。

理解を深めるための例

以下は、Amazon SQS での FIFO キューの動作を示す実用的なシナリオです。

1. シナリオ 1: 単一グループ ID
   - プロデューサーは、同じメッセージグループ ID グループ A を持つ 5 つのメッセージを送信します。
   - コンシューマーは、これらのメッセージを FIFO 順に受信します。コンシューマーがこれらのメッセージを削除するか、可視性タイムアウトの有効期限が切れるまで、グループ A からの追加のメッセージは受信されません。
2. シナリオ 2: 複数のグループ IDs

- プロデューサーはグループ A に 5 つのメッセージを送信し、グループ B に 5 つのメッセージを送信します。
- コンシューマー 1 はグループ A からのメッセージを処理し、コンシューマー 2 はグループ B からのメッセージを処理します。これにより、各グループ内で厳密に順序付けされた並列処理が可能になります。

3. シナリオ 3: バッチ取得
   - プロデューサーはグループ A に 7 つのメッセージを送信し、グループ B に 3 つのメッセージを送信します。
   - 1 つのコンシューマーが最大 10 個のメッセージを取得します。キューが許可する場合、以下を返すことがあります。
     - グループ A からの 7 つのメッセージとグループ B からの 3 つのメッセージ (1 つのグループから利用できるメッセージが少ない場合はそれ以下) 。

Amazon SQS の 1 回のみ処理

標準キューとは異なり、FIFO キューは重複メッセージを受け入れません。FIFOキューを使用すると、重複をキューに送信することを防止するのに役立ちます。5分間の重複除外間隔内にSendMessageアクションを再試行しても、Amazon SQS ではキューに重複を招きません。

重複排除を設定するには、次のいずれかを実行する必要があります。

- コンテンツベースの重複排除を有効にします。これは、Amazon SQSにSHA-256ハッシュを使用して、メッセージ本文を使用したメッセージ重複除外ID(ただしメッセージの属性ではない) を生成するように指示するものです。詳細については、「CreateQueue、GetQueueAttributes、 およびSetQueueAttributes内のアクションAmazon Simple Queue Service APIリファレンスのドキュメント」を参照してください。
- メッセージに明示的にメッセージ重複排除 IDを指定 (またはシーケンス番号を参照) します。詳細については、「SendMessage、SendMessageBatch、 およびReceiveMessage内のアクションAmazon Simple Queue Service APIリファレンスのドキュメント」を参照してください。

Amazon SQS での標準キューから FIFO キューへの移行

既存のアプリケーションで標準キューを利用していて、FIFO キューの特徴である順序付けや 1 回のみ処理を活用したい場合は、キューとアプリケーションの両方を正しく設定する必要があります。

主な考慮事項

- FIFO キューの作成: 既存の標準キューを FIFO キューに変換することはできません。アプリケーション用に新しい FIFO キューを作成するか、既存の標準キューを削除して FIFO キューとして再作成する必要があります。
- 遅延パラメータ: FIFO キューでは、メッセージごとの遅延をサポートしていません。キューごとの遅延のみをサポートしています。アプリケーションで `DelaySeconds` パラメータをメッセージごとに設定している場合は、代わりに `DelaySeconds` をキュー全体に設定するようにアプリケーションを変更する必要があります。
- メッセージグループ ID: 送信するメッセージごとにメッセージグループ ID を指定します。この ID により、各メッセージの順序を維持しながら、メッセージを並列処理できます。メッセージグループ ID に詳細なビジネスディメンションを使用すると、FIFO キューをより適切にスケールできます。メッセージの配布先のメッセージグループ ID が多いほど、消費できるメッセージの数が増えます。
- 高スループットモード: スループットを向上させるには、FIFO キューの推奨される高スループットモードを使用します。メッセージのクォータの詳細については、「Amazon SQS のメッセージキュー」を参照してください。

FIFO キューに移行するためのチェックリスト

FIFO キューにメッセージを送信する前に、以下を確認してください。

1. 遅延設定を構成する
   - メッセージごとの遅延を削除するようにアプリケーションを変更します。
   - `DelaySeconds` パラメータをキュー全体に設定します。
2. メッセージグループ ID を設定する
   - ビジネスディメンションに基づいてメッセージグループ ID を指定し、メッセージをメッセージグループに編成します。
   - スケーラビリティを高めるには、よりきめ細かなビジネスディメンションを使用します。
3. メッセージの重複排除を処理する
   - アプリケーションが同じメッセージ本文でメッセージを送信できない場合は、メッセージごとに一意のメッセージ重複排除 ID を指定します。
   - アプリケーションでメッセージごとに独自のメッセージ本文を使用して送信する場合は、コンテンツベースの重複排除を有効にします。
4. コンシューマーを設定する

- 通常、コンシューマーがコードを変更する必要はありません。
- メッセージの処理に時間がかかり、可視性タイムアウトを長く設定する場合は、`ReceiveMessage` アクションごとに受信リクエスト試行 ID を追加することを検討してください。これにより、ネットワーク障害の発生時に受信試行を繰り返し、受信試行の失敗によるキューの一時停止を防止できます。

以上のステップに従うことで、アプリケーションで FIFO キューが正しく動作し、順序付け機能および 1 回のみ処理機能を最大限に活用できます。詳細については、「Amazon Simple Queue Service APIリファレンス」を参照してください。

Amazon SQS の FIFO キューと Lambda の同時実行動作

FIFO (先入れ先出し) キューを Lambda で使用すると、各メッセージグループ内のメッセージを順序どおりに処理できます。Lambda 関数は、同じメッセージグループに対して複数のインスタンスを同時に実行しないため、順序が維持されます。ただし、複数のメッセージグループを並列処理するようにスケールアップできるため、キューのワークロードを効率的に処理できます。メッセージグループ ID に関して Amazon SQS の FIFO キューのメッセージを Lambda 関数が処理する動作のポイントを以下に示します。

- メッセージグループごとに 1 つのインスタンス: どの時点においても、特定のメッセージグループ ID のメッセージを処理しているのは 1 つの Lambda インスタンスのみです。これにより、同じグループ内のメッセージは順番に処理され、FIFO シーケンスの整合性が維持されます。
- 複数の異なるグループの同時処理: Lambda は、複数のインスタンスを使用して、複数の異なるメッセージグループ ID のメッセージを同時に処理できます。つまり、Lambda の同時実行機能を活用すると、複数のグループを並列処理できるため、Lambda 関数の 1 つのインスタンスで 1 つのメッセージグループ ID のメッセージを処理するとともに、他のインスタンスで他のメッセージグループ ID のメッセージを同時に処理できます。

FIFO キューのメッセージのグループ化

FIFO キューでは、メッセージを送信順どおりに処理します。メッセージグループ ID を使用して、順次処理する必要があるメッセージをグループ化します。

同じメッセージグループ内のメッセージは順番に処理されます。この順序を維持するために、各グループのメッセージは一度に 1 つだけ処理されます。

FIFO キューでの Lambda の同時実行

キューを作成したら、そのキューにメッセージを送信できます。

Amazon SQS の FIFO キューのメッセージを処理するように Lambda 関数を設定すると、Lambda は FIFO キューの順序付けに厳密に従って処理します。メッセージグループ ID を使用して Amazon SQS の FIFO キューのメッセージを処理する際の Lambda 関数の同時実行とスケーリングに関する動作のポイントを以下に示します。

- メッセージグループ内の同時実行: Lambda の 1 つのインスタンスのみが、特定のメッセージグループ ID のメッセージを一度に 1 つ処理します。これにより、グループ内のメッセージが順番に処理されます。
- スケーリングと複数のメッセージグループ: Lambda はメッセージグループが異なる場合に限り、複数のメッセージを同時に処理するようにスケールアップできます。複数のメッセージグループがある場合、Lambda は複数のグループを並列して処理できます。グループごとに別個の Lambda インスタンスで処理します。

詳細については、「AWS Lambda デベロッパーガイド」の「Lambda のスケーリングと同時実行」を参照してください。

ユースケースの例

FIFO キューが同じメッセージグループ ID のメッセージを受信し、Lambda 関数の同時実行の制限が高い (最大 1,000) とします。

グループ ID「A」の 1 つのメッセージを処理中に、グループ ID「A」から別のメッセージが到着した場合、最初のメッセージが完全に処理されるまで、2 番目のメッセージは新しい Lambda インスタンスをトリガーしません。

ただし、グループ ID「A」とグループ ID「B」からメッセージが到着した場合、両方のメッセージは別々の Lambda インスタンスで同時に処理できます。

Amazon SQS の FIFO キューの高スループット

Amazon SQS の高スループットの FIFO キューは、厳格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。

Amazon SQS の高スループットの FIFO キューは、厳密なメッセージの順序付けが重要ではなく、受信メッセージのボリュームが比較的低いか散発的である場合には必要ありません。例えば、低頻度または非連続のメッセージを処理する小規模なアプリケーションの場合、高スループットの FIFO キューに伴う複雑さやコストの増加は割に合わない場合があります。さらに、高スループットの FIFO キューが提供する拡張スループット機能を必要としないアプリケーションでは、Amazon SQS の標準キューを選択する方が、コスト効率が高まり、管理が容易になる場合があります。

高スループットの FIFO キューのリクエストキャパシティを強化するには、メッセージグループの数を増やすことをお勧めします。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 SQS Service Quotas」を参照してください。

キューごとのクォータとデータ分散戦略については、「Amazon SQS のメッセージキュー」および「SQS FIFOキュー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」を参照してください。

Amazon SQS FIFO キューの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、高スループットの FIFO キューのさまざまな用途に注目し、さまざまな業界やシナリオにわたる有効性を示します。

1. リアルタイムのデータ処理: イベント処理やテレメトリデータインジェストなど、リアルタイムのデータストリームを扱うアプリケーションは、高スループットの FIFO キューを利用することで、メッセ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、正確な分析を行うことができます。
2. e コマースの注文処理: 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、高スループットの FIFO キューにより、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処理できます。
3. 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、高スループットの FIFO キューに依存することで、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。
4. 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、高スループットの FIFO キューを使用して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もたらすことができます。

SQS FIFOキュー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

Amazon SQSは、FIFOキューデータをパーティションに保存します。パーティションは、 AWS リージョン内の複数のアベイラビリティーゾーンに自動的にレプリケートされるキューのストレージの割り当てです。パーティションは管理しません。代わりに Amazon SQSがパーティション管理を処理します。

FIFOキューの場合、Amazon SQSは次の状況でキューのパーティションの数を変更します。

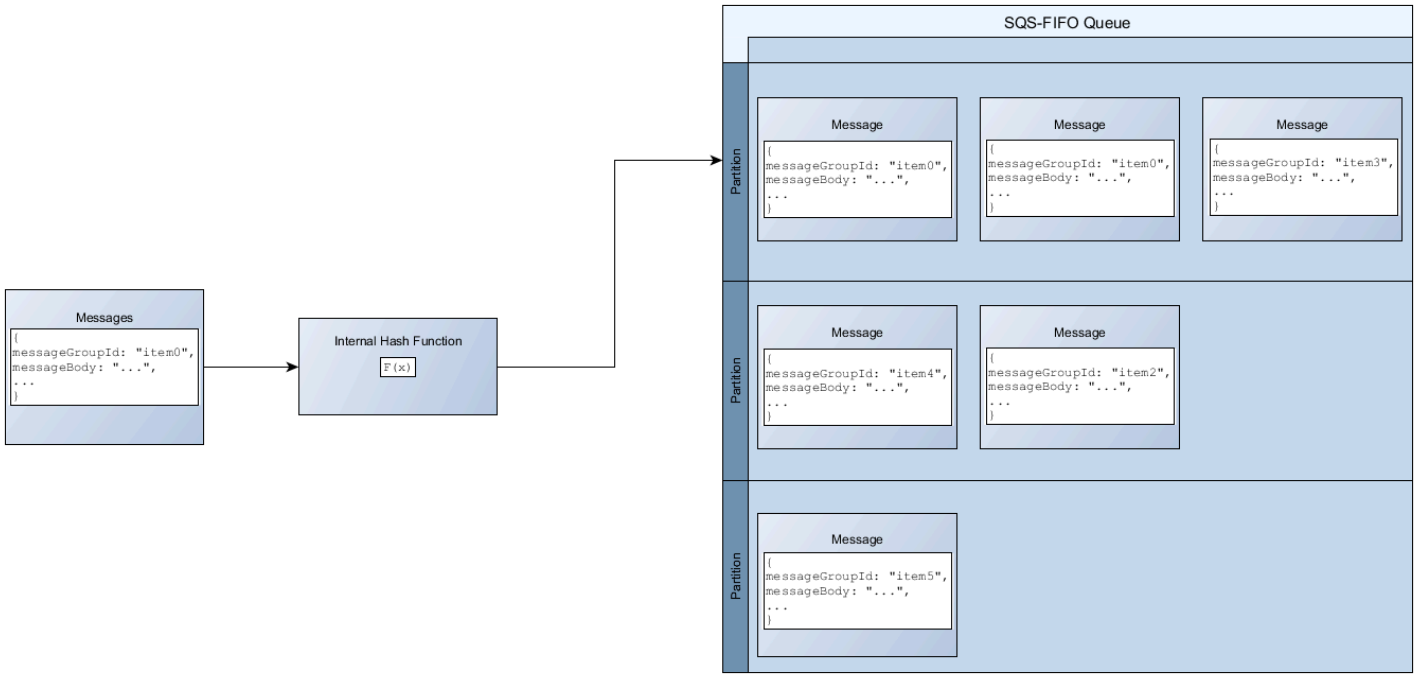
- 現在のリクエストレートが既存のパーティションがサポートできる値に近づいたり超えたりすると、キューがリージョナルクォータに達するまで追加のパーティションが割り当てられます。クオータの詳細については、{Amazon SQS のメッセージキュー}を参照してください。
- 現在のパーティションの使用率が低い場合は、パーティションの数が減ることがあります。

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。キューとメッセージは常に利用可能です。

メッセージグループ ID によるデータの配布

FIFOキューにメッセージを追加するには、Amazon SQS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として使用します。ハッシュ関数からの出力値によって、どのパーティションにメッセージが保存されるが決まります。

次の図は、複数のパーティションにまたがるキューを示しています。キューのメッセージグループIDは、アイテム番号に基づきます。Amazon SQSは、ハッシュ関数を使用して、新しい項目の保存場所を決定します。この場合は、文字列のハッシュ値に基づいていますitem0。アイテムは、キューに追加される順序と同じ順序で格納されることに注意してください。各アイテムの場所は、メッセージグループIDのハッシュ値によって決まります。



> **Note**
>
> Amazon SQS は、パーティションの数に関係なく、FIFO キュー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。

パーティション使用率の最適化

サポートされているリージョンにおいては、各パーティションは、バッチ処理により 1 秒あたり最大 3,000 件のメッセージ、または送信、受信、削除操作では 1 秒あたり最大 300 件のメッセージをサポートします。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 SQS Service Quotas」を参照してください。

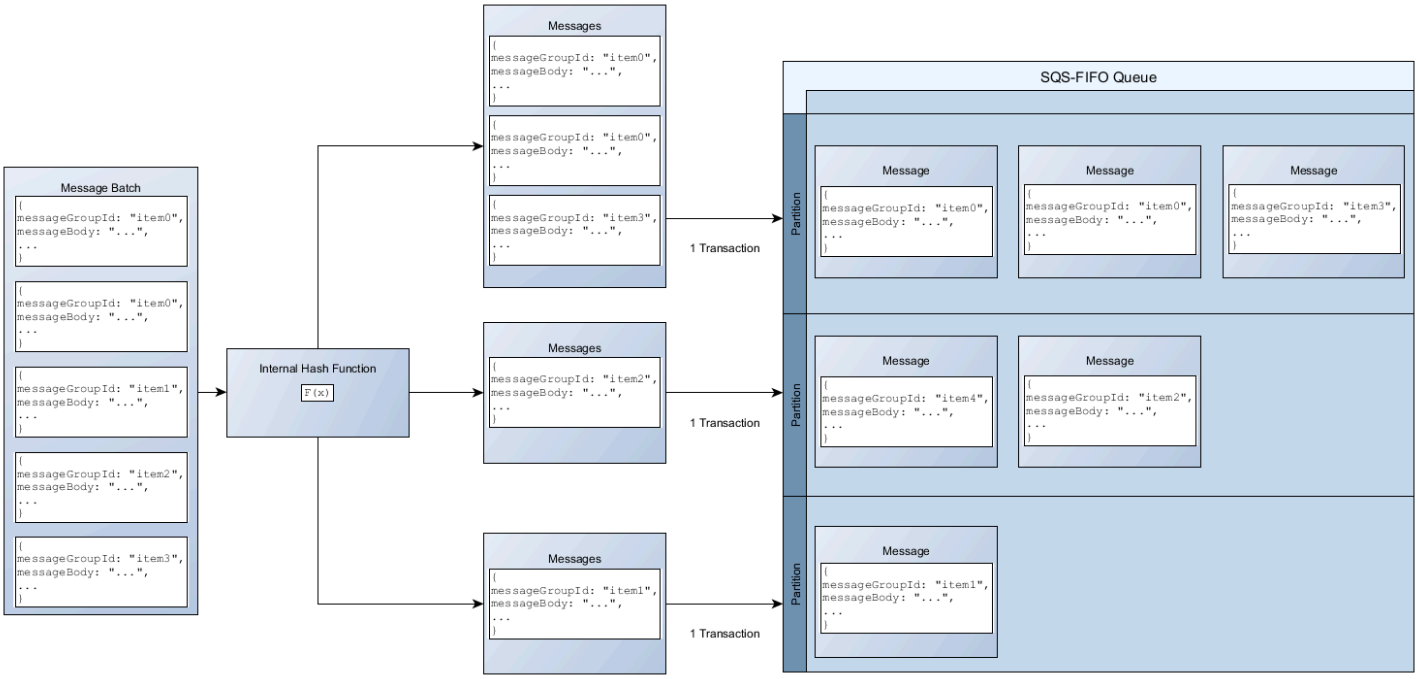
バッチAPI を使用する場合、各メッセージはメッセージグループ ID によるデータの配布で説明されているプロセスに基づいてルーティングされます。同じパーティションにルーティングされたメッセージは、単一のトランザクションでグループ化され、処理されます。

`SendMessageBatch` API のパーティション使用率を最適化するために、 では、可能であれば同じメッセージグループ IDs を持つメッセージのバッチ処理 AWS を推奨します。

`DeleteMessageBatch` および `ChangeMessageVisibilityBatch` APIs のパーティション使用率を最適化するには、 `MaxNumberOfMessages`パラメータを 10 に設定して`ReceiveMessage`リク

エストを使用し、1 つのReceiveMessageリクエストによって返される受信ハンドルをバッチ処理 AWS することをお勧めします。

次の例では、さまざまなメッセージグループ IDを持つメッセージのバッチが送信されます。バッチは3つのグループに分割され、それぞれがパーティションのクォータに対してカウントされます。



> **Note**
>
> Amazon SQS では、同じメッセージグループ ID の内部ハッシュ関数を持つメッセージがバッチリクエスト内でグループ化されている場合にのみ保証されます。内部ハッシュ関数の出力とパーティションの数に応じて、異なるメッセージグループ IDを持つメッセージがグループ化されることがあります。ハッシュ関数またはパーティションの数はいつでも変更できるため、ある時点でグループ化されたメッセージは後でグループ化されない場合があります。

Amazon SQS における FIFO キューの高スループットの有効化

新規または既存の FIFO キューに対して高スループットを有効にできます。この機能には、FIFO キューを作成および編集するときに、次の 3つの新しいオプションが追加されます。

- 高スループット FIFO を有効にする—現在の FIFOキューのメッセージに対し、より高いスループットを有効にします。

- 重複除外スコープ-重複除外をキューレベルまたはメッセージグループレベルのどちらで実行するかを指定します。
- FIFO スループット制限—FIFO キューのメッセージのスループットクォータをキューまたはメッセージグループレベルのどちらで設定するかを指定します。

FIFO キュー(コンソール)の高スループットを有効にするには

1. 作成を起動またはFIFO キューを編集。
2. キューのオプションを指定するときは、高スループット FIFO を有効にする。

   FIFOキューの高スループットを有効にすると、関連するオプションは次のように設定されます。

   - 重複除外スコープは、FIFOキューの高スループットを使用するために必要な設定のメッセージグループ に設定されます。
   - FIFO スループット制限は、FIFO キューの高スループットを使用するために必要な設定のメッセージグループ ID単位 に設定されます。

   FIFO キューの高スループットを使用するために必要な設定のいずれかを変更すると、通常のスループットがキューに対して有効になり、指定どおりに重複除外が実行されます。
3. キューのすべてのオプションをの指定を継続 終了したら、キューの作成または保存を選択します。

FIFOキューを作成または編集した後、メッセージの送信およびメッセージの受信と削除、全てをより高いTPSで行うことができます。高スループットクォータについては、「Amazon SQS のメッセージキュー」の「メッセージスループット」を参照してください。

Amazon SQS の FIFO キューとメッセージの識別子

このセクションでは、FIFO キューの識別子について説明します。これらの識別子は、特定のキューとメッセージを見つけて操作するうえで役立ちます。

Amazon SQS の FIFO キューの識別子

次の識別子の詳細については、Amazon Simpleキューサービス APIリファレンス。を参照してください

キュー名およびURL

新しいキューを作成する際は、 AWS アカウントおよびリージョンに一意なキュー名を指定する必要があります。Amazon SQS は、作成したキューごとにキュー URL と呼ばれる識別子を割り当てます。これには、キュー名と他の Amazon SQS コンポーネントが含まれます。キューでアクションを実行するときは必ず、そのキュー URLを指定します。

FIFOキューの名前は`.fifo`のサフィックスで終わる必要があります。サフィックスは80文字のキュー名クォータにカウントされます。キューがであるかどうかを確認するにはFIFOでは、キュー名の末尾がサフィックスで終わるかどうかでチェックすることがきます。

次に示すのは、AWS アカウント番号 `123456789012` を持つユーザーが所有する `MyQueue` という名前の FIFO キューのキュー URL です。

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue.fifo
```

キューを一覧表示し、アカウント番号に続く文字列を解析することで、キューのURLをプログラムで取得できます。詳細については、「`ListQueues`」を参照してください。

メッセージ ID

各メッセージは、システム割り当てのメッセージ ID を受け取ります。この ID は、Amazon SQS から `SendMessage` レスポンスで返されます。この識別子は、メッセージを特定する場合に役立ちます。メッセージ IDの最大長は100文字です。

受信ハンドル

キューからメッセージを受信するたびに、そのメッセージの受信ハンドルを受け取ります。このハンドルは、メッセージ自体ではなくメッセージ受信のアクションと関連付けられます。メッセージを削除したり、メッセージ可視性を変更したりするには、受信ハンドル (メッセージ ID ではなく) を指定する必要があります。つまり、メッセージを削除する前にメッセージを受信する必要があります (メッセージをキューにおいてから回収することはできません)。受信ハンドルの最大長は1,024文字です。

> **Important**
>
> メッセージを複数回受信した場合、受信するたびに異なる受信ハンドルを受け取ります。メッセージの削除をリクエストするときは、最後に受け取った受信ハンドルを指定してください (そうしないと、メッセージが削除されない可能性があります)。

受信ハンドルの例を次に示します (3行に分割されています)。

```
MbZj6wDWli+JvwwJaBV+3dcjk2YW2vA3+STFFljTM8tJJg6HRG6PYSasuWXPJB+Cw
Lj1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGYWbnLmpRCJVAyeMjeU5ZBdtcQ+QE
auMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/KSbkJ0=
```

Amazon SQS FIFOキューのその他の識別子

次の識別子の詳細については、Amazon SQS の 1 回のみ処理とAmazon Simple キュー サービス API リファレンスを参照してください

メッセージ重複排除ID

メッセージを一意に識別して重複を防ぐために Amazon SQS の FIFO キューで使用するトークン。同じ重複排除 ID を持つ複数のメッセージを 5 分間の重複排除間隔内に送信すると、これらは重複として扱われ、1 つのコピーのみが配信されます。重複排除 ID を指定しない場合、コンテンツベースの重複排除が有効になっていると、Amazon SQS はメッセージ本文をハッシュして重複排除 ID を生成します。このメカニズムは、指定した時間枠内の重複するメッセージを削除することで、正確に 1 回だけ配信されるようにします。

メッセージグループ ID

`MessageGroupId` は、メッセージを個別のグループに整理するために Amazon SQS FIFO (First-In-First-Outキューでのみ使用される属性です。同じメッセージグループ内のメッセージは常に 1 つずつ厳密な順序で処理されるため、同じグループからの 2 つのメッセージが同時に処理されることはありません。標準キューは を使用せず`MessageGroupId`、注文の保証も提供しません。厳密な順序付けが必要な場合は、代わりに FIFO キューを使用します。

シーケンス番号

Amazon SQS が各メッセージに割り当てる、連続しないラージ番号。

Amazon SQS のクォータ

このトピックでは、Amazon SQS FIFO キューと標準キューのクォータと制限について説明し、キューの作成、設定、メッセージ処理にどのように影響するかについて詳しく説明します。メッセージ保持制限、処理中のメッセージ上限、スループットしきい値などの制約と、バッチ処理、API コールの最適化、ロングポーリングを通じて効率を最大化する戦略について説明します。このトピックでは、命名規則、タグ付けルール、および高需要ワークロードに合わせてクォータの引き上げをリクエストする方法についても取り上げ、効果的なキュー管理と最適なパフォーマンスを実現します。

Amazon SQS の FIFO キューのクォータ

Amazon SQS のクォータ

次の表に、FIFO キューに関連するクォータの一覧を示します。

クォータ	説明
遅延キュー	キューのデフォルト (最小) 遅延は0秒です。最大は15分です。
表示されたキュー	`ListQueues` リクエストあたり1,000キュー。
ロングポーリングの待ち時間	ロングポーリングの最大待機時間は 20 秒です。
メッセージグループ	FIFOキュー内のメッセージグループの数にクォータはありません。
キューあたりのメッセージ (バックログ)	Amazon SQSキューに保存できるメッセージ数は無制限です。
キューあたりのメッセージ (インフライト)	FIFO キューは、最大 120,000 件のインフライトメッセージ (コンシューマーが受信したがまだ削除していないメッセージ) をサポートします。この制限に達すると、Amazon SQS はエラーを返しませんが、処理が影響を受ける可能性があります。AWS サポートに連絡して、この制限を超える引き上げをリクエストできます。

クォータ	説明
キュー名	FIFOキューの名前は`.fifo`のサフィックスで終わる必要があります。サフィックスは80文字のキュー名クォータにカウントされます。キューがあるかどうかを確認するにはFIFOでは、キュー名の末尾がサフィックスで終わるかどうかをチェックできます。
キューのタグ	50個を超えるタグをキューに追加することはお勧めしません。タグは、UTF-8 の Unicode 文字をサポートします。
	タグKeyは必須ですが、タグValueはオプションです。
	タグKeyとタグValueの大文字と小文字は区別されます。
	タグKeyとタグValueには、UTF-8のUnicode 英数字と空白を含めることができます。次の特殊文字を使用できます。`_ . : / = + - @`
	タグKeyまたはタグValueには、予約済みプレフィックスの`aws:`を含めることはできません (このプレフィックスが付いているタグキーやタグ値は削除できません)。
	タグKeyの最大長はUTF-8で128Unicode文字です。タグKeyを空またはnullにすることはできません。
	タグValueの最大長はUTF-8で256Unicode文字です。タグValueを空またはnullにすることはできません。
	タグ付けアクションは、 あたり 30 TPS に制限されています AWS アカウント。アプリケーションでより高いスループットが必要な場合、リクエストを送信します。。

Amazon SQS 標準キューのクォータ

次の表に、標準キューに関連するクォータの一覧を示します。

クォータ	説明
遅延キュー	キューのデフォルト (最小) 遅延は0秒です。最大は15分です。
表示されたキュー	`ListQueues` リクエストあたり1,000キュー。
ロングポーリングの待ち時間	ロングポーリングの最大待機時間は 20 秒です。
キューあたりのメッセージ (バックログ)	Amazon SQSキューに保存できるメッセージ数は無制限です。
キューあたりのメッセージ (インフライト)	ほとんどの標準キューには (キュートラフィックとメッセージバックログに応じて) 最大約 120,000 のインフライトメッセージ (コンシューマーがキューから受信し、キューからまだ削除していないもの) が存在する可能性があります。ショートポーリングを使用している場合、このクォータに達すると、Amazon SQS は`OverLimit` エラーメッセージをし返信します。ロングポーリングを使用している場合、 Amazon SQSはエラーメッセージを返しません。クォータに到達しないように、処理されたメッセージはキューから削除する必要があります。メッセージの処理に使用するキューの数を増やすこともできます。クォータの引き上げをリクエストするには、サポートリクエストを送信します。
キュー名	キュー名には最大80文字を使用できます。次の文字を使用できます:英数字、ハイフン (-)、およびアンダースコア (_)。 **Note** キューの名前では、大文字と小文字が区別されます (たとえば、`Test-queue` と`test-queue` は異なるキューです)。

クォータ	説明
キューのタグ	50個を超えるタグをキューに追加することはお勧めしません。タグは、UTF-8 の Unicode 文字をサポートします。
	タグKeyは必須ですが、タグValueはオプションです。
	タグKeyとタグValueの大文字と小文字は区別されます。
	タグKeyとタグValueには、UTF-8のUnicode 英数字と空白を含めることができます。次の特殊文字を使用できます。`_ . : / = + - @`
	タグKeyまたはタグValueには、予約済みプレフィックスのaws:を含めることはできません (このプレフィックスが付いているタグキーやタグ値は削除できません)。
	タグKeyの最大長はUTF-8で128Unicode文字です。タグKeyを空またはnullにすることはできません。
	タグValueの最大長はUTF-8で256Unicode文字です。タグValueを空またはnullにすることはできません。
	タグ付けアクションは、 あたり 30 TPS に制限されています AWS アカウント。アプリケーションでより高いスループットが必要な場合、リクエストを送信します。。

Amazon SQS のメッセージキュー

次の表に、メッセージに関連するクォータの一覧を示します。

クォータ	説明
バッチ処理されたメッセージ ID	バッチ処理されたメッセージ ID には最大80文字を使用できます。次の文字を使用できます。英数字、ハイフン (-)、およびアンダースコア (_)。

クォータ	説明
メッセージ属性	メッセージには、最大10個のメタデータ属性を含めることができます。
メッセージバッチ	ひとつのメッセージBatchリクエストに最大10個のメッセージを含めることができます。詳細については、AmazonSQSBufferedAsyncClient の設定 セクションの Amazon SQSのバッチアクション を参照してください。
メッセージの内容	メッセージには、XML、JSON、およびフォーマットされていないテキストのみを含めることができます。次の Unicode 文字を使用できます。#x9 | #xA | #xD | #x20から#xD7FF | #xE000から#xFFFD | #x10000から#x10FFFF この一覧に含まれていない文字は、すべて拒否されます。詳細については、文字に関するW3C仕様を参照してください。
メッセージグループ ID	MessageGroupId はFIFOキューには必須です。FIFO キューにメッセージを送信MessageGroupId するときに を指定しない場合、アクションは失敗します。標準キューでは、 を使用すると公平キューMessageGroupId が有効になります。 ???公平キューを使用する場合は、すべてのメッセージMessageGroupId に を含めることをお勧めします。 MessageGroupId の最大長は128文字です。有効な値: 英数字と句読点(!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~) 。
メッセージの保持	デフォルトでは、メッセージは4日間保持されます。最小の期間は60秒(1分) です。最大は1,209,600秒(14日)です。

クォータ	説明
メッセージのスループット	標準キュー 標準キューは、API アクション (SendMessage 、ReceiveMessage 、または DeleteMessage) ごとに 1 秒あたり非常に多い、ほぼ無制限の API コール数をサポートします。この高スループットにより、リアルタイムのデータストリーミングや大規模なアプリケーションなど、大量のメッセージをすばやく処理する必要があるユースケースに最適です。標準キューは需要に応じて自動的にスケールされますが、特にワークロードの高いリージョンでは、最適なパフォーマンスを確保するために使用パターンをモニタリングすることが不可欠です。

クォータ	説明
	FIFO キュー • FIFO キュー内の各パーティションは、API アクション (`SendMessage` 、`ReceiveMessage` 、`DeleteMessage`) ごとに 1 秒あたり 300 件のトランザクションに制限されます。この制限は、特に高スループット以外のモードに適用されます。高スループットモードに切り替えることで、このデフォルトの制限を超えることができます。高スループットモードを有効にするには、「Amazon SQS における FIFO キューの高スループットの有効化」を参照してください。 • バッチ処理を使用すると、FIFO キューは API アクション (`SendMessage` 、`ReceiveMessage` 、`DeleteMessage`) ごとに 1 秒あたり最大 3,000 件のメッセージをサポートします。1 秒あたり 3,000 件のメッセージは、API コールごとのメッセージ数を 10 件として、300 件の API コールを表します。 FIFOキューの高スループット Amazon SQS の FIFO 制限は、メッセージ制限ではなく、API リクエスト数に基づきます。高スループットモードでは、これらの API リクエスト制限は以下のとおりです。 トランザクションスループットの制限 (非バッチ処理 API コール) これらの制限は、各 API オペレーション (SendMessage、ReceiveMessage、DeleteMessage など) を個別に実行できる頻度を定義し、許可された 1 秒あたりのトランザクション数 (TPS) 内で効率的なシステムパフォーマンスを確保します。 以下の制限は、非バッチ処理 API コールに基づいています。

クォータ	説明
	• 米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド): 最大 70,000 のトランザクション/秒 (TPS)。 • 米国東部 (オハイオ) と欧州 (フランクフルト): 最大 19,000 TPS。 • アジアパシフィック (ムンバイ)、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): 最大 9,000 TPS。 • 欧州 (ロンドン) と南米 (サンパウロ): 最大 4,500 TPS。 • その他すべて AWS リージョン: デフォルトのスループットは 2,400 TPS です。 バッチ処理によるスループットの最大化 1 回の API コールで複数のメッセージを処理するため、効率が大幅に向上します。バッチ処理では、各メッセージを個別に処理する代わりに、1 つの API リクエストで最大 10 件のメッセージを送信、受信、または削除できます。これにより、API コールの総数が減少し、リージョンのトランザクション制限 (TPS) 内に留まりながら 1 秒ごとにより多くのメッセージを処理できるため、スループットとシステムパフォーマンスを最大化できます。詳細については、「Amazon SQS での水平スケーリングとアクションのバッチ処理を使用したスループットの向上」を参照してください。 以下の制限は、バッチ処理 API コールに基づいています。 • 米国東部 (バージニア北部)、米国西部 (オレゴン)、欧州 (アイルランド): 1 秒あたり最大 700,000 のメッセージ (非バッチ処理の制限である 70,000 TPS の 10 倍)。

クォータ	説明
	• 米国東部 (オハイオ) と欧州 (フランクフルト): 1 秒あたり最大 190,000 のメッセージ。 • アジアパシフィック (ムンバイ)、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): 1 秒あたり 90,000 のメッセージ。 • 欧州 (ロンドン) と南米 (サンパウロ): 1 秒あたり最大 45,000 のメッセージ。 • その他すべて AWS リージョン: 1 秒あたり最大 24,000 メッセージ。 バッチ処理を超えたスループットの最適化 バッチ処理はスループットを大幅に向上させますが、FIFO パフォーマンスを最適化する他の戦略を検討することが重要です。 • 複数のメッセージグループ ID へのメッセージの分散 – 1 つのグループ内のメッセージは順番に処理されるため、ワークロードを複数のメッセージグループに分散すると、並列処理と全体的なスループットが向上します。詳細については、「SQS FIFOキュー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」を参照してください。 • API コールの効率的な使用 – 頻繁な可視性の変更やメッセージの削除の繰り返しなど、不要な API コールを最小限に抑えて、利用可能な TPS の使用を最適化し、効率を向上させます。 • ロングポーリング受信の使用 – 受信リクエストで `WaitTimeSeconds` を設定することでロングポーリングを使用し、メッセージがない場合の空のレスポンスを減らし、不要な API コールを減らして、TPS クォータをより有効に活用します。

クォータ	説明
	• スループット引き上げのリクエスト – アプリケーションでデフォルトの制限を超えるスループットを必要とする場合は、Service Quotas コンソールを使用して引き上げをリクエストします。これは、需要の高いワークロードや、デフォルト制限の低いリージョンで必要になる場合があります。高スループットモードを有効にするには、「Amazon SQS における FIFO キューの高スループットの有効化」を参照してください。
メッセージタイマー	メッセージのデフォルト (最小) 遅延は0秒です。最大は15分です。
メッセージサイズ	最小メッセージサイズは1バイト(1文字) です。最大は 1,048,576 バイト (1 MiB) です。 1 MiB を超えるメッセージを送信するには、Java 用 Amazon SQS 拡張クライアントライブラリと Python 用 Amazon SQS 拡張クライアントライブラリを使用できます。このライブラリでは、 Amazon SQSのメッセージペイロードAmazon S3へのリファレンスを含むメッセージを送信できます。最大ペイロードサイズは2GBです。 Note この拡張ライブラリは、同期クライアントでのみ機能します。
メッセージ可視性タイムアウト	デフォルトの可視性タイムアウトは30秒です。最小は0秒です。最大は12時間です。
ポリシー情報	最大クォータは8,192バイト、20ステートメント、50プリンシパル、または 10条件になります。詳細については、「Amazon SQS ポリシーのクォータ」を参照してください。

Amazon SQS ポリシーのクォータ

次の表に、ポリシーに関連するクォータの一覧を示します。

名前	最大値
バイト	8,192
条件	10
プリンシパル	50
ステートメント	20
ステートメントあたりのアクション数	7

Amazon SQS の特徴と機能

このトピックでは、メッセージキューの管理、パフォーマンスの最適化、信頼性の高いメッセージ配信の確保、およびメッセージ処理の効率化のために、Amazon SQS でよく使用される機能について説明します。

Amazon SQS でのデッドレターキューの使用

Amazon SQS は、デッドレターキュー (DLQ) をサポートしています。これは、正常に処理されなかったメッセージをソースキューから送信する先のターゲットキューとなります。DLQ を使用すると、未消費のメッセージを分離して処理が成功しなかった理由を判断できるため、アプリケーションのデバッグに役立ちます。パフォーマンスを最適化するには、ソースキューと DLQ を同じ AWS アカウント およびリージョン内に維持することがベストプラクティスです。メッセージがデッドレターキューに入ったら、以下の操作を実行できます。

- デッドレターキューにメッセージが移動する原因となったと思われる例外をログで調べる。
- デッドレターキューに移動したメッセージの内容を分析してアプリケーションの問題を診断する。
- コンシューマーがメッセージを処理するための十分な時間を割り当てられていたかどうかを判断する。
- デッドレターキューの再処理を使用して、デッドレターキューからメッセージを移動する。

最初にキューを作成してから、これをデッドレターキューとして設定する必要があります。Amazon SQS コンソールを使用してデッドレターキューを設定する方法の詳細については、「Amazon SQS コンソールを使用してデッドレターキューを設定する」を参照してください。デッドレターキューに移動したメッセージにアラームを設定する方法など、デッドレターキューのヘルプについては、「Amazon CloudWatch を使用したデッドレターキューのアラームの作成」を参照してください。

> **Note**
>
> メッセージまたは操作の正確な順序を維持する必要がある場合は、FIFOキューでデッドレターキューを使用しないでください。たとえば、ビデオ編集スイートで Edit Decision List (EDL)の指示でデッドレターキューを使用しないでください。編集の順序が変わると、後続の編集作業のコンテキストが変わります。

デッドレターキューのポリシーの使用

再処理ポリシーを使用して maxReceiveCount を指定します。maxReceiveCount は、メッセージがデッドレターキューに移動するまでに、コンシューマがソースキューからメッセージを受信できる回数です。例えば、maxReceiveCount を 1 などの低い値に設定した場合、メッセージの受信に 1 回失敗すると、メッセージはデッドレターキューに移動します。システムがエラーに対して回復力を持つようにするには、十分な再試行ができるように maxReceiveCount を高めに設定してください。

[許可ポリシーの再実行] は、デッドレターキューにアクセスできるソースキューを指定します。デッドレターキューの使用を、すべてのソースキューを許可するか、特定のソースキューを許可するか、すべてのソースキューに拒否するかを選択できます。デフォルトでは、デッドレターキューの使用をすべてのソースキューに許可します。byQueue オプションを使用して特定のキューを許可することを選択する場合は、ソースキューの Amazon リソースネーム (ARN)を使用して最大 10 個のソースキューを指定できます。denyAll を指定すると、キューをデッドレターキューとして使用することはできません。

デッドレターキューのメッセージ保持期間を理解する

標準キューの場合、メッセージの有効期限は常に元のエンキューのタイムスタンプに基づきます。デッドレターキューに移動すると、エンキューのタイムスタンプは変更されません。ApproximateAgeOfOldestMessage メトリクスは、メッセージが最初に送信されたときではなく、メッセージがデッドレターキューに移動したときを示します。たとえば、メッセージがデッドレターキューに移動される前に、元のキューで1日費やすと仮定します。デッドレターキューの保持期間が4日間である場合、メッセージは3日後にデッドレターキューから削除され、ApproximateAgeOfOldestMessageは3日間です。したがって、デッドレターキューの保持期間を、元のキューの保持期間よりも長く設定することがベストプラクティスです。

FIFO キューでは、メッセージがデッドレターキューに移動すると、エンキューのタイムスタンプがリセットされます。ApproximateAgeOfOldestMessage メトリクスは、メッセージがデッドレターキューに移動した日を示します。上記の同じ例では、メッセージは 4 日後にデッドレターキューから削除され、 ApproximateAgeOfOldestMessageは 4 日間です。

Amazon SQS コンソールを使用してデッドレターキューを設定する

デッドレターキュー (DLQ) は、ソースキューと呼ばれる別のキューから正常に処理されなかったメッセージを受信するキューです。Amazon SQSはデッドレターキューを自動的に作成しません。デッドレターキューとして使用する前に、最初にキューを作成する必要があります。DLQ を設定す

る場合、キュータイプはソースキュータイプと一致する必要があります。FIFO キューは FIFO DLQ のみを使用し、標準キューは標準 DLQ のみを使用できます。デッドレターキューは、キューを作成または編集するときに設定できます。詳細については、「Amazon SQS でのデッドレターキューの使用 」を参照してください。

既存のキュー(コンソール)のデッドレターキューを設定する

1. Amazon SQSコンソールを開きますhttps://console.aws.amazon.com/sqs/。
2. ナビゲーションペインで [キュー] を選択します。
3. ソースキュー (失敗したメッセージをデッドレターキューに送信するキュー) を選択し、編集を選択します。
4. デッドレターキューセクションまでスクロールし、有効に切り替えます。
5. デッドレターキュー設定で、デッドレターキューとして使用する既存のキューの Amazon リソースネーム (ARN) を選択します。
6. 最大受信値を設定します。この値は、デッドレターキューに送信されるまでにメッセージを受信できる回数を定義します (有効な範囲: 1 ~ 1,000) 。
7. [保存] を選択します。

Amazon SQS でデッドレターキューの再処理を設定する方法について説明します。

デッドレターキューリドライブを使用して、未使用のメッセージをデッドレターキューから別の宛先に移動して処理します。デフォルトでは、デッドレターキューの再処理は、デッドレターキューからソースキューにメッセージを移動します。ただし、どちらのキューも同じタイプであれば、他の任意のキューを再処理の送信先として設定することもできます。例えば、デッドレターキューが FIFO キューの場合、再処理の送信先キューも FIFO キューでなければなりません。さらに、再処理の速度を設定して Amazon SQS がメッセージを移動するレートを設定することもできます。

> **Note**
>
> メッセージが FIFO キューから FIFO DLQ に移動されると、元のメッセージの重複排除 ID は元のメッセージの ID に置き換えられます。これは、重複排除 ID を共有している 2 つの独立したメッセージが DLQ 重複排除によって保存されなくなることがないようにするためです。

デッドレターキューは、メッセージを受信順に、最も古いものから再処理します。ただし、送信先キューは、再処理されたメッセージと他のプロデューサーからの新しいメッセージを、受信順に取り込みます。例えば、プロデューサーがソース FIFO キューにメッセージを送信しているときに、デッドレターキューから再処理されたメッセージを同時に受信した場合、再処理されたメッセージはプロデューサーからの新しいメッセージと混在します。

> **Note**
>
> 再処理タスクにより、保持期間がリセットされます。すべての再処理されたメッセージは、新しい `messageID` を持つ新しいメッセージと見なされ、再処理されたメッセージに `enqueueTime` が割り当てられます。

Amazon SQS API を使用して既存の標準キューにデッドレターキューの再処理を設定する

デッドレターキューの再処理は、`StartMessageMoveTask`、`ListMessageMoveTasks`、`CancelMessageMoveTask` API アクションを使用して設定できます。

API アクション	説明
`StartMessageMoveTask`	指定されたソースキューから指定された送信先キューにメッセージを移動する非同期タスクを開始します。
`ListMessageMoveTasks`	特定のソースキューにある最新のメッセージ移動タスク (最大 10 個) を取得します。
`CancelMessageMoveTask`	指定されたメッセージ移動タスクをキャンセルします。メッセージの移動は、現在のステータスが RUNNING の場合にのみキャンセルできます。

Amazon SQS コンソールを使用して既存の標準キューにデッドレターキューの再処理を設定する

1. Amazon SQSコンソール (https://console.aws.amazon.com/sqs/) を開きます。
2. ナビゲーションペインで [Queues(キュー)] を選択します。
3. デッドレターキューとして設定されたキューの名前を選択します。
4. [DLQ 再処理の開始] を選択します。
5. [設定を再処理] の [メッセージの送信先] で、次のいずれかを実行します。
   - 再処理のためにメッセージをソースキューに入れるには、[再処理のためにソースキューに入れる] を選択します。
   - 再処理のためにメッセージを別のキューに入れるには、[再処理のためにカスタム送信先に移動] を選択します。次に、既存の宛先キューのAmazonリソースネーム (ARNを入力します。
6. Velocityコントロール設定で、次のいずれかを選択します。
   - システム最適化 - デッドレターキューメッセージを 1 秒あたり最大のメッセージ数で再処理します。
   - カスタム最大速度 - 1 秒あたりカスタム最大レートのメッセージ数でデッドレターキューメッセージを再処理します。許可される最大レートは1秒に500 メッセージです。
     - カスタム最大速度を小さい値から始めて、ソースキューがメッセージでいっぱいにならないことを確認することをお勧めします。そこから、ソースキューの状態を引き続き監視しながら、カスタム最大速度の値を徐々に上げていきます。
7. デッドレターキューの再処理の設定が完了したら、[メッセージをリドライブ] を選択します。

   > **Important**
   >
   > Amazon SQS は、デッドレターキューからメッセージを再処理するときにメッセージのフィルタリングと変更をサポートしていません。
   > デッドレターキューの再処理タスクは、最大 36 時間実行できます。Amazon SQS は、アカウントごとに最大 100 件のアクティブな再処理タスクをサポートしています。

8. メッセージの再処理タスクをキャンセルする場合は、キューの [詳細] ページで、[DLQ 再処理をキャンセル] を選択します。進行中のメッセージの再処理をキャンセルしても、移動先キューに正常に移動済みのメッセージは、移動先キューに残ります。

デッドレターキューの再処理に対するキューのアクセス許可の設定

ポリシーにアクセス許可を追加することで、ユーザーに特定のデッドレターキューアクションへのアクセスを許可できます。デッドレターキューの再処理に最低限必要なアクセス許可は次のとおりです。

最小限必要なアクセス権限	必要な API メソッド
メッセージの再処理を開始するには	• デッドレターキューの `sqs:StartMessageMoveTask` 、`sqs:ReceiveMessage` 、`sqs:DeleteMessage` 、`sqs:GetQueueAttributes` を追加します。デッドレターキューまたは元のソースキュー (SSE キューとも呼ばれます) のいずれかが暗号化されている場合、`kms:Decrypt` メッセージの暗号化に使用されたすべての KMS キーも必要になります。 • 送信先キュー `sqs:SendMessage` を追加します。送信先キューが暗号化されている場合、`kms:GenerateDataKey` と `kms:Decrypt` も必要になります。
進行中のメッセージの再処理をキャンセルするには	• デッドレターキューの `sqs:CancelMessageMoveTask` 、`sqs:ReceiveMessage` 、`sqs:DeleteMessage` 、`sqs:GetQueueAttributes` を追加します。デッドレターキューが暗号化されている場合 (SSE キューとも呼ばれる)、`kms:Decrypt` も必要になります。
メッセージの移動状況を表示するには	• デッドレターキューの `sqs:ListMessageMoveTasks` と `sqs:GetQueueAttributes` を追加します。

暗号化されたキューペア (デッドレターキューのあるソースキュー) のアクセス許可を設定するには

次の手順を使用して、デッドレターキュー (DLQ) リドライブの最小アクセス許可を設定します。

1. IAM コンソール (https://console.aws.amazon.com/iam/) 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. 新しいポリシーを作成し、次のアクセス許可を追加します。再処理オペレーションを実行する IAM ユーザーまたはロールにポリシーをアタッチします。

   - DLQ のアクセス許可 (ソースキュー):
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:GetQueueAttributes`
     - `sqs:ListDeadLetterSourceQueues`
     - DLQ (ソースキュー) のリソース ARN を指定します (例: "arn:aws:sqs:*<DLQ_region>*：*<DLQ_accountId>*：*<DLQ_name>*")。
   - 送信先キューのアクセス許可：
     - `sqs:SendMessage`
     - 送信先キューResource ARNの を指定します (例: "arn:aws:sqs:*<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*")。
   - KMS キーのアクセス許可：
     - `kms:Decrypt` (DLQ でメッセージを復号化する必要があります) 。
     - `kms:GenerateDataKey` (送信先キュー内のメッセージを暗号化する必要があります) 。
       - `Resource` ARNs:
         - DLQ (ソースキュー) 内のメッセージの暗号化に使用される KMS キーの ARN (例: "arn:aws:kms:*<region>*：*<accountId>*:key/*<SourceQueueKeyId>*")。
         - 送信先キュー内のメッセージの暗号化に使用される KMS キーの ARN (例: "arn:aws:kms:*<region>*：*<accountId>*:key/*<DestinationQueueKeyId>*")。

アクセスポリシーは以下のようになります。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "sqs:StartMessageMoveTask",
                "sqs:CancelMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:ListDeadLetterSourceQueues"
            ],
            "Resource":
 "arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "source"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource":
 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "destination"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": [
                "arn:aws:kms:<region>:<accountId>:key/<SourceQueueKeyId>",
                "arn:aws:kms:<region>:<accountId>:key/<DestQueueKeyId>"
            ]
        }
    ]
}
```

暗号化されていないキューペア (デッドレターキューのあるソースキュー) を使用してアクセス許可を設定するには

以下の手順に従って、標準の暗号化されていないデッドレターキュー (DLQ) の処理に必要な最小限のアクセス許可を設定します。最低限必要なアクセス許可は、デッドレターキューからの属性の受信、削除、取得、およびソースキューへの属性の送信です。

1. IAM コンソール (https://console.aws.amazon.com/iam/) 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. 新しいポリシーを作成し、次のアクセス許可を追加します。再処理オペレーションを実行する IAM ユーザーまたはロールにポリシーをアタッチします。

   - DLQ のアクセス許可 (ソースキュー):
     - `sqs:StartMessageMoveTask`
     - `sqs:CancelMessageMoveTask`
     - `sqs:ListMessageMoveTasks`
     - `sqs:ReceiveMessage`
     - `sqs:DeleteMessage`
     - `sqs:ListDeadLetterSourceQueues`
     - DLQ (ソースキュー) のリソース ARN を指定します (例: "arn:aws:sqs:*<DLQ_region>*：*<DLQ_accountId>*：*<DLQ_name>*")。
   - 送信先キューのアクセス許可：
     - `sqs:SendMessage`
     - 送信先キューResource ARNの を指定します (例: "arn:aws:sqs:*<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*")。

   アクセスポリシーは以下のようになります。

   JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "sqs:StartMessageMoveTask",
                "sqs:CancelMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:ListDeadLetterSourceQueues"
            ],
            "Resource":
 "arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "source"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource":
 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/QueueRole": "destination"
                }
            }
        }
    ]
}
```

VPC エンドポイントアクセスコントロールでのデッドレターキューリドライブの使用

`aws:sourceVpc` 条件を使用して特定の VPCs へのキューアクセスを制限する場合、デッドレターキュー (DLQ) の再処理機能を有効にするには、 AWS サービスの例外を作成する必要があります。これは、メッセージを移動するときに Amazon SQS サービスが VPC の外部で動作するためです。

DLQ 再処理オペレーションを許可するには、キューポリシーに `aws:CalledViaLast`条件を追加します。これにより、Amazon SQS は直接アクセスの VPC 制限を維持しながら、ユーザーに代わって API コールを行うことができます。

VPC 制限付きアクセスと DLQ リドライブの両方を許可するには：

1. キューポリシーで `aws:CalledViaLast`条件を使用します。
2. ソースキューと DLQ の両方にポリシーを適用する
3. 他のソースからの直接アクセスに関する VPC 制限を維持する

これらの要件を実装するポリシーの例を次に示します。

```
{
  "Version": "2012-10-17",
  "Id": "SQSRedriveWithVpcRestriction",
  "Statement": [
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DestQueue",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:DestQueue",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "StringNotEqualsIfExists": {
          "aws:CalledViaLast": "sqs.amazonaws.com"
        }
      }
    },
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DLQ",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:Dlq",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "StringNotEqualsIfExists": {
          "aws:CalledViaLast": "sqs.amazonaws.com"
        }
      }
    }
  ]
```

```
}
```

- プレースホルダー値を実際の値に置き換えます。
- このポリシーは、条件付きの「拒否」ステートメントを使用します。これは「許可」ステートメントを使用するよりも安全です。
- `StringNotEqualsIfExists` 演算子は、条件キーがリクエストコンテキストに存在しない場合を処理します。

または、 `aws:ViaAWSService`条件キーを使用して、VPC 制限を維持しながらサービスベースのアクセスを許可することもできます。この条件キーは、リクエストが AWS サービスから送信されたかどうかを示します。`aws:ViaAWSService` の代わりに を使用するポリシーの例を次に示します`aws:CalledViaLast`。

```
{
  "Version": "2012-10-17",
  "Id": "SQSRedriveWithVpcRestriction",
  "Statement": [
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DestQueue",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:DestQueue",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    },
    {
      "Sid": "DenyOutsideVPCUnlessAWSService_DLQ",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:Dlq",
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-1234567890abcdef0"
```

```
        },
        "BoolIfExists": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

`aws:ViaAWSService` 条件付きの BoolIfExists 演算子を使用すると、直接アクセスの VPC 制限を維持しながら、サービスからのリクエストが許可されます。これは、最後の呼び出しを行った AWS サービスをチェックするのではなく、 サービスによってリクエストが行われたかどうかを直接チェックするため、理解と保守が簡単になる場合があります。

IAM ポリシーとリソースポリシーで使用される条件キーの詳細については、「IAM JSON ポリシー要素: Condition」を参照してください。

Amazon SQS デッドレターキューの再処理に関する CloudTrail の更新とアクセス許可の要件

2023 年 6 月 8 日、Amazon SQS は SDK および (CLI) の AWS デッドレターキュー AWS Command Line Interface (DLQ) リドライブを導入しました。この機能は、 AWS コンソールで既にサポートされている DLQ リドライブに追加されています。以前に AWS コンソールを使用してデッドレターキューメッセージをリドライブしたことがある場合は、次の変更の影響を受ける可能性があります。

CloudTrail イベントの名前変更

2023 年 10 月 15 日、デッドレターキューの再処理向けの CloudTrail イベント名が Amazon SQS コンソールで変更されます。これらの CloudTrail イベントにアラームを設定している場合は、今すぐ更新する必要があります。DLQ 再処理向けの新しい CloudTrail イベント名は以下のとおりです。

以前のイベント名	新しいイベント名
`CreateMoveTask`	`StartMessageMoveTask`
`CancelMoveTask`	`CancelMessageMoveTask`

アクセス許可の更新

SDK および CLI リリースに含まれている Amazon SQS では、セキュリティのベストプラクティスに準拠するために、DLQ 再処理のキューのアクセス許可も更新しています。DLQ のメッセージを再処理するには、以下のキューのアクセス許可タイプを使用します。

1. アクションベースのアクセス許可 (DLQ API アクション用に更新)
2. Amazon SQS のマネージドポリシーのアクセス許可
3. sqs:* ワイルドカードを使用するアクセス許可ポリシー

> **Important**
>
> SDK または CLI で DLQ 再処理を使用するには、上記のオプションのいずれかと一致する DLQ 再処理のアクセス許可ポリシーが必要です。

DLQ 再処理のキューのアクセス許可が上記のオプションのいずれとも一致しない場合は、2023 年 8 月 31 日までにアクセス許可を更新する必要があります。現在から 2023 年 8 月 31 日までは、以前に DLQ 再処理を使用していたリージョンでのみ、 AWS コンソールで設定したアクセス許可を使用して、アカウントでメッセージを再処理できます。例えば、us-east-1 と eu-west-1 の両方に「アカウント A」があるとします。「アカウント A」は、2023 年 6 月 8 日より前に us-east-1 の AWS コンソールでメッセージを再処理するために使用されましたが、eu-west-1 では使用されていません。2023 年 6 月 8 日から 2023 年 8 月 31 日の間、「アカウント A の」ポリシーのアクセス許可が上記のオプションのいずれかと一致しない場合、us-east-1 の AWS コンソールでメッセージを再処理するためにのみ使用でき、eu-west-1 では使用できません。

> **Important**
>
> 2023 年 8 月 31 日が過ぎても DLQ 再処理のアクセス許可が上記のオプションのいずれとも一致しない場合、アカウントでは AWS コンソールを使用して DLQ メッセージを再処理できなくなります。
> ただし、2023 年 8 月に AWS コンソールで DLQ リドライブ機能を使用した場合は、2023 年 10 月 15 日まで拡張機能を使用して、これらのオプションのいずれかに従って新しいアクセス許可を採用できます。
> 詳細については、「the section called “影響を受けるポリシーの特定”」を参照してください。

各 DLQ 再処理オプションのキューのアクセス許可の例を以下に示します。サーバー側の暗号化 (SSE) キューを使用する場合は、対応する AWS KMS キーのアクセス許可が必要です。

アクションベース

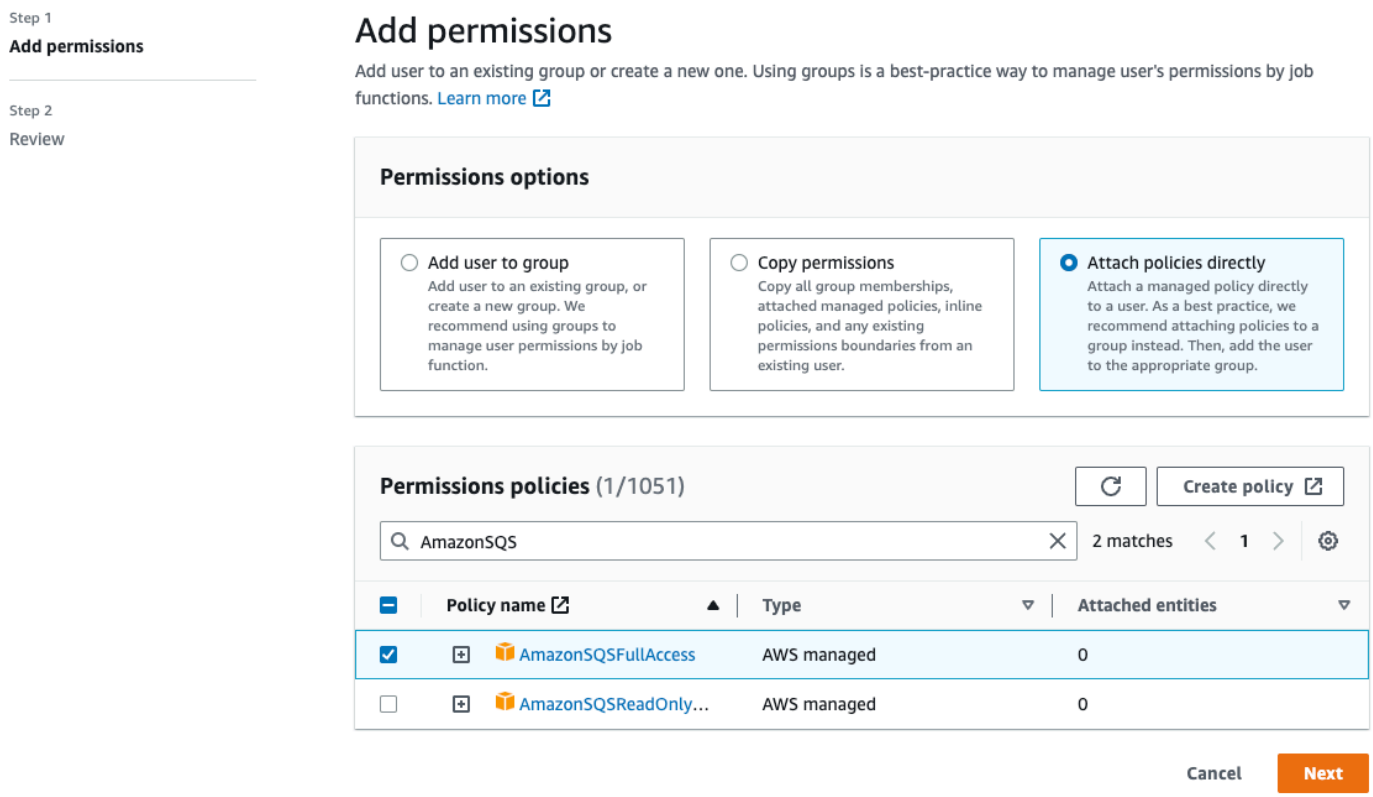
JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage",
                "sqs:DeleteMessage",
                "sqs:GetQueueAttributes",
                "sqs:StartMessageMoveTask",
                "sqs:ListMessageMoveTasks",
                "sqs:CancelMessageMoveTask"
            ],
            "Resource": "arn:aws:sqs:<DLQ_region>:<DLQ_accountId>:<DLQ_name>"
        },
        {
            "Effect": "Allow",
            "Action": "sqs:SendMessage",
            "Resource":
 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>"
        }
    ]
}
```

マネージドポリシー

以下のマネージドポリシーには、必要なアクセス許可の更新が含まれています。

- AmazonSQSFullAccess – デッドレターキューの再処理タスクとして、開始、キャンセル、リストが含まれています。
- AmazonSQSReadOnlyAccess – 読み取り専用アクセスを提供し、デッドレターキューの再処理タスクとしてリストが含まれています。



sqs* ワイルドカードを使用するアクセス許可ポリシー

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "*"
    }
  ]
}
```

影響を受けるポリシーの特定

カスタマー管理ポリシー (CMPs) を使用している場合は、 AWS CloudTrail と IAM を使用して、キューのアクセス許可の更新の影響を受けるポリシーを特定できます。

> **Note**
>
> `AmazonSQSFullAccess` と `AmazonSQSReadOnlyAccess` を使用している場合、それ以上のアクションは必要ありません。

1. AWS CloudTrail コンソールにサインインします。
2. [イベント履歴] ページの [ルックアップ属性] で、ドロップダウンメニューを使用して [イベント名] を選択します。次に、`CreateMoveTask` を検索します。
3. イベントを選択して、[詳細] ページを開きます。[イベントレコード] セクションで、`userIdentity` ARN から `UserName` または `RoleName` を取得します。
4. IAM コンソールにサインインします。
   - ユーザーの場合は、[ユーザー] を選択します。前のステップで特定した `UserName` を持つユーザーを選択します。
   - ロールの場合は、[ロール] を選択します。前のステップで特定した `RoleName` を持つユーザーを検索します。
5. [詳細] ページの [アクセス許可] セクションで、`sqs:` プレフィックスを持つポリシーを参照するか、`Resource` で Amazon SQS キューを定義したポリシーを参照します。

Amazon CloudWatch を使用したデッドレターキューのアラームの作成

`ApproximateNumberOfMessagesVisible` メトリクスを使用してデッドレターキュー内のメッセージをモニタリングするように CloudWatch アラームを設定します。詳細な手順については、「Amazon SQSメトリクスの CloudWatchアラームを作成する」を参照してください。メッセージがデッドレターキューに移動されたことを示すアラームがトリガーされると、キューをポーリングして確認および取得できます。

Amazon SQS のメッセージメタデータ

メッセージ属性を使用して、アプリケーションの Amazon SQS メッセージにカスタムメタデータを追加します。メッセージシステム属性を使用して AWS のサービス、 などの他の と統合するためのメタデータを保存します AWS X-Ray。

Amazon SQSメッセージ属性

Amazon SQS では、メッセージ属性を使用して、構造化メタデータ (タイムスタンプ、地理空間データ、署名、識別子など) をメッセージに含めることができます。各メッセージには最大10個の属性を指定できます。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 (ただし、メッセージ本文とともに送信されます)。コンシューマーはメッセージ属性を使用して、最初にメッセージ本文を処理することなく、特定の方法でメッセージを処理できます。Amazon SQSコンソールを使用して属性とメッセージを送信する方法については、「Amazon SQS を使用した属性を含むメッセージの送信」を参照してください。

> **Note**
>
> メッセージ属性をメッセージシステム属性と混同しないでください。メッセージ属性を使用してアプリケーションの Amazon SQS メッセージにカスタムメタデータをアタッチできますが、メッセージシステム属性を使用して、 などの他の AWS サービスのメタデータを保存できます AWS X-Ray。

トピック

- メッセージ属性コンポーネント
- メッセージ属性のデータ型
- メッセージ属性のMD5メッセージダイジェストの計算

メッセージ属性コンポーネント

> **Important**
>
> メッセージ属性のすべてのコンポーネントは、1 MiB のメッセージサイズ制限に含まれています。
>
> `Name`、`Type`、`Value`、およびメッセージ本文を空または Nullにすることはできません。

各メッセージ属性は、次のコンポーネント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。以下の制限が適用されます。

  - 最大256文字です
  - AWS.またはAmazon.(大文字小文字が異なるものを含む) でスタートすることはできません
  - 大文字と小文字を区別します
  - メッセージのすべての属性名で一意である必要があります
  - 先頭と末尾をピリオドにすることはできません
  - シーケンスにピリオドを含めることはできません
- タイプ-メッセージ属性のデータタイプ サポートされるタイプには String、Number、Binary などがあります。任意のデータ型のカスタム情報を追加することもできます。データタイプには、メッセージ本文と同じ制限があります (詳細については、 SendMessageの「Amazon Simple キューサービス API リファレンス」を参照してください)。また、以下の制限も適用されます。
  - 最大256文字です
  - 大文字と小文字を区別します
- 値-メッセージ属性値。Stringデータ型の場合、属性値の値にはメッセージ本文と同じ制限があります。

メッセージ属性のデータ型

メッセージ属性のデータ型により、対応するメッセージ属性値を処理する方法が Amazon SQS に指示されます。たとえば、タイプが Numberの場合、 Amazon SQSは数値を検証します。

Amazon SQS では、String、Number、Binary の各論理データタイプと、オプションの*.custom-data-type*形式のカスタムデータタイプラベルをサポートしています。

- 文字列-String属性はすべての有効なXML文字を使用してUnicodeテキストを保存できます。
- 数値-Number属性には、正または負の数値が保存できます。数値は最大38桁の精度で、10^-128から10^+126までの間とします。

> **Note**
>
> Amazon SQSでは先頭および末尾の0は削除されます。

- バイナリ-バイナリ属性には、圧縮データ、暗号化データ、イメージなど、すべてのバイナリデータが保存できます。
- カスタム-カスタムデータ型を作成するには、任意のデータ型にカスタム型ラベルを追加します。例:

- `Number.byte`、`Number.short`、`Number.int`、および `Number.float` は、数値型の区別ができます。
- `Binary.gif` および `Binary.png` は、ファイルタイプの区別ができます。

> **Note**
> Amazon SQSが、追加されたデータを解釈、検証、または使用することはありません。カスタム型ラベルには、メッセージ本文と同じ制限があります。

メッセージ属性のMD5メッセージダイジェストの計算

を使用する場合は AWS SDK for Java、このセクションをスキップできます。SDK for Javaの `MessageMD5ChecksumHandler` クラスでは、Amazon SQS メッセージ属性の MD5 メッセージダイジェストがサポートされています。

クエリ API または Amazon SQS メッセージ属性の MD5 メッセージダイジェストをサポートしていないいずれかの AWS SDKs を使用する場合は、次のガイドラインを使用して MD5 メッセージダイジェスト計算を実行する必要があります。

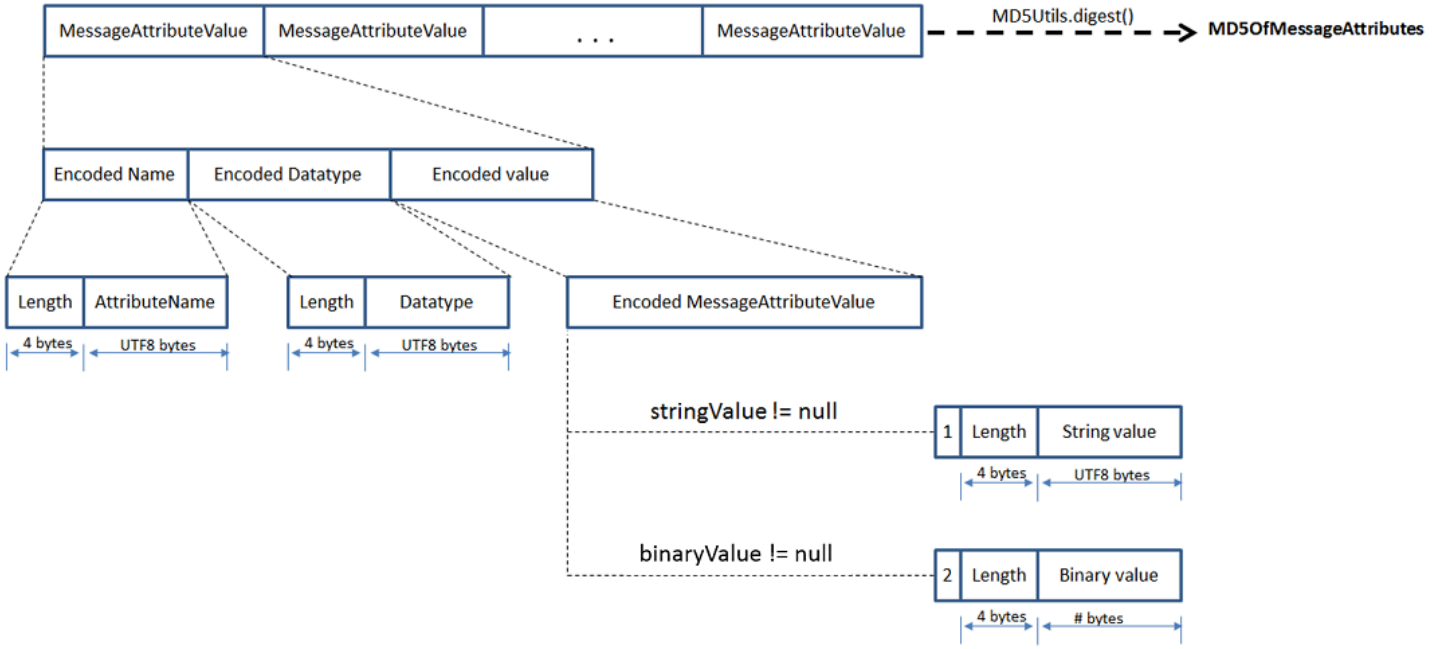
> **Note**
> MD5メッセージダイジェストの計算には、常にカスタムデータタイプサフィックスを含めてください。

概要

MD5メッセージダイジェスト計算アルゴリズムの概要を以下に示します。

1. すべてのメッセージ属性を名前で昇順にソートします。
2. 各属性 (`Name`、`Type`、`Value`) の個々の部分をバッファにエンコードします。
3. バッファ全体のメッセージダイジェストを計算します。

次の図は、1つのメッセージ属性のMD5メッセージダイジェストをエンコードする方法を示しています。



単一のAmazon SQSメッセージ属性をエンコードするには

1. 名前をエンコードします: 名前の長さ (4バイト) およびUTF-8バイト。
2. データ型をエンコードします:データ型の長さ (4バイト) および UTF-8バイト。
3. 値 (1バイト) の転送型 (`String`または`Binary`) をエンコードします。

> **Note**
>
> 論理データ型 `String`および`Number`では、`String`転送型が使用されます。
> 論理データ型 `Binary`では、`Binary`転送型が使用されます。

   a. `String`転送型の場合、1をエンコードします。
   b. `Binary`転送型の場合、2をエンコードします。
4. 属性値をエンコードします。
   a. `String`転送型の場合、属性値をエンコードします:値の長さ(4バイト) + UTF-8バイト。
   b. `Binary`転送型の場合、属性値をエンコードします:値の長さ(4バイト) + rawバイト。

Amazon SQSメッセージシステム属性

メッセージ属性を使用してアプリケーションの Amazon SQS メッセージにカスタムメタデータをアタッチできますが、メッセージシステム属性 AWS を使用して、 AWS X-Rayなどの他のサービスのメタデータを保存できます。詳細については、Amazon Simple キューサービス API リファレンス「SendMessageBatchAPI アクションMessageSystemAttributeのリクエストパラメーターSendMessage、API アクションAWSTraceHeaderの属性ReceiveMessage、およびMessageSystemAttributeValueのデータタイプ。」を参照してください。

メッセージシステム属性は、メッセージ属性とまったく同じ構造ですが、次の例外があります。

- 現在、サポートされているメッセージシステム属性はAWSTraceHeaderのみです。そのタイプはStringで、値は正しくフォーマットされた AWS X-Ray トレースヘッダー文字列である必要があります。
- メッセージシステム属性のサイズは、メッセージの合計サイズに対してはカウントされません。

Amazon SQSメッセージの処理に必要なリソース

Amazon SQS は、処理に必要なリソースの評価に役立つキュー内の遅延メッセージ、表示メッセージ、非表示メッセージの概算数の見積もりを提供します。可視性の詳細については、「Amazon SQS の可視性タイムアウト」を参照してください。

> **Note**
>
> 一部のメトリクスでは、Amazon SQS の分散アーキテクチャにより、結果は概算になります。ほとんどの場合、カウントはキュー内の実際のメッセージ数に近い数値になります。

以下の表に、GetQueueAttributesアクションで使用する属性名の一覧を示します:

タスク	属性名
キューから取得可能なメッセージのおおよその数を取得します。	ApproximateNumberOfMessagesVisible
キュー内の、遅延が発生したためにすぐに読み取ることができないメッセージのおおよその数	ApproximateNumberOfMessagesDelayed

タスク	属性名
を取得します。これは、キューが遅延キューとして設定されている場合、またはメッセージが遅延パラメータとともに送信された場合に発生することがあります。	
処理中のメッセージのおおよその数を取得します。メッセージがクライアントに送信されたが、まだ削除されていない場合、または表示期限に達していない場合、メッセージはインフライトとみなされます。	`ApproximateNumberOfMessagesNotVisible`

Amazon SQS リストキューのページネーション

`listQueues` および `listDeadLetterQueues` APIメソッドは、オプションのページネーションコントロールをサポートします。デフォルトでは、これらのAPIメソッドはレスポンスメッセージで最大1000通のキューを返信します。`MaxResults`パラメータを指定すると、各レスポンスで返信される結果が少なくなります。

`listQueues` または`listDeadLetterQueues` リクエストにパラメータ`MaxResults`を設定して、レスポンスで返信される結果の最大数を指定します。設定しない場合`MaxResults`の場合、レスポンスには最大1,000件の結果が含まれ、`NextToken`応答の値が NULL となります。

さらに表示する結果がある場合は、 `MaxResults`を設定するとレスポンスに`NextToken`の値が含まれます。結果の次のページを受け取るには、`listQueues` に対する次のリクエストで `NextToken` をパラメータとして使用します。レスポンスで`NextToken`の値がnullの場合、表示できる追加の結果はありません。

Amazon SQSコスト配分タグ

コスト配分のために Amazon SQS キューを整理および識別する場合は、キューの目的、所有者、または環境を識別するメタデータタグを追加できます。これは、キューが多数ある場合に特に便利です。Amazon SQSコンソールを使用してタグを設定するには、「the section called “キューにタグを設定する”」を参照してください。

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、タグキーと値を含めるための AWS アカウント 請求書を取得しま

す。詳細については、AWS Billing ユーザーガイドの月別コスト配分レポートの設定を参照してください。

各タグは、ユーザー定義のキー-バリュー（1つのキーと1つの値）で構成されます。たとえば、次のようなタグをキューに付けると、本番稼働用とテスト用のキューを簡単に識別できます。

キュー	キー	値
MyQueueA	QueueType	Production
MyQueueB	QueueType	Testing

> **Note**
>
> キュータグを使用する場合は、以下のガイドラインに注意してください。
>
> - 1つのキューに50 個以上のタグを追加することはお勧めしません。タグは、UTF-8 の Unicode 文字をサポートします。
> - タグには意味論的な意味がありません。Amazon SQSはタグを文字列として解釈します。
> - タグは、大文字と小文字が区別します。
> - 既存のタグと同じキーを持つ新しいタグは、既存のタグを上書きします。
> - タグ付けアクションは、 あたり 30 TPS に制限されています AWS アカウント。アプリケーションが高スループットが必要な場合、リクエストを送信します。。
>
> タグの制限事項一覧については、「Amazon SQS 標準キューのクォータ」を参照してください。

Amazon SQSショートポーリングとロングポーリング

Amazon SQS には、キューからメッセージを受信するためのオプションとして、ショートポーリングとロングポーリングがあります。この 2 つのポーリングオプションのいずれかを選択する場合は、応答性とコスト効率に関するアプリケーションの要件を考慮してください。

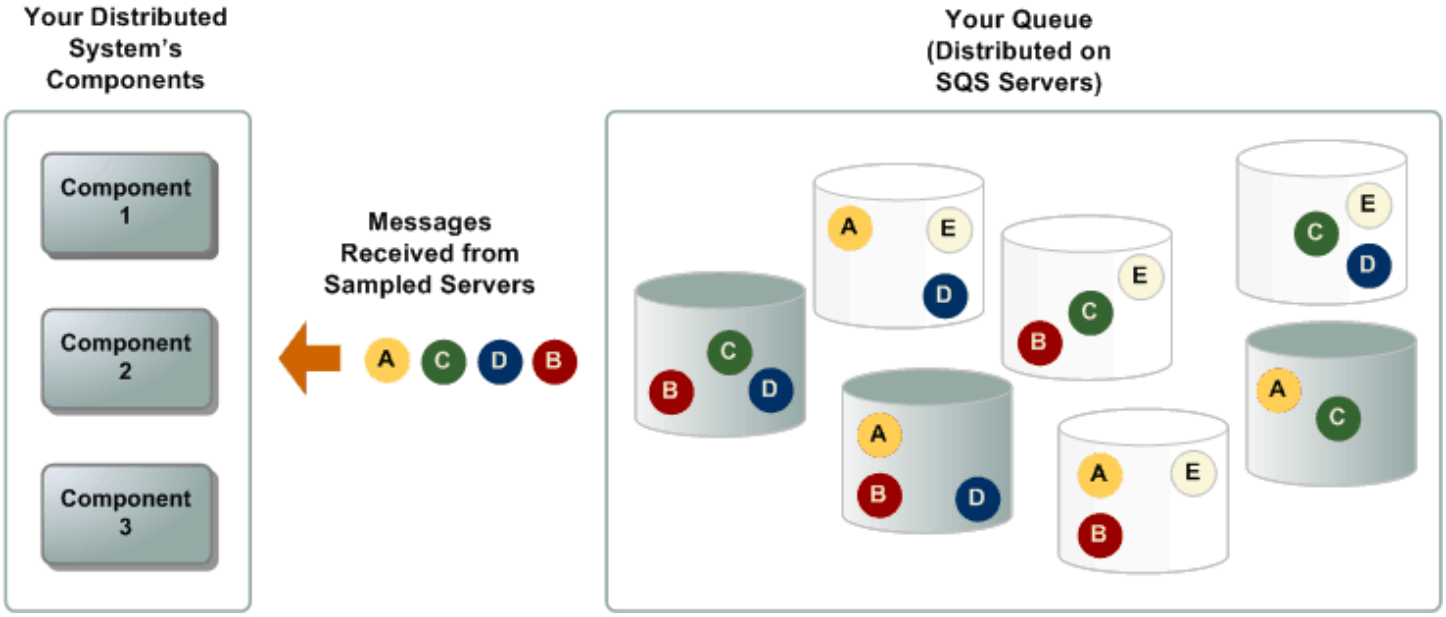
- ショートポーリング (デフォルト) – `ReceiveMessage` リクエストは、サーバーのサブセット (加重ランダム分布に基づく) にクエリを実行して使用可能なメッセージを検索し、メッセージが見つからない場合でもレスポンスを即時送信します。
- ロングポーリング – `ReceiveMessage` は、すべてのサーバーに対してメッセージの有無をクエリし、1 つ以上のメッセージが (指定された最大数まで) 使用可能になると、レスポンスを送信します。ポーリングの待機時間が期限切れになった場合にのみ、空のレスポンスが送信されます。このオプションを使用すると、空のレスポンスの数が減少し、コストが削減される可能性があります。

次のセクションでは、ショートポーリングとロングポーリングの詳細について説明します。

ショートポーリングを使用したメッセージの処理

ショートポーリングを使用して (FIFO または標準) キューのメッセージを消費すると、Amazon SQS はサーバーのサブセットを (加重ランダム分布に基づいて) サンプリングし、これらのサーバーのメッセージだけを返します。したがって、特定の `ReceiveMessage` リクエストで、すべてのメッセージは返されない場合があります。ただし、キューにあるメッセージが1,000未満の場合、後続のリクエストではメッセージが返されます。キューから処理し続けた場合、Amazon SQSによりすべてのサーバーがサンプリングされ、すべてのメッセージを受信します。

次の図は、システムコンポーネントの 1 つが受信リクエストを行った後で、標準キューからメッセージを返すショートポーリングの動作を示しています。Amazon SQSは、複数のサーバー (灰色) をサンプリングし、それらのサーバーからメッセージ A、C、D、および B を返信します。メッセージ E はこのリクエストに返されませんが、後続のリクエストには返されます。



ロングポーリングを使用したメッセージの使用

待ち時間がいつになるかReceiveMessageAPI アクションが0より大きい場合ロングポーリングが有効です。ロングポーリングの最大待機時間は 20 秒です。ロングポーリングは、空のレスポンス (ReceiveMessage リクエストに対して利用可能なメッセージがない場合) や偽の空のレスポンス (メッセージが利用可能でもレスポンスに含まれない場合) の数を減らすことにより、Amazon SQS の使用コストの削減に役立ちます。Amazon SQSコンソールを使用して、新しいキューまたは既存のキューのロングポーリングを有効にする方法については、Amazon SQS コンソールを使用したキューのパラメータの設定を参照してください。ベストプラクティスについては、Amazon SQS でのロングポーリングの設定を参照してください。

ロングポーリングには次の利点があります。

- レスポンスの送信前にメッセージがキューで使用可能になるまで Amazon SQSが待機できるように、空のレスポンス数を削減します。接続がタイムアウトしない限り、ReceiveMessageリクエストに対するレスポンスに、使用可能なメッセージが少なくとも1つ、最大でReceiveMessageアクションに指定されたメッセージ数まで含まれます。まれに、キューにまだメッセージが含まれている場合でも、空の応答が受信されることがあります、特に、ReceiveMessageWaitTimeSecondsパラメータに低い値を指定した場合。
- サブセットではなく、すべての Amazon SQSサーバーにクエリを実行して、偽の空のレスポンスを減らします。
- 利用可能になるとすぐにメッセージを返します。

キューが空であることを確認する方法については、Amazon SQS キューが空であることを確認するを参照してください。

ロングポーリングとショートポーリングの違い

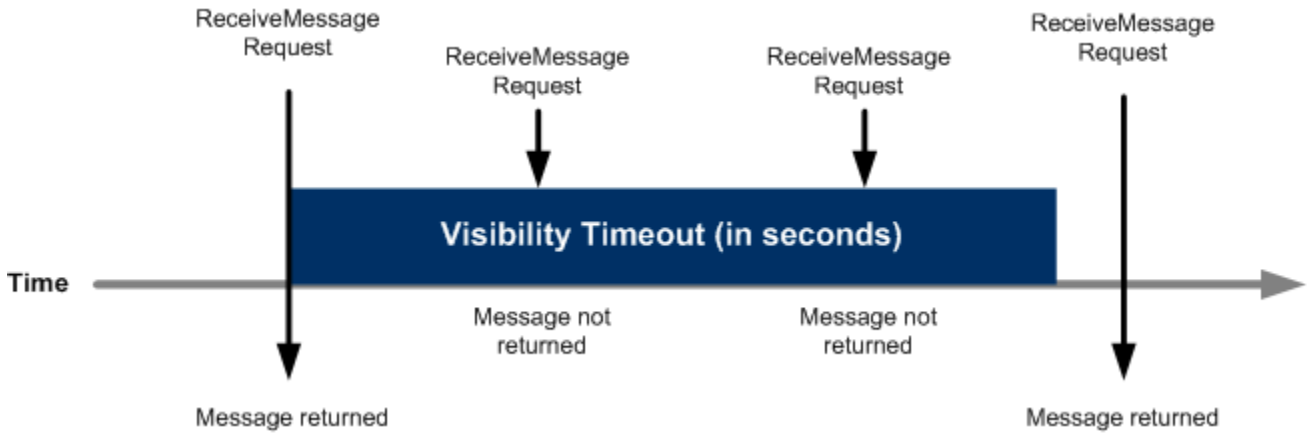
ショートポーリングは、WaitTimeSecondsリクエストのReceiveMessageパラメータを次の2通りの方法で0に設定すると行われます。

- ReceiveMessageコールは WaitTimeSeconds を 0 に設定します。
- ReceiveMessage コールでは WaitTimeSeconds を設定しませんが、キューの属性 ReceiveMessageWaitTimeSeconds は 0 に設定されます。

Amazon SQS の可視性タイムアウト

Amazon SQS キューからメッセージを受信すると、キューに残りますが、他のコンシューマーには一時的に見えなくなります。この可視性は可視性タイムアウトによって制御されます。これにより、作業中に他のコンシューマーが同じメッセージを処理できなくなります。Amazon SQS には、処理後にメッセージを削除するための 2 つのオプションがあります。

- 手動削除 – `DeleteMessage`アクションを使用してメッセージを明示的に削除します。
- 自動削除 – 特定の AWS SDKsでサポートされ、処理が成功するとメッセージが自動的に削除され、ワークフローが簡素化されます。



可視性タイムアウトのユースケース

長時間実行されるタスクの管理 – 可視性タイムアウトを使用して、処理時間が長いタスクを処理します。処理時間が長くなるメッセージには、適切な可視性タイムアウトを設定します。これにより、他のコンシューマーが処理中に同じメッセージを受信しないようにし、重複する作業を防ぎ、システム効率を維持します。

再試行メカニズムの実装 – 初期タイムアウト内に完了しなかったタスクの可視性タイムアウトをプログラムで延長します。タスクが最初の可視性タイムアウト内に完了しなかった場合は、タイムアウトをプログラムで延長できます。これにより、システムは他のコンシューマーに表示されずにメッセージの処理を再試行できるため、耐障害性と信頼性が向上します。をデッドレターキュー (DLQs) と組み合わせて、永続的な障害を管理します。

分散システムの調整 – SQS 可視性タイムアウトを使用して、分散システム全体のタスクを調整します。さまざまなコンポーネントの予想される処理時間と一致する可視性タイムアウトを設定します。これにより、一貫性を維持し、複雑な分散アーキテクチャでの競合状態を防ぐことができます。

リソース使用率の最適化 – SQS 可視性タイムアウトを調整して、アプリケーションのリソース使用率を最適化します。適切なタイムアウトを設定することで、リソースを不必要に結び付けることなく、メッセージが効率的に処理されるようにできます。これにより、システム全体のパフォーマンスとコスト効率が向上します。

可視性タイムアウトの設定と調整

可視性タイムアウトは、メッセージが配信されるとすぐに開始されます。この期間中は、メッセージを処理および削除する必要があります。タイムアウトの有効期限が切れる前に削除しない場合、メッセージはキューに再び表示され、別のコンシューマーが取得できます。キューのデフォルトの可視性タイムアウトは 30 秒ですが、アプリケーションがメッセージを処理および削除するために必要な時間に合わせて調整できます。キューの全体的な設定を変更せずに、個々のメッセージの特定の可視性タイムアウトを設定することもできます。`ChangeMessageVisibility` アクションを使用して、必要に応じてタイムアウトをプログラムで延長または短縮します。

インフライトメッセージとクォータ

Amazon SQSでは、インフライトメッセージは、コンシューマーが受信したが、まだ削除されていないメッセージです。標準キューの場合、キュートラフィックとメッセージバックログに応じて、約 120,000 個の処理中のメッセージに制限があります。この制限に達すると、Amazon SQS は`OverLimit`エラーを返し、一部の処理中のメッセージが削除されるまで追加のメッセージを受信できないことを示します。FIFO キューの場合、制限はアクティブなメッセージグループによって異なります。

- ショートポーリングを使用する場合 – ショートポーリングの使用中にこの制限に達すると、Amazon SQS は`OverLimit`エラーを返し、一部の処理中のメッセージが削除されるまで追加のメッセージを受信できないことを示します。
- ロングポーリングを使用する場合 – ロングポーリングを使用している場合、Amazon SQS は処理中のメッセージ制限に達したときにエラーを返しません。代わりに、インフライトメッセージ数が制限を下回るまで、新しいメッセージを返しません。

転送中のメッセージを効果的に管理するには：

1. プロンプト削除 – 処理後にメッセージ (手動または自動) を削除し、処理中の数を減らします。
2. CloudWatch でモニタリング – 制限に達しないように、処理中の数が多いアラームを設定します。
3. 負荷を分散する – 大量のメッセージを処理している場合は、追加のキューまたはコンシューマーを使用して負荷を分散し、ボトルネックを回避します。

4. クォータの引き上げをリクエストする – より高い制限が必要な場合は、 AWS サポートにリクエストを送信します。

標準キューと FIFO キューの可視性タイムアウトについて

標準キューと FIFO (First-In-First-Outキューの両方で、可視性タイムアウトは、複数のコンシューマーが同じメッセージを同時に処理するのを防ぐのに役立ちます。ただし、Amazon SQS at-least-onceであるため、可視性タイムアウト期間中にメッセージが複数回配信されないという絶対的な保証はありません。

- 標準キュー – 標準キューの可視性タイムアウトにより、複数のコンシューマーが同じメッセージを同時に処理できなくなります。ただし、at-least-once配信モデルのため、Amazon SQS は可視性タイムアウト期間内にメッセージが複数回配信されないことを保証しません。
- FIFO キュー – FIFO キューの場合、同じメッセージグループ ID を持つメッセージは厳密な順序で処理されます。メッセージグループ ID を持つメッセージが処理中の場合、そのグループ内の後続のメッセージは、処理中のメッセージが削除されるか、可視性タイムアウトの有効期限が切れるまで使用できなくなります。ただし、これはグループを無期限に「ロック」しません。各メッセージは順番に処理され、各メッセージが削除されるか再び表示される場合にのみ、そのグループ内の次のメッセージがコンシューマーに利用可能になります。このアプローチにより、グループがメッセージを配信するのを不必要にロックすることなく、グループ内の順序付けられた処理が保証されます。

失敗の処理

アプリケーションエラー、クラッシュ、または接続の問題により、可視性タイムアウトの期限が切れる前にメッセージを処理および削除しない場合、メッセージはキューに再び表示されます。その後、同じコンシューマーまたは別のコンシューマーが別の処理を試みるために取得できます。これにより、初期処理が失敗してもメッセージが失われることがなくなります。ただし、可視性タイムアウトの設定が高すぎると、未処理のメッセージの再表示が遅れ、再試行が遅くなる可能性があります。タイムリーなメッセージ処理には、予想される処理時間に基づいて適切な可視性タイムアウトを設定することが重要です。

可視性タイムアウトの変更と終了

可視性タイムアウトは、 `ChangeMessageVisibility`アクションを使用して変更または終了できます。

- タイムアウトの変更 – を使用して可視性タイムアウトを動的に調整しますChangeMessageVisibility。これにより、処理ニーズに合わせてタイムアウト期間を延長または短縮できます。
- タイムアウトの終了 – 受信したメッセージを処理しない場合は、 ChangeMessageVisibility アクションを通じて を 0 VisibilityTimeout 秒に設定して可視性タイムアウトを終了します。これに伴って、他のコンシューマーがメッセージをすぐに処理できるようになります。

ベストプラクティス

タイムアウトの設定、調整、延長、デッドレターキュー (DLQ) を使用した未処理メッセージへの対応など、Amazon SQS の可視性タイムアウトを管理するには、以下のベストプラクティスを使用します。

- タイムアウトの設定と調整。まず、アプリケーションがメッセージを処理して削除するのに通常必要な最大時間に合わせて可視性タイムアウトを設定します。正確な処理時間が不明な場合は、短いタイムアウト (2 分など) で開始し、必要に応じて延長します。ハートビートメカニズムを実装して可視性タイムアウトを定期的に延長し、処理が完了するまでメッセージが見えないようにします。これにより、未処理メッセージの再処理の遅れを最小限に抑えるとともに、再表示が早すぎないようにします。
- タイムアウトの延長と 12 時間の制限の処理。処理時間が変動するか、最初に設定されたタイムアウトを超える可能性がある場合は、 ChangeMessageVisibilityアクションを使用して、メッセージの処理中に可視性タイムアウトを延長します。可視性タイムアウトの上限は、メッセージを最初に受信してから 12 時間であることに注意してください。タイムアウトを延長しても、この 12 時間の制限はリセットされません。処理にこの制限よりも時間がかかる場合は、 を使用する AWS Step Functions か、タスクを小さなステップに分割することを検討してください。
- 未処理のメッセージの処理。複数の処理試行に失敗したメッセージを管理するには、デッドレターキュー (DLQ) を設定します。これにより、複数回の再試行後に処理できないメッセージは、さらなる分析や処理のために個別にキャプチャされ、メインキューで繰り返し循環することが防止されます。

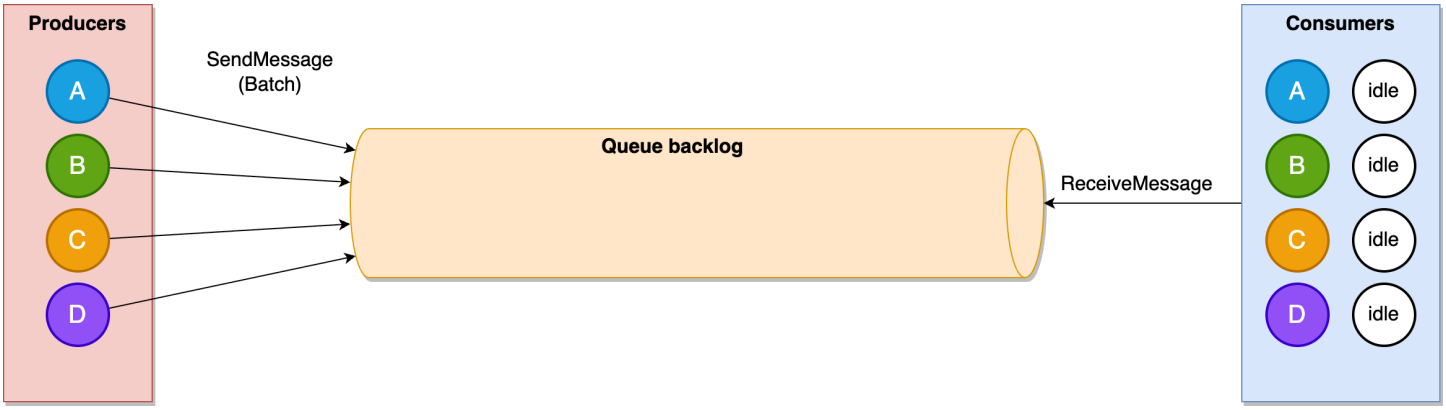
Amazon SQS 公平キュー

Amazon SQS 公平キューは、顧客、クライアントアプリケーション、メッセージタイプなど、複数の論理エンティティからのメッセージを含むマルチテナントキューのノイズの多い近隣の影響を自動的に軽減します。これらの共有キュー環境では、重要なパフォーマンスメトリクスの 1 つはドウェル時間です。ドウェル時間は、メッセージがキューに到着してから処理までに費やされた合計時間

を測定します。あるテナントが、システムが処理できるよりも多くのメッセージを公開してキューにバックログを作成すると、公平キューは他のテナントのドウェル時間への影響を最小限に抑えます。

定常状態

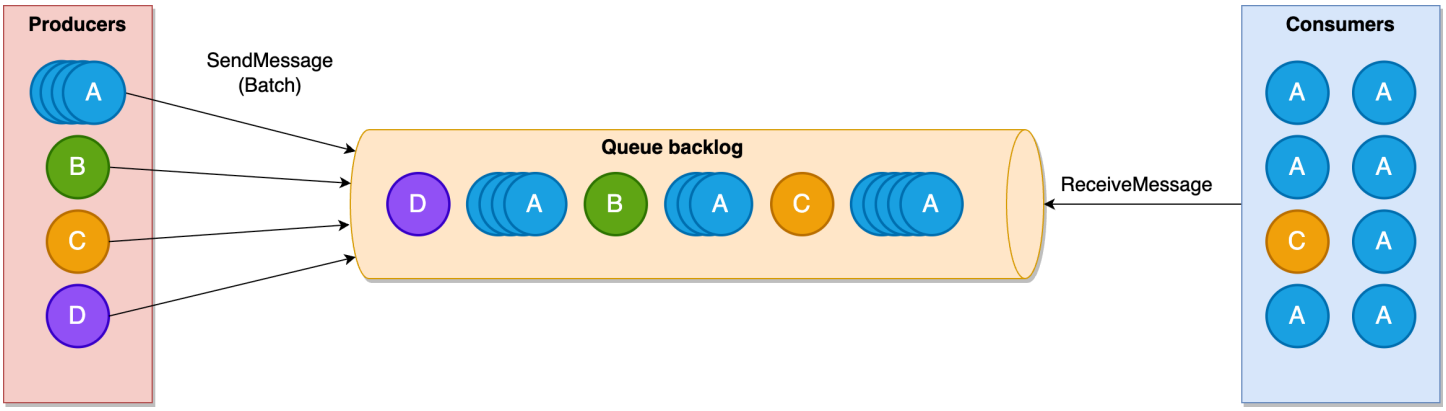
次の図は、4 つの異なるテナント (ラベル付き A、B、C、D) からのメッセージを含むマルチテナントキューを示しています。キューは定常状態で動作し、コンシューマーがキューに表示されるとすぐにメッセージを受信するため、メッセージバックログはありません。すべてのテナントのドウェル時間は短くなります。すべてのコンシューマー容量がこの定常状態で完全に使用されているわけではありません。



ノイズの多い近隣への影響

ノイズの多い近隣影響は、マルチテナントキュー内の 1 つのテナントがバックログを作成し、他のすべてのテナントのメッセージドウェル時間を増やすときに発生します。テナントは、他のテナントよりも大量のメッセージを送信したり、コンシューマーがその特定のテナントからのメッセージを処理するのに時間がかかると、ノイズの多い隣人になる可能性があります。

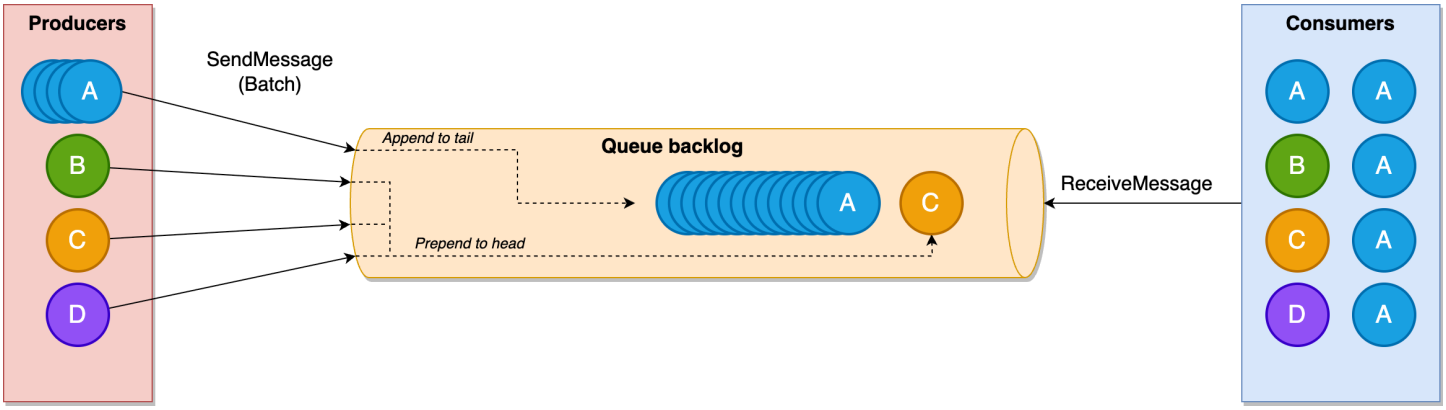
この図は、テナント A からのトラフィックの増加がキューにバックログを作成する方法を示しています。コンシューマーはテナント A からのメッセージのみの処理にビジー状態で、他のテナントからのメッセージはバックログで待機するため、すべてのテナントのドウェル時間が長くなります。



公平キューによる緩和

Amazon SQSは、処理中にテナント間のメッセージ分散 (「処理中」状態) をモニタリングすることで、ノイズの多い近隣を検出します。テナントが他のテナントと比較して不釣り合いに多数のインフライトメッセージを持っている場合、Amazon SQS はそのテナントをノイズの多い隣人として識別し、他のテナントのメッセージ配信を優先します。このアプローチにより、他のテナントへのドウェル時間の影響が軽減されます。

この図は、Amazon SQS 公平キューがノイズの多い近隣問題にどのように対処するかを示しています。1 つのテナント (テナント A) がノイズが多い場合、Amazon SQS は他のテナント (B、C、D) からのメッセージを返すことを優先します。この優先順位付けは、クワイエットテナントのテナント B、C、D の低いドウェル時間を維持するのに役立ちます。一方、テナント A のメッセージのドウェル時間は、キューバックログが他のテナントに影響を与えずに消費されるまで長くなります。



> **Note**
>
> Amazon SQS はテナントあたりの消費レートを制限しません。これにより、コンシューマー容量があり、キューに返す他のメッセージがない場合に、ノイズの多い近隣テナントからメッセージを受信できます。Amazon SQS 標準キューと同様に、公平キューは事実上無制限のスループットを可能にし、キューに含めることができるテナントの数に制限はありません。

FIFO キューとの違い

FIFO キューは、各テナントからの転送中のメッセージの数を制限することで、厳密な順序を維持します。これによりノイズの多いネイバーを防ぐことができますが、各テナントのスループットが制限されます。公平キューは、高スループット、低ドウェル時間、公平なリソース割り当てが優先されるマルチテナントシナリオ向けに設計されています。公平キューを使用すると、複数のコンシューマー

が同じテナントからのメッセージを同時に処理しながら、すべてのテナントが一貫した滞留時間を維持できます。

公平キューの使用

メッセージプロデューサーは、送信メッセージMessageGroupIdに を設定することでテナント識別子を追加できます。

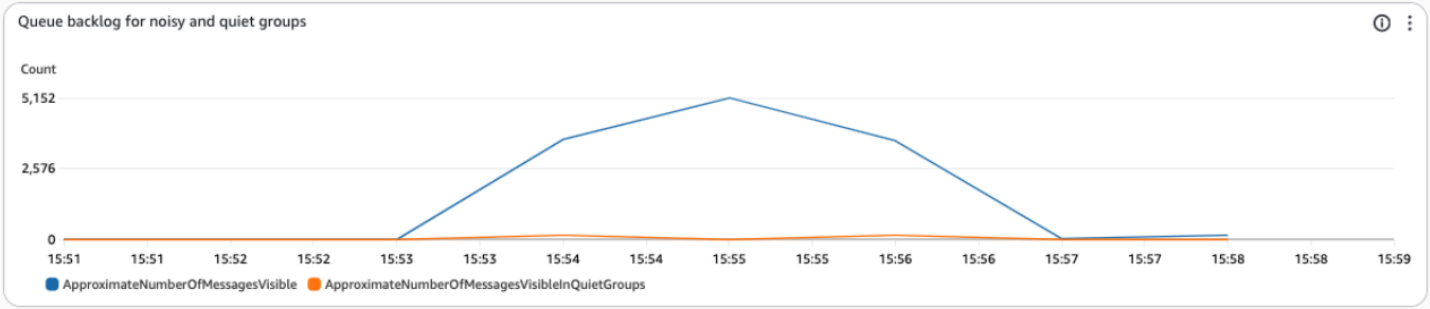
```
// Send message with tenant identifier
SendMessageRequest request = new SendMessageRequest()
    .withQueueUrl(queueUrl)
    .withMessageBody(messageBody)
    .withMessageGroupId("tenant-123"); // Tenant identifier
sqs.sendMessage(request);
```

公平性機能は、MessageGroupId プロパティを持つメッセージのすべての Amazon SQS 標準キューに自動的に適用されます。コンシューマーコードを変更する必要はありません。API レイテンシーには影響せず、スループットの制限もありません。

公平キュー CloudWatch メトリクス

Amazon SQS には、ノイズの多い近隣への影響の軽減をモニタリングするのに役立つ追加の CloudWatch メトリクスが用意されています。例えば、Approximate..InQuietGroupsメトリクスを標準のキューレベルのメトリクスと比較できます。特定のテナントのトラフィックが急増すると、一般的なキューレベルのメトリクスで、バックログの増加やメッセージ経過時間が古くなる可能性があります。ただし、クワイエットグループを個別に見ると、ノイズのないメッセージグループやテナントのほとんどが影響を受けないことを特定できます。

以下に、ノイズの多いテナントが原因で標準キューのバックログメトリクス (ApproximateNumberOfMessagesVisible) が増加し、ノイズのないテナントのバックログ (ApproximateNumberOfMessagesVisibleInQuietGroups) は低いままである例を示します。



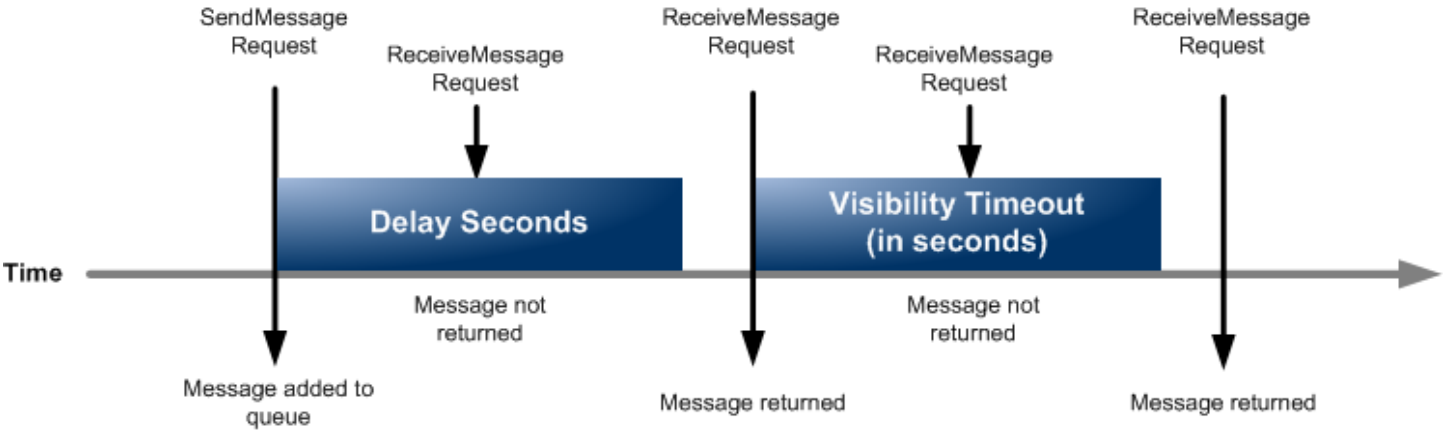
Amazon SQS CloudWatch メトリクスとその説明の完全なリストについては、「Amazon SQS の CloudWatch メトリクス Amazon SQS」を参照してください。

Amazon SQS 遅延キュー

遅延キューは、例えば、コンシューマアプリケーションがメッセージ処理にさらに時間を必要な場合、コンシューマへ新しいメッセージの配信を数秒延期することができます。遅延キューを作成した場合、そのキューに送信したすべてのメッセージは遅延期間中にコンシューマーに表示されません。キューのデフォルト（最小）遅延は0秒です。最大数は15分です。コンソールを使用して遅延キューを設定する方法の詳細については、「Amazon SQS コンソールを使用したキューのパラメータの設定」を参照してください。

> **Note**
>
> 標準キューの場合は、キューごとの遅延設定には遡及性はありません。設定を変更しても、既にキューにあるメッセージの遅延には影響しません。
> FIFOキューの場合は、キューごとの遅延設定に遡及性があります-設定を変更すると、既にキューにあるメッセージの遅延に影響します。

遅延キューでは、コンシューマーはメッセージを一定の時間使用できなくなるため、可視性タイムアウトと似ています。両者の違いは、遅延キューの場合はメッセージが最初にキューに追加されたときに非表示になるのに対し、可視性タイムアウトの場合は、メッセージがキューから処理された後のみ非表示になるという点です。次の図は、遅延キューと可視性タイムアウトの関係を示しています。



拡張スケジューリングオプション

Amazon SQS 遅延キューとメッセージタイマーでは、最大 15 分後にメッセージ配信をスケジュールできますが、より柔軟なスケジューリング機能が必要になる場合があります。このような場合

は、EventBridge スケジューラの使用を検討してください。EventBridge スケジューラを使用すると、時間制限なしで数十億回または定期的な API アクションをスケジュールできます。EventBridge スケジューラは、高度なメッセージスケジューリングのユースケースに推奨されるソリューションです。

キュー全体ではなく個々のメッセージに遅延秒を設定するには、メッセージタイマーを使用して、Amazon SQS が遅延キュー`DelaySeconds`の値の代わりにメッセージタイマー`DelaySeconds`の値を使用できるようにします。EventBridge スケジューラは、個々のメッセージのスケジューリングもサポートしています。

Amazon SQS一時キュー

一時キューは、リクエストとレスポンスのような一般的なメッセージパターンを使用する場合に、開発時間と開発コストを削減するのに役立ちます。一時キュークライアントを使用して、高スループットでコスト効率の高いアプリケーション管理の一時キューを作成します。

クライアントは複数のマッピングを行います。一時キュー— 特定のプロセスに対してオンデマンドで作成されるアプリケーション管理キューを、単一の Amazon SQSキューに自動的に作成します。これにより、各一時キューへのトラフィックが少ないときのアプリケーションのAPI呼び出しが少なくなり、スループットを高めることができます。一時キューが使用されなくなると、クライアントを使用する一部のプロセスが正常にシャットダウンされない場合でも、クライアントは一時キューを自動的にクリーンアップします。

一時キューの利点を以下に示します:

- これらは、特定のスレッドまたはプロセスの軽量通信チャネルとして機能します。
- 追加のコストが発生することなく作成および削除できます。
- これらは静的 (通常)Amazon SQSキューとAPI互換です。つまり、メッセージを送受信する既存のコードは、仮想キューとの間でメッセージを送受信できます。

仮想キュー

仮想キューは、Temporary Queue Clientが作成するローカルデータ構造です。仮想キューを使用すると、トラフィックの少ない複数の宛先を単一のAmazon SQSキューに結合できます。ベストプラクティスについては、「仮想キューで同じメッセージグループ ID を再利用しないようにする」を参照してください。

> **Note**
>
> - 仮想キューを作成すると、コンシューマーがメッセージを受信するための一時的なデータ構造のみが作成されます。仮想キューはAmazon SQSへのAPI呼び出しを行わないため、仮想キューにはコストはかかりません。
> - TPSクォータは、単一ホストキューのすべての仮想キューに適用されます。詳細については、「Amazon SQS のメッセージキュー」を参照してください。

`AmazonSQSVirtualQueuesClient`ラッパークラスは、仮想キューに関連する属性のサポートを追加します。仮想キューを作成するには、 `CreateQueue` 属性を使用して`HostQueueURL`API アクションを呼び出す必要があります。この属性は、仮想キューをホストする既存のキューを指定します。

仮想キューのURLは次の形式になります。

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue#MyVirtualQueueName
```

プロデューサーが仮想キュー URLで`SendMessage`または`SendMessageBatch` API アクションを呼び出すと、Temporary Queue Clientでは次の処理が実行されます:

1. 仮想キュー名を抽出します。
2. 追加のメッセージ属性として仮想キュー名にアタッチします。
3. ホストのキューにメッセージを送信します。

プロデューサーがメッセージを送信する間、バックグラウンドスレッドはホストキューをポーリングし、対応するメッセージ属性に従って受信メッセージを仮想キューに送信します。

コンシューマが仮想キュー URL で`ReceiveMessage`API アクションを呼び出す間、Temporary Queue Clientは、バックグラウンドスレッドが仮想キューにメッセージを送信するまで、ローカルで呼び出しをブロックします。(このプロセスは、バッファ付き非同期クライアントでのメッセージのプリフェッチに似ています。1 つの APIアクションで最大 10 個の仮想キューにメッセージを提供できます。) 仮想キューを削除すると、Amazon SQS自体を呼び出すことなく、クライアント側のリソースが削除されます。

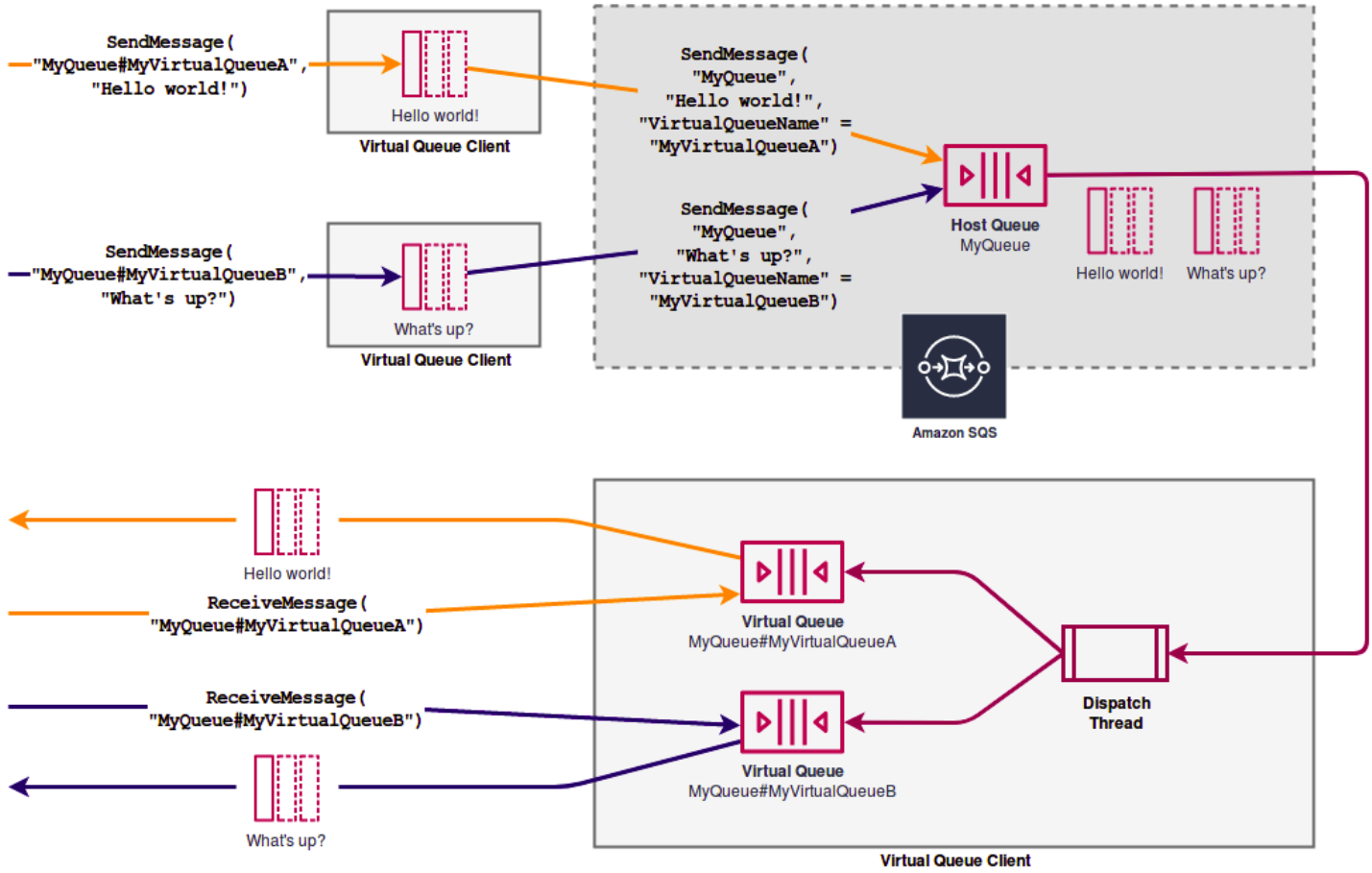
`AmazonSQSTemporaryQueuesClient`クラスは、作成したすべてのキューを一時キューに自動的に変換します。また、オンデマンドで、同じキュー属性を持つホストキューを自動的に作成しま

す。これらのキューの名前は、一時キューとして識別される共通の構成可能なプレフィックス(デフォルトでは__RequesterClientQueues__) を共有します。これにより、クライアントは、キューを作成および削除する既存のコードを最適化するドロップイン置換として機能できるようになります。クライアントには、キュー間の双方向通信を可能にする AmazonSQSRequester および AmazonSQSResponder インターフェイスも含まれています。

リクエスト–レスポンスメッセージングパターン(仮想キュー)

一時キューの最も一般的な使用例は、リクエスト-レスポンスメッセージングパターンです。このパターンでは、要求者が各レスポンスメッセージを受信するための一時キューを作成します。レスポンスメッセージごとにAmazon SQSキューが作成されないように、Temporary Queue Clientを使用すると、Amazon SQS API呼び出しを行わずに複数の一時キューを作成および削除できます。詳細については、「リクエスト-レスポンスシステムの実装」を参照してください。

このパターンを使用した一般的な構成を次の図に示します。



シナリオ例:ログインリクエストの処理

次のシナリオ例では、AmazonSQSRequesterとAmazonSQSResponderインターフェイスを使用して、ユーザーのログインリクエストを処理する方法を示しています。

クライアント側

```
public class LoginClient {

    // Specify the Amazon SQS queue to which to send requests.
    private final String requestQueueUrl;

    // Use the AmazonSQSRequester interface to create
    // a temporary queue for each response.
    private final AmazonSQSRequester sqsRequester =
            AmazonSQSRequesterClientBuilder.defaultClient();

    LoginClient(String requestQueueUrl) {
        this.requestQueueUrl = requestQueueUrl;
    }

    // Send a login request.
    public String login(String body) throws TimeoutException {
        SendMessageRequest request = new SendMessageRequest()
                .withMessageBody(body)
                .withQueueUrl(requestQueueUrl);

        // If no response is received, in 20 seconds,
        // trigger the TimeoutException.
        Message reply = sqsRequester.sendMessageAndGetResponse(request,
                20, TimeUnit.SECONDS);

        return reply.getBody();
    }
}
```

ログインリクエストを送信すると、次の処理が実行されます。

1. 一時テーブルを作成します。
2. 一時キューの URLを属性としてメッセージに添付します。
3. メッセージを送信します。

4. 一時キューからのレスポンスを受け取ります。
5. 一時キューを削除します。
6. レスポンスを返します。

サーバー側

次の例では、構築時に、キューをポーリングしてすべてのメッセージに対してhandleLoginRequest()メソッドを呼び出すスレッドが作成されることを想定しています。さらに、doLogin() メソッドを想定しています。

```
public class LoginServer {

    // Specify the Amazon SQS queue to poll for login requests.
    private final String requestQueueUrl;

    // Use the AmazonSQSResponder interface to take care
    // of sending responses to the correct response destination.
    private final AmazonSQSResponder sqsResponder =
            AmazonSQSResponderClientBuilder.defaultClient();

    LoginServer(String requestQueueUrl) {
        this.requestQueueUrl = requestQueueUrl;
    }

    // Process login requests from the client.
    public void handleLoginRequest(Message message) {

        // Process the login and return a serialized result.
        String response = doLogin(message.getBody());

        // Extract the URL of the temporary queue from the message attribute
        // and send the response to the temporary queue.
        sqsResponder.sendResponseMessage(MessageContent.fromMessage(message),
                new MessageContent(response));
    }
}
```

キューをクリーンアップする

Amazon SQS. が仮想キューで使用されているメモリ内リソースを確実に回収するために、アプリケーションで Temporary Queue Client が不要になったら、shutdown()メソッドを呼び出す必要が

あります。shutdown()インターフェイスのAmazonSQSRequesterメソッドを使用することもできます。

Temporary Queue Client孤立したホストキューを削除する方法も提供します。一定期間 (デフォルトでは5分間) にわたって API 呼び出しを受信する各キューについて、クライアントはTagQueueAPIアクションを使用して、使用中のキューにタグを付けます。

> **Note**
>
> キューで実行されたAPIアクションは、メッセージを返さないReceiveMessage アクションを含め、キューをアイドル以外としてマークします。

バックグラウンドスレッドは ListQueues および ListTags APIアクションを使用して、構成されたプレフィックスを持つすべてのキューをチェックし、少なくとも5分間タグ付けされていないキューを削除します。このようにして、1つのクライアントが正常にシャットダウンしない場合、他のアクティブなクライアントはその後クリーンアップします。作業の重複を減らすために、同じプレフィックスを持つすべてのクライアントは、プレフィックスにちなんで命名された共有内部作業キューを介して通信します。

Amazon SQSメッセージタイマー

メッセージタイマーを使用すると、メッセージがキューに追加されるときに、メッセージの初期非表示期間を設定できます。たとえば、45 秒のタイマーでメッセージを送信した場合、最初の 45 秒間はコンシューマーから非表示のままになります。メッセージのデフォルト (最小) 遅延は0秒です。最大数は15分です。コンソール を使用してタイマーでメッセージを送信する方法については、「標準キューを使用したメッセージの送信」を参照してください。

> **Note**
>
> FIFOキューは、個々のメッセージのタイマーをサポートしていません。

個々のメッセージではなくキュー全体に遅延期間を設定するには、遅延キューを使用します。個々のメッセージのメッセージタイマー設定は、Amazon SQS 遅延キューのすべての DelaySeconds 値よりも優先されます。

拡張スケジューリングオプション

Amazon SQS 遅延キューとメッセージタイマーでは、最大 15 分後にメッセージ配信をスケジュールできますが、より柔軟なスケジューリング機能が必要になる場合があります。このような場合は、EventBridge スケジューラの使用を検討してください。EventBridge スケジューラを使用すると、時間制限なしで数十億回または定期的な API アクションをスケジュールできます。EventBridge スケジューラは、高度なメッセージスケジューリングのユースケースに推奨されるソリューションです。

Amazon SQS コンソールから Amazon EventBridge Pipes へのアクセス

Amazon EventBridge Pipes はソースをターゲットに接続します。パイプは、サポートされているソースとターゲット間のポイントツーポイント統合を目的としており、高度な変換とエンリッチメントをサポートしています。EventBridge Pipes は、Amazon SQS キューを Step Functions、Amazon SQS、API Gateway などの AWS サービスや、Salesforce などのサードパーティーの Software as a Service (SaaS) アプリケーションに接続するための非常にスケーラブルな方法を提供します。

パイプをセットアップするには、ソースを選択し、オプションのフィルタリングを追加し、オプションのエンリッチメントを定義し、イベントデータのターゲットを選択します。

Amazon SQS キューの詳細ページでは、そのキューをソースとして使用しているパイプを表示できます。そこから、次の事柄も実行できます。

- EventBridge コンソールを起動し、パイプの詳細を表示します。
- EventBridge コンソールを起動し、キューをソースとする新しいパイプを作成します。

Amazon SQS キューのパイプソースとしての設定に関する詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon SQS キューをソースとする場合」を参照してください。EventBridge Pipes 全般の詳細については、「Amazon EventBridge Pipes」を参照してください。

特定の Amazon SQS キューの EventBridge パイプにアクセスするには

1. Amazon SQSコンソールの キュー ページを開きます。
2. キューを選択します。
3. キューの詳細ページで、[EventBridge Pipes] タブを選択します。

   [EventBridge Pipes] タブには、選択されたキューをソースとして使用するように現在設定されているすべてのパイプのリストが含まれており、これには以下が記載されています。

- パイプ名
- 現在の状態
- パイプのターゲット
- パイプが最後に変更された時間

4. 必要に応じて、パイプの詳細を表示するか、新しいパイプを作成します。

   - パイプの詳細にアクセスするには:

     パイプ名を選択します。

     これにより、EventBridge コンソールの [パイプの詳細] ページが起動します。

   - 新しいパイプを作成するには:

     [Amazon SQS キューをパイプに接続] を選択します。

     これにより、EventBridge コンソールの [パイプの作成] ページが起動し、Amazon SQS キューがパイプのソースとして指定された状態になります。詳細については、「Amazon EventBridge ユーザーガイド」の「Amazon EventBridge パイプの作成」を参照してください。

     > **Important**
     >
     > Amazon SQS キュー上のメッセージは、単一のパイプによって読み取られてから、処理された後でキューから削除されます。これは、そのパイプに設定できるフィルターにメッセージが一致するかどうかを問わず、実行されます。複数のパイプが同一のキューをソースとして使用するように設定する場合は、十分に注意を払ってください。

拡張クライアントライブラリと Amazon Simple Storage Service を使用した大量の Amazon SQS メッセージの管理

Amazon Amazon SQS Extended Client Library for Java と Amazon SQS Extended Client Library for Python を使用して、特に 256 KB ~ 2 GB のペイロードで大きなメッセージを送信します。これらのライブラリは、メッセージペイロードを Amazon S3 バケットに保存し、Amazon SQS キューに保存されているオブジェクトへの参照を送信します。

> **Note**
>
> Amazon SQS 拡張クライアントライブラリは、標準キューと FIFO キューの両方と互換性があります。

Java と Amazon S3 を使用した大量の Amazon SQS メッセージの管理

Amazon S3 で Java 用 Amazon SQS 拡張クライアントライブラリを使用して、特に 256 KB から 2 GB までのペイロードで、大きな Amazon SQS メッセージを管理します。 Amazon S3 ライブラリはメッセージペイロードを Amazon S3 バケットに保存し、Amazon SQS キューに保存されているオブジェクトへの参照を含むメッセージを送信します。

Java 用 Amazon SQS 拡張クライアントライブラリを使用すると、次のことができます。

- メッセージを常に Amazon S3 に保存するか、メッセージのサイズが 256 KB を超えた場合にのみ保存するかを指定する
- S3 バケットに保存されている単一のメッセージオブジェクトを参照するメッセージを送信する
- Amazon S3 バケットからメッセージオブジェクトを取得する
- Amazon S3 バケットからメッセージオブジェクトを削除する

前提条件

次の例では、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。

サンプルコードを実行する前に、 AWS 認証情報を設定します。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。

SDK for Java と Java 用 Amazon SQS 拡張クライアントライブラリには、J2SE Development Kit 8.0 以降が必要です。

> **Note**
>
> Java 用 Amazon SQS 拡張クライアントライブラリで Amazon SQS メッセージを管理できるのは、 AWS SDK for Javaで Amazon S3 を使用した場合のみです。これは AWS

CLI、、Amazon SQS コンソール、Amazon SQS HTTP API、またはその他の SDKs では AWS 実行できません。

AWS SDK for Java 2.x 例: Amazon S3 を使用して大きな Amazon SQS メッセージを管理する

次の SDK for SDK for Java 2.x の例では、Java 用拡張クライアントライブラリを使用して大規模なメッセージを操作します。コンストラクタでは、次のコードを使用します。

- ランダムな名前の Amazon S3 バケットを作成します。
- で始まる SQS キューを作成します `MyQueue`
- 標準の Java SDK Amazon S3 クライアントを のインスタンスにラップします。`AmazonSQSExtendedClient`

`sendAnReceiveMessage` メソッドでは、256 KB (標準の最大メッセージサイズ) を超えるため、Amazon S3 バケットに保存されているランダムなメッセージを送信します。最後に、 メソッドはメッセージを取得し、その情報をコンソールに表示します。

完全な例については、「」を参照してくださいhttps://github.com/awsdocs/aws-doc-sdk-examples/blob/94d1b24df12deda0f4fd91433b8231fed6d18b85/javav2/example_code/sqs/src/main/java/com/example/sqs/SqsExtendedClientExample.java#L1。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

             import com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
```

```
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.BucketLifecycleConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ExpirationStatus;
import software.amazon.awssdk.services.s3.model.LifecycleExpiration;
import software.amazon.awssdk.services.s3.model.LifecycleRule;
import software.amazon.awssdk.services.s3.model.LifecycleRuleFilter;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsRequest;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import software.amazon.awssdk.services.s3.model.PutBucketLifecycleConfigurationRequest;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;

import java.util.Arrays;
import java.util.List;
import java.util.UUID;


/**
 * Examples of using Amazon SQS Extended Client Library for Java 2.x
 *
 */
public class SqsExtendedClientExamples {
    // Create an Amazon S3 bucket with a random name.
    private final static String amzn-s3-demo-bucket = UUID.randomUUID() + "-"
            + DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());

    public static void main(String[] args) {

        /*
```

```
         * Create a new instance of the builder with all defaults (credentials
         * and region) set automatically. For more information, see
         * Creating Service Clients in the AWS SDK for Java Developer Guide.
         */
        final S3Client s3 = S3Client.create();

        /*
         * Set the Amazon S3 bucket name, and then set a lifecycle rule on the
         * bucket to permanently delete objects 14 days after each object's
         * creation date.
         */
        final LifecycleRule lifeCycleRule = LifecycleRule.builder()
                .expiration(LifecycleExpiration.builder().days(14).build())
                .filter(LifecycleRuleFilter.builder().prefix("").build())
                .status(ExpirationStatus.ENABLED)
                .build();
        final BucketLifecycleConfiguration lifecycleConfig =
 BucketLifecycleConfiguration.builder()
                .rules(lifeCycleRule)
                .build();

        // Create the bucket and configure it
        s3.createBucket(CreateBucketRequest.builder().bucket(amzn-s3-demo-
bucket).build());

 s3.putBucketLifecycleConfiguration(PutBucketLifecycleConfigurationRequest.builder()
                .bucket(amzn-s3-demo-bucket)
                .lifecycleConfiguration(lifecycleConfig)
                .build());
        System.out.println("Bucket created and configured.");

        // Set the Amazon SQS extended client configuration with large payload support
 enabled
        final ExtendedClientConfiguration extendedClientConfig = new
 ExtendedClientConfiguration().withPayloadSupportEnabled(s3, amzn-s3-demo-bucket);

        final SqsClient sqsExtended = new
 AmazonSQSExtendedClient(SqsClient.builder().build(), extendedClientConfig);

        // Create a long string of characters for the message object
        int stringLength = 300000;
        char[] chars = new char[stringLength];
        Arrays.fill(chars, 'x');
        final String myLongString = new String(chars);
```

```
        // Create a message queue for this example
        final String queueName = "MyQueue-" + UUID.randomUUID();
        final CreateQueueResponse createQueueResponse =
sqsExtended.createQueue(CreateQueueRequest.builder().queueName(queueName).build());
        final String myQueueUrl = createQueueResponse.queueUrl();
        System.out.println("Queue created.");

        // Send the message
        final SendMessageRequest sendMessageRequest = SendMessageRequest.builder()
                .queueUrl(myQueueUrl)
                .messageBody(myLongString)
                .build();
        sqsExtended.sendMessage(sendMessageRequest);
        System.out.println("Sent the message.");

        // Receive the message
        final ReceiveMessageResponse receiveMessageResponse =
sqsExtended.receiveMessage(ReceiveMessageRequest.builder().queueUrl(myQueueUrl).build());
        List<Message> messages = receiveMessageResponse.messages();

        // Print information about the message
        for (Message message : messages) {
            System.out.println("\nMessage received.");
            System.out.println("  ID: " + message.messageId());
            System.out.println("  Receipt handle: " + message.receiptHandle());
            System.out.println("  Message body (first 5 characters): " +
message.body().substring(0, 5));
        }

        // Delete the message, the queue, and the bucket
        final String messageReceiptHandle = messages.get(0).receiptHandle();

sqsExtended.deleteMessage(DeleteMessageRequest.builder().queueUrl(myQueueUrl).receiptHandle(me
        System.out.println("Deleted the message.");


sqsExtended.deleteQueue(DeleteQueueRequest.builder().queueUrl(myQueueUrl).build());
        System.out.println("Deleted the queue.");

        deleteBucketAndAllContents(s3);
        System.out.println("Deleted the bucket.");

    }
```

```
    private static void deleteBucketAndAllContents(S3Client client) {
        ListObjectsV2Response listObjectsResponse =
 client.listObjectsV2(ListObjectsV2Request.builder().bucket(amzn-s3-demo-
bucket).build());

        listObjectsResponse.contents().forEach(object -> {
            client.deleteObject(DeleteObjectRequest.builder().bucket(amzn-s3-demo-
bucket).key(object.key()).build());
        });

        ListObjectVersionsResponse listVersionsResponse =
 client.listObjectVersions(ListObjectVersionsRequest.builder().bucket(amzn-s3-demo-
bucket).build());

        listVersionsResponse.versions().forEach(version -> {
            client.deleteObject(DeleteObjectRequest.builder().bucket(amzn-s3-demo-
bucket).key(version.key()).versionId(version.versionId()).build());
        });

        client.deleteBucket(DeleteBucketRequest.builder().bucket(amzn-s3-demo-
bucket).build());
    }
}
```

Apache Maven を使用して、Java プロジェクト用の Amazon SQS 拡張クライアントを設定および構築したり、SDK 自体を構築したりできます。アプリケーションで使用する個々のモジュールを SDK から指定します。

```
<properties>
    <aws-java-sdk.version>2.20.153</aws-java-sdk.version>
</properties>

<dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>sqs</artifactId>
      <version>${aws-java-sdk.version}</version>
    </dependency>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
```

```
      <artifactId>s3</artifactId>
      <version>${aws-java-sdk.version}</version>
    </dependency>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>amazon-sqs-java-extended-client-lib</artifactId>
      <version>2.0.4</version>
    </dependency>

    <dependency>
      <groupId>joda-time</groupId>
      <artifactId>joda-time</artifactId>
      <version>2.12.6</version>
    </dependency>
</dependencies>
```

Python と Amazon S3 を使用した大量の Amazon SQS メッセージの管理

Amazon S3 で Python 用の Amazon SQS Amazon SQS 拡張クライアントライブラリを使用して、特に 256 KB ~ 2 GB のペイロードで大きな Amazon SQS メッセージを管理します。 Amazon S3 ライブラリはメッセージペイロードを Amazon S3 バケットに保存し、Amazon SQS キューに保存されているオブジェクトへの参照を含むメッセージを送信します。

Python 用 Amazon SQS 拡張クライアントライブラリを使用すると、次のことができます。

- ペイロードが常に Amazon S3 に保存されるか、ペイロードサイズが 256 KB を超えた場合にのみ Amazon S3 に保存されるかを指定します。
- Amazon S3 バケットに保存されている単一のメッセージオブジェクトを参照するメッセージを送信する
- Amazon S3 バケットから対応するペイロードオブジェクトを取得する
- Amazon S3 バケットから対応するペイロードオブジェクトを削除する

前提条件

Python 用 Amazon SQS 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。

- 必要な認証情報を持つ AWS アカウント。 AWS アカウントを作成するには、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、「認証情報」を参照してください。
- 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セットアップするには、「AWS SDK for Python デベロッパーガイド」の「AWS SDK for Python」ドキュメントを参照してください。
- Python 3.x (または以降) および `pip`。
- Python 用 Amazon SQS 拡張クライアントライブラリ (PyPI から利用可能)。

> **Note**
>
> Amazon SQS Extended Client Library for Python を使用して、 AWS SDK for Python でのみ Amazon S3 を使用して Amazon SQS メッセージを管理できます。 Amazon S3 これは、 AWS CLI、Amazon SQS コンソール、Amazon SQS HTTP API、またはその他の AWS SDKs では実行できません。

メッセージストレージの設定

Amazon SQS 拡張クライアントは、以下のメッセージ属性を使用して Amazon S3 のメッセージストレージオプションを設定します。

- `large_payload_support`: 大量のメッセージを保存するための Amazon S3 バケットの名前。
- `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。`False` の場合、256 KB 未満のメッセージは s3 バケットにシリアル化されません。デフォルトは `False` です。
- `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。

Python 用拡張クライアントライブラリを使用した大量の Amazon SQS メッセージの管理

次の例では、Amazon S3 バケットをランダムな名前で作成します。次に、`MyQueue` という名前の Amazon SQS を作成し、S3 バケットに保存されている 256 KB を超えるメッセージをキューに送信

します。最後に、コードはメッセージを取得し、そのメッセージに関する情報を返してから、メッセージ、キュー、バケットを削除します。

```
import boto3
import sqs_extended_client

#Set the Amazon SQS extended client configuration with large payload.
sqs_extended_client = boto3.client("sqs", region_name="us-east-1")
sqs_extended_client.large_payload_support = "amzn-s3-demo-bucket"
sqs_extended_client.use_legacy_attribute = False


# Create an SQS message queue for this example. Then, extract the queue URL.
queue = sqs_extended_client.create_queue(
    QueueName = "MyQueue"
)
queue_url = sqs_extended_client.get_queue_url(
    QueueName = "MyQueue"
)['QueueUrl']


# Create the S3 bucket and allow message objects to be stored in the bucket.
sqs_extended_client.s3_client.create_bucket(Bucket=sqs_extended_client.large_payload_support)

# Sending a large message
small_message = "s"
large_message = small_message * 300000 # Shall cross the limit of 256 KB

send_message_response = sqs_extended_client.send_message(
    QueueUrl=queue_url,
    MessageBody=large_message
)
assert send_message_response['ResponseMetadata']['HTTPStatusCode'] == 200

# Receiving the large message
receive_message_response = sqs_extended_client.receive_message(
    QueueUrl=queue_url,
    MessageAttributeNames=['All']
)
assert receive_message_response['Messages'][0]['Body'] == large_message
receipt_handle = receive_message_response['Messages'][0]['ReceiptHandle']
```

```
# Deleting the large message
# Set to True for deleting the payload from S3
sqs_extended_client.delete_payload_from_s3 = True
delete_message_response = sqs_extended_client.delete_message(
    QueueUrl=queue_url,
    ReceiptHandle=receipt_handle
)

assert delete_message_response['ResponseMetadata']['HTTPStatusCode'] == 200

# Deleting the queue
delete_queue_response = sqs_extended_client.delete_queue(
    QueueUrl=queue_url
)

assert delete_queue_response['ResponseMetadata']['HTTPStatusCode'] == 200
```

Amazon SQS コンソールを使用した Amazon SQS キューの設定

Amazon SQS コンソールを使用して、Amazon SQS キューと機能を設定および管理します。以下の操作も可能です。

- セキュリティを強化するためにサーバー側の暗号化を有効にします。
- デッドレターキューを関連付けて、未処理のメッセージを処理します。
- イベント駆動型処理のために Lambda 関数を呼び出すトリガーを設定します。

Amazon SQS の属性ベースのアクセス制御

ABAC とは

属性ベースのアクセス制御 (ABAC) は、ユーザーおよび AWS リソースにアタッチされたタグに基づいてアクセス許可を定義する認可プロセスです。ABAC は、属性と値に基づいてきめ細かく柔軟なアクセス制御を実現し、再構成されたロールベースのポリシーに関連するセキュリティリスクを軽減し、監査とアクセスポリシー管理を一元化します。ABAC の詳細については、「IAM ユーザーガイド」の「AWSの ABAC とは」を参照してください。

Amazon SQS は、Amazon SQS キューに関連付けられているタグとエイリアスに基づいて Amazon SQS キューへのアクセスを制御できるようにすることで、ABAC をサポートしています。Amazon SQS の ABAC を有効にするタグおよびエイリアスの条件キーは、ポリシーを編集したりグラントを管理することなく、IAM プリンシパルが Amazon SQS キューを使用することを許可します。ABAC の条件キーの詳細については、「サービス認可リファレンス」の「Amazon SQS の条件キー」を参照してください。

ABAC では、タグを使用して Amazon SQS キューの IAM アクセス許可とポリシーを設定できます。これにより、アクセス許可管理をスケールできます。各ビジネスロールに追加するタグを使用して、IAM で単一のアクセス許可ポリシーを作成できます。新しいリソースを追加するたびにポリシーを更新する必要はありません。IAM プリンシパルにタグをアタッチして ABAC ポリシーを作成することもできます。呼び出しを行う IAM ユーザーロールのタグが Amazon SQS キュータグと一致した場合に Amazon SQS 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。でのタグ付けの詳細については AWS、AWS 「タグ付け戦略」と「」を参照してくださいAmazon SQSコスト配分タグ。

> **Note**
>
> Amazon SQS の ABAC は現在、Amazon SQS が利用可能なすべての AWS 商用リージョンで使用できますが、以下の例外があります。
>
> - 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)
> - 欧州 (スペイン)
> - 欧州 (チューリッヒ)

Amazon SQS で ABAC を使用すべき理由は何ですか。

Amazon SQS で ABAC を使用すると、以下のようなメリットがあります。

- Amazon SQS 用 ABAC では、必要なアクセス許可ポリシーが少なくて済みます。職務機能ごとに異なるポリシーを作成する必要はありません。複数のキューに適用されるリソースタグとリクエストタグを使用できるため、運用上のオーバーヘッドが軽減できます。
- ABAC を使用すると、チームを迅速にスケールできます。リソースの作成時に適切なタグが付けられると、新しいリソースのアクセス許可はタグに基づいて自動的に付与されます。
- IAM プリンシパルのアクセス許可を使用して、リソースへのアクセスを制限します。IAM プリンシパルのタグを作成し、そのタグを使用して IAM プリンシパルのタグと一致する特定のアクションへのアクセスを制限できます。これにより、リクエストのアクセス許可を付与するプロセスを自動化できます。
- リソースにアクセスしているユーザーを追跡できます。セッションの ID は、 AWS CloudTrailのユーザー属性を調べることで判断できます。

トピック

- Amazon SQS でのアクセス制御のタグ付け
- IAM ユーザーと Amazon SQS キューの作成
- Amazon SQS での属性ベースのアクセス制御のテスト

Amazon SQS でのアクセス制御のタグ付け

以下は、Amazon SQS でのアクセスコントロールにタグを使用する例です。IAM ポリシーは、キー environment および値 production のタグを持つリソースタグを含むすべてのキューに対して、IAM ユーザーにすべての Amazon SQS アクションを制限します。詳細については、「タグを使用した属性ベースのアクセスコントロール」と AWS 「Organizations」を参照してください。

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessForProd",
      "Effect": "Deny",
      "Action": "sqs:*",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "prod"
        }
      }
    }
  ]
}
```

IAM ユーザーと Amazon SQS キューの作成

次の例では、 AWS Management Console と を使用して Amazon SQS へのアクセスを制御する ABAC ポリシーを作成する方法について説明します AWS CloudFormation。

の使用 AWS Management Console

IAM ユーザーの作成

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーションペインで [ユーザー] を選択します。
3. [ユーザーを追加] を選択し、[ユーザー名] テキストボックスに名前を入力します。

4. [アクセスキー - プログラムによるアクセス] ボックスを選択し、[次へ: 許可] を選びます。
5. [Next:Tags] (次のステップ: タグ) を選択します。
6. タグのキーを `environment`、タグの値を `beta` として追加します。
7. [次の手順: 確認]、[作成] の順に選択します。
8. アクセスキー ID とシークレットアクセスキーを安全な場所にコピーして保存します。

IAM ユーザーのアクセス許可を追加する

1. 作成した IAM ユーザーを選択します。
2. [Add inline policy] (インラインポリシーの追加) を選択します。
3. [JSON] タブに、以下のポリシーを貼り付けます。

   JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowAccessForSameResTag",
      "Effect": "Allow",
      "Action": [
        "sqs:SendMessage",
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/environment": "${aws:PrincipalTag/environment}"
        }
      }
    },
    {
      "Sid": "AllowAccessForSameReqTag",
      "Effect": "Allow",
      "Action": [
        "sqs:CreateQueue",
        "sqs:DeleteQueue",
        "sqs:SetQueueAttributes",
```

```
          "sqs:tagqueue"
        ],
        "Resource": "*",
        "Condition": {
          "StringEquals": {
            "aws:RequestTag/environment": "${aws:PrincipalTag/environment}"
          }
        }
      },
    ]
  }
```

4. [ポリシーの確認] を選択します。
5. [Create policy] (ポリシーの作成) を選択します。

の使用 AWS CloudFormation

次のサンプル AWS CloudFormation テンプレートを使用して、インラインポリシーと Amazon SQS キューがアタッチされた IAM ユーザーを作成します。

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "CloudFormation template to create IAM user with custom inline policy"
Resources:
    IAMPolicy:
        Type: "AWS::IAM::Policy"
        Properties:
            PolicyDocument: |
                {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "AllowAccessForSameResTag",
                            "Effect": "Allow",
                            "Action": [
                                "sqs:SendMessage",
                                "sqs:ReceiveMessage",
                                "sqs:DeleteMessage"
                            ],
                            "Resource": "*",
                            "Condition": {
                                "StringEquals": {
```

```
                                    "aws:ResourceTag/environment": "${aws:PrincipalTag/
environment}"
                                }
                            }
                        },
                        {
                            "Sid": "AllowAccessForSameReqTag",
                            "Effect": "Allow",
                            "Action": [
                                "sqs:CreateQueue",
                                "sqs:DeleteQueue",
                                "sqs:SetQueueAttributes",
                                "sqs:tagqueue"
                            ],
                            "Resource": "*",
                            "Condition": {
                                "StringEquals": {
                                    "aws:RequestTag/environment": "${aws:PrincipalTag/
environment}"
                                }
                            }
                        },
                        {
                            "Sid": "DenyAccessForProd",
                            "Effect": "Deny",
                            "Action": "sqs:*",
                            "Resource": "*",
                            "Condition": {
                                "StringEquals": {
                                    "aws:ResourceTag/stage": "prod"
                                }
                            }
                        }
                    ]
                }

            Users:
              - "testUser"
            PolicyName: tagQueuePolicy

    IAMUser:
        Type: "AWS::IAM::User"
        Properties:
            Path: "/"
```

```
            UserName: "testUser"
            Tags:
              -
                Key: "environment"
                Value: "beta"
```

Amazon SQS での属性ベースのアクセス制御のテスト

以下の例は、Amazon SQS で属性ベースのアクセス制御をテストする方法について示しています。

タグキーを environment に設定し、タグ値を prod に設定してキューを作成します。

この AWS CLI コマンドを実行して、タグキーを環境に設定し、タグ値を prod に設定してキューの作成をテストします。 AWS CLI がない場合は、マシン用にダウンロードして設定できます。

```
aws sqs create-queue --queue-name prodQueue –region us-east-1 –tags "environment=prod"
```

Amazon SQS エンドポイントから `AccessDenied` エラーを受け取ります。

```
An error occurred (AccessDenied) when calling the CreateQueue operation: Access to the
 resource <queueUrl> is denied.
```

これは、IAM ユーザーのタグ値が `CreateQueue` API コールで渡されたタグと一致しないためです。キーを `environment` に、値を `beta` に設定したタグを IAM ユーザーに適用したことを思い出してください。

タグキーを environment に設定し、タグ値を beta に設定してキューを作成する

この CLI コマンドを実行して、タグキーを `environment` に設定し、タグ値を `beta` に設定したキューの作成をテストします。

```
aws sqs create-queue --queue-name betaQueue –region us-east-1 –tags "environment=beta"
```

キューが正常に作成されたことを確認する次のようなメッセージが表示されます。

```
{
"QueueUrl": "<queueUrl>“
}
```

キューへメッセージを送信する

この CLI コマンドを実行して、キューへのメッセージ送信をテストします。

```
aws sqs send-message --queue-url <queueUrl> --message-body testMessage
```

レスポンスには、Amazon SQS キューへのメッセージ配信が成功したことが示されます。IAM ユーザーアクセス許可により、`beta` タグを持つキューにメッセージを送信できます。レスポンスにはメッセージを含む `MD5OfMessageBody` と `MessageId` が含まれます。

```
{
"MD5OfMessageBody": "<MD5OfMessageBody>",
"MessageId": "<MessageId>"
}
```

Amazon SQS コンソールを使用したキューのパラメータの設定

キューを作成または編集するときに、次のパラメータを設定できます。

- 可視性タイムアウト— キューから受信したメッセージが (あるコンシューマによって) 他のメッセージコンシューマーに表示されない時間の長さ。詳細については、「可視性」を参照してください。

> **Note**
> コンソールを使用して可視性タイムアウトを設定すると、キュー内のすべてのメッセージのタイムアウト値が設定されます。単一または複数のメッセージのタイムアウトを設定するには、いずれかの AWS SDKsを使用する必要があります。

- メッセージの保持期間— キューに残っているメッセージを Amazon SQS が保持する時間は。デフォルトでは、キューは4日間メッセージを保持します。最大14日間までメッセージを保持するようにキューを設定できます。詳細については、「メッセージの保持期間」を参照してください。
- 配信の遅延—キューに追加されたメッセージを配信する前に Amazon SQSが遅延する時間。詳細については、「配信の遅延」を参照してください。
- メッセージの最大サイズ—このキューの最大メッセージサイズ。詳細については、「メッセージの最大サイズ」を参照してください。

- メッセージの受信待ち時間—キューが受信リクエストを受け取った後、Amazon SQS がメッセージが使用可能になるまでの待機する最大時間。詳細については、「Amazon SQSショートポーリングとロングポーリング」を参照してください。
- コンテンツベースの重複除外を有効—Amazon SQS は、メッセージの本文に基づいて重複除外ID を自動的に作成できます。詳細については、「Amazon SQS FIFO キュー」を参照してください。
- 高スループット FIFOを有効にする—キューのメッセージの高スループットを有効にするために使用します。このオプションを選択すると、関連するオプションが (重複除外スコープそしてFIFO スループットの制限) を使用して、FIFOキューの高スループットを有効にするために必要な設定に変更されます。詳細については、「Amazon SQS の FIFO キューの高スループット」および「Amazon SQS のメッセージキュー」を参照してください。
- 許可ポリシーの再実行: どのソースキューがこのキューをデッドレターキューとして使用できるを定義します。詳細については、「Amazon SQS でのデッドレターキューの使用 」を参照してください。

既存のキュー (コンソール)のキューパラメータを設定するには

1. Amazon SQS コンソール を開きますhttps://console.aws.amazon.com/sqs/。
2. ナビゲーションペインで [キュー] を選択します。キューを選択し、編集を選択します。
3. 設定セクションにスクロールします。
4. 可視性タイムアウトを使用する場合、期間と単位を入力します。範囲は0秒から12時間です。デフォルト値は30秒です。
5. メッセージの保持期間を使用する場合、期間と単位を入力します。有効範囲は1分から14日です。デフォルト値は4日です。
6. 標準キューの場合は、メッセージの受信待ち時間の値を入力します。範囲は 0から20秒です。デフォルト値は0秒で、ショートポーリングで設定します。0以外の値を指定すると、ロングポーリングが設定されます。
7. 配信の遅延を使用する場合、期間と単位を入力します。範囲は0秒から15 分です。デフォルト値は0 秒です。
8. 最大メッセージサイズを使用する場合、値を入力します。範囲は 1 KiB ~ 1024 KiB です。デフォルト値は 1024 KiB です。
9. FIFO キューの場合は、コンテンツベースの重複除外を有効にするためにコンテンツベースの重複除外を有効にします。デフォルト設定は無効です。

10. (オプション) FIFO キューの場合、キュー内のメッセージの送受信でより高スループットを有効にするには、[高スループット FIFO を有効にする] を選択します。

    このオプションを選択すると、関連するオプション (重複除外のスコープおよびFIFO スループットの制限) を使用して、FIFO キューの高スループットを有効にするために必要な設定に変更されます。高スループット FIFO の使用に必要な設定のいずれかを変更すると、キューに対して通常のスループットが有効になり、指定されたとおりに重複除外が実行されます。詳細については、「Amazon SQS の FIFO キューの高スループット」および「Amazon SQS のメッセージキュー」を参照してください。
11. [許可ポリシーの再実行] で、[有効] を選択します。[すべて許可] (デフォルト)、[キュー別]、または [すべて拒否] から選択します。キュー別を選んだとき、 Amazon リソースネーム (ARN) で最大10個のソースキューのリストを指定します。
12. キューパラメータの設定が完了したら、保存を選択します。

Amazon SQS でのアクセスポリシーの設定

キューを編集するときは、キューを操作できるユーザーを制御するようにアクセスポリシーを設定できます。

- アクセスポリシーは、キューへのアクセス許可を持つアカウント、ユーザー、ロールを定義します。
- 、、 `SendMessage` `ReceiveMessage`などの許可されたアクションを指定します`DeleteMessage`。
- デフォルトでは、キュー所有者のみがメッセージを送受信するアクセス許可を持ちます。

既存のキューのアクセスポリシーを設定するには (コンソール)

1. Amazon SQS コンソール (https://console.aws.amazon.com/sqs/) を開きます。
2. ナビゲーションペインで [キュー] を選択します。
3. キューを選択し、[編集] を選択します。
4. アクセスポリシーセクションにスクロール。
5. 入力ボックスでアクセスポリシーステートメントを編集します。アクセスポリシーステートメントの詳細については、「Amazon SQSでの Identity and Access Management」を参照してください。
6. アクセスポリシーの設定が完了したら、保存を選びます。

SQS マネージド暗号化キーを使用したキューに対するサーバー側の暗号化の設定

Amazon SQS マネージド SSE (SSE-SQS) では、Amazon SQS が管理するデフォルトのサーバー側の暗号化オプションに加えて、SQS マネージド暗号化キーを使用して、メッセージキューを介して送信される機密データを保護する、カスタムマネージドサーバー側の暗号化を作成できます。SSE-SQSでは、暗号化キーの作成および管理、データを暗号化のためにコードを修正する必要はありません。SSE-SQSを使用すると、データを安全に送信し、追加の費用なしで厳格な暗号化コンプライアンスと規制要件に対応できます。

SSE-SQS は、256 ビットの高度暗号化スタンダード (AES-256)の 暗号化を使用して保存中のデータを保護します。SSEはAmazon SQSサービスがメッセージを受信するとすぐに暗号化します。Amazon SQS は暗号化された形式でメッセージを保存し、承認済みのコンシューマーに送信した場合のみ解読されます。

> **Note**
>
> - デフォルトの SSE オプションは、暗号化属性を指定せずにキューを作成した場合にのみ有効です。
> - Amazon SQS では、キューの暗号化をすべてオフにすることができます。したがって、KMS-SSE をオフにしても SQS-SSE は自動的には有効になりません。KMS-SSE を無効にした後に SQS-SSE を有効にする場合は、リクエストに属性の変更を追加する必要があります。

キューにSSE-SQS暗号化を設定するには(コンソール)

> **Note**
>
> HTTP (非 TLS) エンドポイントを使用して作成された新しいキューは、デフォルトで SSE-SQS 暗号化を有効化しません。HTTPS または署名バージョン 4 エンドポイントを使用して Amazon SQS キューを作成するのがセキュリティのベストプラクティスです。

1. Amazon SQSコンソール (https://console.aws.amazon.com/sqs/) を開きます。
2. ナビゲーションペインで [Queues(キュー)] を選択します。
3. キューを選択し、{編集}を選択します。

4. 暗号化を拡張します。
5. サーバー側の暗号化を使用するには、[有効] (デフォルト) を選択します。

> **Note**
>
> SSE を有効にすると、暗号化されたキューへの匿名の `SendMessage` リクエストと `ReceiveMessage` リクエストは拒否されます。Amazon SQS のセキュリティベストプラクティスでは、匿名リクエストを使用しないことを推奨しています。Amazon SQS キューに匿名リクエストを送信する場合は、必ず SSE を無効にしてください。

6. Amazon SQSキー (SSE-SQS)を選択します。このオプションは追加料金なしで使用されます。
7. [保存] を選択します。

Amazon SQS を使用したキューに対するサーバー側の暗号化の設定

キューのメッセージのデータを保護するために、Amazon SQS では、新しく作成されたキューのサーバー側の暗号化がデフォルトで有効になっています。Amazon SQS は、Amazon Web Services キー管理サービス (Amazon Web Services KMS) と統合してサーバー側の暗号化 (SSE) 用の KMS キーを管理します。SSE の使用詳細については、「Amazon SQS での保管中の暗号化」をご参照ください。

キューに割り当てる KMS キーは、キューの使用が承認されたすべてのプリンシパルの許可を含むキーポリシーが必要です。詳細については、キー管理をご参照ください。

KMS 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 SQS コンソールで KMS キーに関する情報を閲覧できません。これらの許可を付与するように、KMS キーの所有者へ依頼してください。詳しい情報については、キー管理をご参照ください。

キューの作成または編集する場合は、SSE-KMS を設定できます。

既存のキューに SSE-KMSを設定するには(コンソール)

1. Amazon SQSコンソールを開きます https://console.aws.amazon.com/sqs/。
2. ナビゲーションペインで [Queues(キュー)] を選択します。
3. キューを選択し、{編集}を選択します。

4. 暗号化を拡張します。
5. サーバー側の暗号化を使用するには、[有効] (デフォルト) を選択します。

> **Note**
>
> SSE を有効にすると、暗号化されたキューへの匿名の `SendMessage` リクエストと `ReceiveMessage` リクエストは拒否されます。Amazon SQS のセキュリティベストプラクティスでは、匿名リクエストを使用しないことを推奨しています。Amazon SQS キューに匿名リクエストを送信する場合は、必ず SSE を無効にしてください。

6. AWS キーマネジメントサービスキー (SSE-KMS)を選びます。

   コンソールは [Description] (概要)、[Account] (アカウント)、KMS キーの [KMS key ARN] (KMS キー ARN) を表示します。
7. キューの KMS キー ID 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。
   a. [Choose a KMS key alias] (KMS キーのエイリアスを選択) オプションを選択します。
   b. デフォルトキーは、Amazon SQS 用の Amazon Web Services マネージド KMS キーです。このキーを使用する場合、[KMS key] (KMS キー) リストから選択してください。
   c. Amazon Web Services アカウントでカスタム KMS キーを使用する場合、[KMS key] (KMS キー) リストから選択してください。カスタム KMS キーの作成手順については、Amazon Web Services キー管理サービスデベロッパーガイドのキー作成をご参照ください。
   d. リストにないカスタム KMS キーを使用、または別の Amazon Web Services アカウントのカスタム KMS キーを使用する場合、[Enter the KMS key alias] (KMS キーのエイリアス入力) を選択し、KMS キー Amazon リソースネーム (ARN) を入力します。
8. (任意) [Data key reuse period] (データキー再利用期間) について、1 分から 24 時間の範囲内で値を指定します。デフォルト値は 5 分です。詳細については、「データキー再利用期間について」を参照してください。
9. SSE-KMS の設定が完了したら、保存を選択します。

Amazon SQS コンソールを使用したキューのコスト配分タグの設定

Amazon SQS キューを整理して識別するには、コスト配分タグを追加できます。詳細については、「Amazon SQSコスト配分タグ」を参照してください。

- 詳細ページのタグ付けタブには、キューのタグが表示されます。
- キューを作成または編集するときに、タグを追加または変更できます。

既存のキューのタグを設定するには(コンソール)

1. Amazon SQSコンソールを開きます https://console.aws.amazon.com/sqs/。
2. ナビゲーションペインで [キュー] を選択します。
3. キューを選択し、編集を選択します。
4. タグセクションまでスクロールします。
5. キュータグを追加、変更、または削除するには
   a. タグを追加するには、{新しタグを追加}を選択して{キーを}と{値}を入力し、{新しいタグの追加}を選択します。
   b. タグを更新するには、変更します。キーと値を変更します。
   c. タグを削除するには、キー値ペアの横にある {削除} を選択します。
6. タグの設定が完了したら、保存を選択します。

Amazon SQS コンソールを使用した Amazon SNS トピックへのキューのサブスクライブ

1 つ以上の Amazon SQS キューを Amazon SNS トピックにサブスクライブできます。トピックにメッセージを発行すると、Amazon SNS はサブスクライブされた各キューにメッセージを送信します。Amazon SQS はサブスクリプションを管理し、必要なアクセス許可を処理します。Amazon SNS の詳細については、Amazon Simple Notification Service デベロッパーガイドの、「Amazon SNS とは」を参照してください。

Amazon SNS トピックに Amazon SQS Amazon SNS キューをサブスクライブすると、Amazon SNS は HTTPS を使用してメッセージを Amazon SQS に転送します。暗号化された Amazon SQS キューでの Amazon SNS の使用については、「AWS サービスの KMS アクセス許可を設定する」を参照してください。

> **Important**
>
> Amazon SQS は、アクセスポリシーごとに最大 20 個のステートメントをサポートします。Amazon SNS トピックにサブスクライブすると、このようなステートメントが 1 つ追加されます。この数を超えると、トピックサブスクリプションの配信が失敗します。

Amazon SNS トピックにキューをサブスクライブするには (コンソール)

1. Amazon SQSコンソール (https://console.aws.amazon.com/sqs/) を開きます。
2. ナビゲーションペインで [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. [Actions](アクション) メニューで、[Subscribe to Amazon SNS topic](Amazon SNS トピックを購読する) を選択します。
5. このキューで使用できる Amazon SNS トピックを指定するメニューから、キューの Amazon SNS トピックを選択します。

   SNS トピックがリストされていない場合は、Amazon SNSトピック ARN を入力」を選択し、トピックの Amazon リソースネーム (ARN) を入力します。
6. [保存] を選択します。
7. サブスクリプションを確認するには、 トピックにメッセージを発行し、キューにメッセージを表示します。詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「Amazon SNS メッセージの公開」を参照してください。

クロスアカウントサブスクリプション

Amazon SQS キューと Amazon SNS トピックが異なる場合は AWS アカウント、追加のアクセス許可が必要です。

トピック所有者 (アカウント A)

Amazon SNS トピックのアクセスポリシーを変更して、Amazon SQS キューの が AWS アカウントサブスクライブできるようにします。ポリシーステートメントの例 :

```
{
  "Effect": "Allow",
```

```
  "Principal": { "AWS": "arn:aws:iam::111122223333:root" },
  "Action": "sns:Subscribe",
  "Resource": "arn:aws:sns:us-east-1:123456789012:MyTopic"
}
```

このポリシーにより、アカウントは 111122223333にサブスクライブできますMyTopic。

キュー所有者 (アカウント B)

Amazon SQS Amazon SNS キューのアクセスポリシーを変更します。ポリシーステートメントの例：

```
{
  "Effect": "Allow",
  "Principal": { "Service": "sns.amazonaws.com" },
  "Action": "sqs:SendMessage",
  "Resource": "arn:aws:sqs:us-east-1:111122223333:MyQueue",
  "Condition": {
    "ArnEquals": { "aws:SourceArn": "arn:aws:sns:us-east-1:123456789012:MyTopic" }
  }
}
```

このポリシーにより、 MyTopic は にメッセージを送信できますMyQueue。

クロスリージョンサブスクリプション

別の で Amazon SNS トピックをサブスクライブするには AWS リージョン、以下を確認してください。

- Amazon SNS トピックのアクセスポリシーは、クロスリージョンサブスクリプションを許可します。
- Amazon SQS キューのアクセスポリシーは、Amazon SNS トピックがリージョン間でメッセージを送信することを許可します。

詳細については、「Amazon Simple Notification Service デベロッパーガイド」の「別のリージョンの Amazon SQS キューまたは関数に Amazon SNS メッセージを送信 Amazon SQS AWS Lambda する」を参照してください。

AWS Lambda 関数をトリガーするように Amazon SQS キューを設定する

Lambda 関数を使用して、Amazon SQS キューからのメッセージを処理できます。Lambda はキューをポーリングし、関数を同期的に呼び出して、メッセージのバッチをイベントとして渡します。

可視性タイムアウトの設定

キューの可視性タイムアウトを関数タイムアウトの少なくとも 6 倍に設定します。これにより、Lambda は、前のバッチの処理中に関数がスロットリングされた場合に再試行するのに十分な時間を確保できます。

デッドレターキュー (DLQ) の使用

Lambda 関数が処理に失敗したメッセージをキャプチャするデッドレターキューを指定します。

複数のキューと関数の処理

Lambda 関数は、キューごとに個別のイベントソースを作成することで、複数のキューを処理できます。複数の Lambda 関数を同じキューに関連付けることもできます。

暗号化されたキューのアクセス許可

暗号化されたキューを Lambda関数に関連付けても Lambda がメッセージをポーリングしない場合は、`kms:Decrypt`Lambda 実行ロールへのアクセスを許可します。

制限事項

キューと Lambda 関数は同じ にある必要があります AWS リージョン。

デフォルトキー (Amazon SQS のAWS マネージド KMS キー) を使用する暗号化されたキューは、別の で Lambda 関数を呼び出すことはできません AWS アカウント。

実装の詳細については、「 AWS Lambda デベロッパーガイド」のAmazon SQSでの の使用 AWS Lambda」を参照してください。

前提条件

Lambda関数トリガーを設定するには、以下の要件を満たしている必要があります。

- ユーザーを使用する場合、Amazon SQSロールに以下のアクセス許可が含まれている必要があります。

  - `lambda:CreateEventSourceMapping`
  - `lambda:ListEventSourceMappings`
  - `lambda:ListFunctions`
- Lambda実行ロールに以下のアクセス許可が含まれている必要があります。
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `sqs:ReceiveMessage`
- 暗号化されたキューを Lambda 関数に関連付ける場合は、`kms:Decrypt`Lambda 実行ロールへのアクセス許可をする必要があります。

詳細については、「Amazon SQS でのアクセス管理の概要」を参照してください。

Lambda関数(コンソール)をトリガーするためにキューを設定

1. Amazon SQSコンソール を開きますhttps://console.aws.amazon.com/sqs/。
2. ナビゲーションペインで [Queues(キュー)] を選択します。
3. [キュー]ページで、設定するキューを選択します。
4. キューのページでLambdaトリガータブを選択します。
5. Lambdaトリガーページで、Lambda トリガーを選択します。

   リストに、必要な Lambdaトリガーが含まれていない場合は、Lambda関数トリガーを設定するを選択します。Lambda関数のAmazon リソースネーム (ARN)を入力するか、既存のリソースを選択します。次に、[Save]を選択します。
6. [保存] を選択します。コンソールは設定を保存し、キューのの詳細ページを表示します。

   詳細ページでLambda トリガータブに Lambda 関数とそのステータスが表示されます。Lambda関数がキューに関連付けられるまで約1分かかります。
7. 設定の結果を確認するには、キューにメッセージを送信して、トリガーされたLambdaコンソールの Lambda関数が表示できます。

Amazon EventBridge を使用して AWS サービスから Amazon SQS への通知を自動化する

Amazon EventBridge を使用すると、アプリケーションの問題やリソースの変更などのイベントをほぼリアルタイムで自動化 AWS のサービス して対応できます。

- ルールを作成して特定のイベントをフィルタリングし、ルールがイベントに一致するときに自動アクションを定義できます。
- EventBridge は、JSON 形式でイベントを受信する Amazon SQS 標準キューや FIFO キューなど、複数のターゲットをサポートしています。

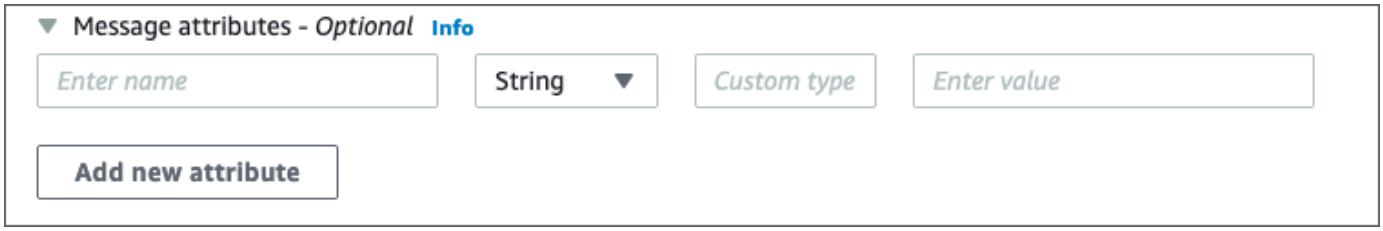
詳細については、『Amazon EventBridge ユーザーガイド』の「Amazon EventBridge ターゲット」を参照してください。

Amazon SQS を使用した属性を含むメッセージの送信

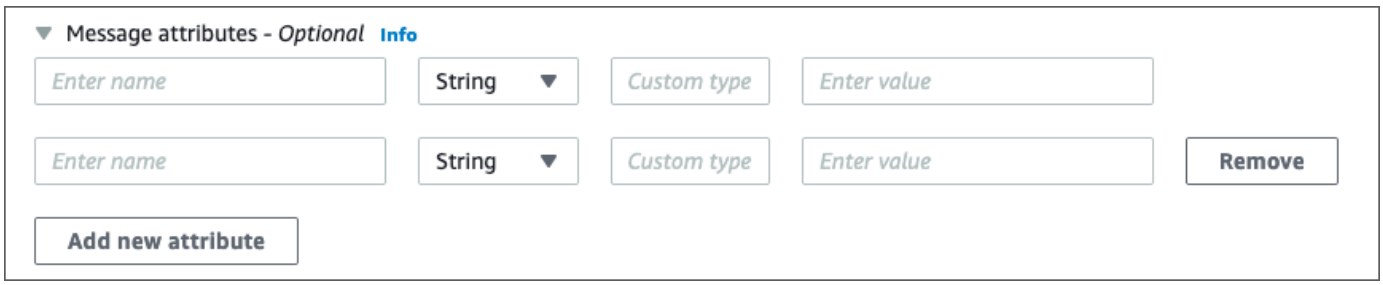
標準キューと FIFO キューでは、タイムスタンプ、地理空間データ、署名、識別子 などの構造化メタデータをメッセージに含めることができます。詳細については、「Amazon SQSメッセージ属性」を参照してください。

Amazon SQS コンソールを使用して、属性を含むメッセージをキューに送信するには

1. Amazon SQSコンソール (https://console.aws.amazon.com/sqs/) を開きます。
2. ナビゲーションペインで [Queues(キュー)]を選択します。
3. キュー ページで、キュー を選択します。
4. [メッセージの送信と受信] を選択します。
5. メッセージ属性のパラメータを入力します。
   a. {名前テキストボックス}に、[固有の名前]を256文字以下で(入力)します。
   b. 属性タイプについては、文字列、数値、 またはバイナリを選択します。
   c. (オプション) カスタムデータタイプを入力します。たとえば、**`byte`**、**`int`**、 または**`float`**のカスタムデータタイプとして数値を追加することができます。
   d. {値]}テキストボックスに、[メッセージ属性の値]を(入力)します。



6. 属性を追加するには、[新しい属性を追加] を選択します。



7. メッセージを送信する前に属性の値をいつでも変更することができます。
8. 属性を削除するには、{削除}を(選択)します。最初の属性を削除するには、{メッセージ属性}を(選択)します。
9. メッセージへの属性の追加を終了したら、{メッセージの送信} を(選択)します。メッセージが送信され、コンソールに成功メッセージが表示されます。送信したメッセージのメッセージ属性に関する情報を表示するには、{詳細を表示}を選択します。{実行}を (選択) してメッセージの詳細ダイアログボックス閉じます。

Amazon SQS のベストプラクティス

Amazon SQSはメッセージキューを管理および処理し、アプリケーションのさまざまな部分でメッセージを確実かつ大規模に交換できるようにします。このトピックでは、ロングポーリングを使用した空のレスポンスの削減、処理エラーを処理するデッドレターキューの実装、セキュリティのためのキューアクセス許可の最適化など、運用上の重要なベストプラクティスについて説明します。

トピック

- Amazon SQS のエラー処理および問題ありのメッセージ
- Amazon SQS のメッセージ重複排除とグループ化
- Amazon SQS のメッセージ処理とタイミング

Amazon SQS のエラー処理および問題ありのメッセージ

このトピックでは、リクエストエラーの処理、問題ありのメッセージのキャプチャ、メッセージの信頼性を確保するためのデッドレターキュー保持の設定など、Amazon SQS での エラーの管理と軽減に関する詳細な手順を示します。

トピック

- Amazon SQS でのリクエストエラーの処理
- 問題ありのメッセージのキャプチャ
- Amazon SQS でのデッドレターキュー保持の設定

Amazon SQS でのリクエストエラーの処理

リクエストのエラーを処理するには、次の方法のいずれかを使用します:

- AWS SDK を使用する場合は、自動再試行とバックオフのロジックが既に用意されています。詳細については、「Amazon Web Services 全般のリファレンス」の「エラーの再試行と AWSでのエクスポネンシャルバックオフ」を参照してください。
- 再試行とバックオフに AWS SDK 機能を使用しない場合は、Amazon SQS からメッセージ、タイムアウト、エラーメッセージを受信せずに ReceiveMessage アクションを再試行する前に一時停止 (200 ミリ秒など) してください。 Amazon SQS 同じ結果が得られる `ReceiveMessage` をそれ以降に使用するには、それよりも長い一時停止 (たとえば、400ms) を許可します。

問題ありのメッセージのキャプチャ

処理できないすべてのメッセージをキャプチャし、{CloudWatch メトリクス}の正確さを確保するには、[デッドレターキュー]を設定します。

- Redrive ポリシーは、ソースキューがメッセージの処理の失敗を指定回数繰り返した後に、デッドレターキューにメッセージをリダイレクトします。
- デッドレターキューを使用するとメッセージ数が減少し、ポイズンピルメッセージ (受信されたが処理できないメッセージ) が発生する可能性が低下します。
- キューにポイズンピルメッセージを含めると、ポイズンピルメッセージの期間が正しく表示されないため、`ApproximateAgeOfOldestMessage`CloudWatchメトリクスが歪む可能性があります。デッドレターキューを設定すると、このメトリクスを使用する場合の誤ったアラームの回避に役立ちます。

Amazon SQS でのデッドレターキュー保持の設定

標準キューの場合、メッセージの有効期限は常に元のエンキューのタイムスタンプに基づきます。デッドレターキューに移動すると、エンキューのタイムスタンプは変更されません。`ApproximateAgeOfOldestMessage` メトリクスは、メッセージが最初に送信されたときではなく、メッセージがデッドレターキューに移動したときを示します。たとえば、メッセージがデッドレターキューに移動される前に、元のキューで1日費やすと仮定します。デッドレターキューの保持期間が4日間である場合、メッセージは3日後にデッドレターキューから削除され、`ApproximateAgeOfOldestMessage`は3日間です。したがって、デッドレターキューの保持期間を、元のキューの保持期間よりも長く設定することがベストプラクティスです。

FIFO キューでは、メッセージがデッドレターキューに移動すると、エンキューのタイムスタンプがリセットされます。`ApproximateAgeOfOldestMessage` メトリクスは、メッセージがデッドレターキューに移動した日を示します。上記の同じ例では、メッセージは 4 日後にデッドレターキューから削除され、`ApproximateAgeOfOldestMessage` は 4 日間です。

Amazon SQS のメッセージ重複排除とグループ化

このトピックでは、Amazon SQS で一貫したメッセージ処理を確保するためのベストプラクティスについて説明します。ここでは、以下を使用する方法について説明します。

- `MessageDeduplicationId` FIFO キューでのメッセージの重複を防ぎます。
- `MessageGroupId` は、個別のメッセージグループ内のメッセージの順序を管理します。

トピック

- Amazon SQS での一貫性のないメッセージ処理の回避
- Amazon SQS でのメッセージ重複排除 ID の使用
- Amazon SQS FIFO キューでのメッセージグループ ID の使用
- Amazon SQS受信リクエスト試行 ID の使用

Amazon SQS での一貫性のないメッセージ処理の回避

Amazon SQS は分散システムであるため、`ReceiveMessage` API メソッド呼び出しから正常に戻るときにAmazon SQS がメッセージを配信済みとマークしても、コンシューマーはメッセージを受信しない可能性があります。この場合、Amazon SQSはコンシューマーがメッセージを受信していない場合でも、少なくとも 1 回はメッセージを配信済みとして記録します。これらの状況ではメッセージの配信は再試行されないため、デッドレターキューの最大受信数を 1 に設定することはお勧めしません。

Amazon SQS でのメッセージ重複排除 ID の使用

`MessageDeduplicationId` は、メッセージの重複配信を防ぐために Amazon SQS FIFO キューでのみ使用されるトークンです。これにより、5 分間の重複排除ウィンドウ内で、同じ重複排除 ID を持つメッセージのインスタンスが 1 つだけ処理および配信されます。

Amazon SQS が特定の重複排除 ID を持つメッセージを既に受け入れている場合、同じ ID を持つ後続のメッセージは確認されますが、コンシューマーには配信されません。

> **Note**
>
> Amazon SQS は、メッセージの受信と削除後も重複排除 ID の追跡を続けます。

トピック

- Amazon SQS でメッセージ重複排除 ID を指定するタイミング
- Amazon SQS でのシングルプロデューサー/コンシューマーシステムにおける重複排除の有効化
- Amazon SQS での停止復旧シナリオ
- Amazon SQS での可視性タイムアウトの設定

Amazon SQS でメッセージ重複排除 ID を指定するタイミング

プロデューサーは、次のシナリオでメッセージ重複排除 ID を指定する必要があります。

- 一意として扱う必要がある同一のメッセージ本文を送信する場合。
- 同じコンテンツで異なるメッセージ属性を持つメッセージを送信する場合は、各メッセージが個別に処理されるようにします。
- 異なるコンテンツ (メッセージ本文の再試行カウンターなど) でメッセージを送信するが、Amazon SQS に重複として認識させる場合。

Amazon SQS でのシングルプロデューサー/コンシューマーシステムにおける重複排除の有効化

単一のプロデューサーと単一のコンシューマーがあり、本文にアプリケーション固有のメッセージ ID が含まれているためにメッセージが一意である場合は、次のベストプラクティスに従ってください。

- キューでコンテンツベースの重複排除を有効にします (メッセージ本文がそれぞれ一意)。プロデューサーはメッセージ重複排除 ID を省略できます。
- Amazon SQS の FIFO キュー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、メッセージを重複排除 ID 付きで送信すると、`SendMessage` 重複排除 ID により、生成済みのコンテンツベースの重複排除 ID が上書きされます。
- コンシューマーでは各リクエストに対する受信リクエスト試行 ID は必須ではありませんが、失敗再試行シーケンスの実行が速くなるため、これがベストプラクティスです。
- FIFOキュー内のメッセージの順序が妨げられることはないのでリクエストの送信や受信は再試行できます。

Amazon SQS での停止復旧シナリオ

FIFOキューの重複排除プロセスでは、時間的制約があります。アプリケーションを設計するときは、重複や処理の失敗を発生させることなく、プロデューサーとコンシューマーの両方がクライアントまたはネットワークの停止から回復できることを確認してください。

プロデューサーに関する考慮事項

- Amazon SQS は 5 分間の重複排除ウィンドウを適用します。

- プロデューサーが 5 分後に`SendMessage`リクエストを再試行すると、Amazon SQS はリクエストを新しいメッセージとして扱い、重複が生じる可能性があります。

コンシューマーの考慮事項

- 可視性タイムアウトが期限切れになる前にコンシューマーがメッセージを処理できなかった場合、別のコンシューマーがメッセージを受信して処理し、処理が重複する可能性があります。
- アプリケーションの処理時間に基づいて可視性タイムアウトを調整します。
- `ChangeMessageVisibility` API を使用して、メッセージの処理中にタイムアウトを延長します。
- メッセージを繰り返し処理できない場合は、無期限に再処理するのではなく、デッドレターキュー (DLQ) にルーティングします。

- プロデューサーは、キューの重複排除間隔を認識する必要があります。Amazon SQS には 5 分の重複排除間隔があります。重複排除間隔の期限後に `SendMessage` リクエストを再試行すると、キューに重複メッセージが作成される場合があります。たとえば、車内からモバイルデバイスで、順番が重要なメッセージを送信するとします。確認を受信する前に車が一定時間モバイル接続を失った場合、モバイル接続が回復した後もう一度リクエストを再試行すると、重複が発生します。
- コンシューマーには、可視性タイムアウトが切れる前にメッセージを処理できなくなるリスクを最小化する可視性タイムアウトが必要です。`ChangeMessageVisibility` アクションをコールして、メッセージが処理されている間に可視性タイムアウトを延長できます。ただし、可視性タイムアウトの期限が切れると、別のコンシューマーがすぐにメッセージの処理を開始するため、1 つのメッセージが複数回処理されてしまいます。このシナリオを回避するには、デッドレターキューを設定します。

Amazon SQS での可視性タイムアウトの設定

信頼性の高いメッセージ処理を行うには、可視性タイムアウトを AWS SDK 読み取りタイムアウトよりも長く設定します。これは、ショートポーリングとロングポーリングの両方で `ReceiveMessage` API を使用する場合に適用されます。可視性タイムアウトを長くすると、元のリクエストが完了する前に他のコンシューマーがメッセージを使用できなくなるため、処理が重複するリスクが軽減されます。

Amazon SQS FIFO キューでのメッセージグループ ID の使用

FIFO (First-In-First-Out) キューでは、 `MessageGroupId`はメッセージを個別のグループに整理する属性です。同じメッセージグループ内のメッセージは常に 1 つずつ厳密な順序で処理されるため、同じグループからの 2 つのメッセージが同時に処理されることはありません。標準キューでは、を使用すると公平キュー`MessageGroupId`が有効になります。 ???厳密な順序付けが必要な場合は、FIFO キューを使用します。

トピック

- Amazon SQS での複数の順序付けされたメッセージグループのインターリーブ
- Amazon SQS のマルチプロデューサー/コンシューマーシステムでの重複処理の防止
- Amazon SQS で同じメッセージグループ ID を持つ大きなメッセージバックログを避ける
- Amazon SQS で仮想キューでの同じメッセージグループ ID の再利用を避ける

Amazon SQS での複数の順序付けされたメッセージグループのインターリーブ

1 つの FIFO キュー内で複数の順序付けられたメッセージグループをインターリーブするには、各グループに `MessageGroupId`を割り当てます (異なるユーザーのセッションデータなど) 。これにより、同じグループ内のメッセージが順番に処理されると同時に、複数のコンシューマーがキューから同時に読み取ることができます。

特定の のメッセージが処理`MessageGroupId`され、表示されない場合、可視性タイムアウトが期限切れになるか、メッセージが削除されるまで、他のコンシューマーは同じグループからのメッセージを処理できません。

Amazon SQS のマルチプロデューサー/コンシューマーシステムでの重複処理の防止

メッセージの順序付けが優先されない高スループットで低レイテンシーのシステムでは、プロデューサーは各メッセージ`MessageGroupId`に一意の を割り当てることができます。これにより、Amazon SQS FIFO キューは、マルチプロデューサー/マルチコンシューマーのセットアップでも重複を排除できます。このアプローチは重複するメッセージを防止しますが、各メッセージは独自の独立したグループとして扱われるため、メッセージの順序付けは保証されません。

複数のプロデューサーとコンシューマーを持つシステムでは、常に配信が重複するリスクがあります。可視性タイムアウトの有効期限が切れる前にコンシューマーがメッセージを処理できなかった場合、Amazon SQS はメッセージを再度利用可能にし、別のコンシューマーがメッセージを取得できる可能性があります。これを軽減するには、処理時間に基づいて適切なメッセージ確認と可視性タイムアウト設定を確認します。

Amazon SQS で同じメッセージグループ ID を持つ大きなメッセージバックログを避ける

FIFO キューは、最大 120,000 件のインフライトメッセージ (コンシューマーが受信したがまだ削除していないメッセージ) をサポートします。この制限に達すると、Amazon SQS はエラーを返しませんが、処理が影響を受ける可能性があります。AWS サポートに連絡して、この制限を超える引き上げをリクエストできます。

FIFO キューは、最初の 120,000 件のメッセージをスキャンして、使用可能なメッセージグループを決定します。大きなバックログが 1 つのメッセージグループに蓄積された場合、後で送信される他のグループのメッセージは、バックログが処理されるまでブロックされたままになります。

> **Note**
>
> コンシューマーがメッセージの処理に繰り返し失敗すると、メッセージバックログが発生する可能性があります。これは、メッセージコンテンツの問題またはコンシューマー側の障害が原因である可能性があります。メッセージ処理の遅延を防ぐには、試行が複数回失敗した後に未処理のメッセージを移動するようにデッドレターキューを設定します。これにより、同じメッセージグループ内の他のメッセージを処理できるため、システムのボトルネックを防ぐことができます。

Amazon SQS で仮想キューでの同じメッセージグループ ID の再利用を避ける

共有ホストキューで仮想キューを使用する場合は、異なる仮想キュー`MessageGroupId`間で同じ を再利用しないでください。複数の仮想キューが同じホストキューを共有し、同じ を持つメッセージが含まれている場合`MessageGroupId`、それらのメッセージは相互にブロックされ、効率的な処理が妨げられる可能性があります。メッセージ処理をスムーズにするには、異なる仮想キュー内のメッセージに一意の`MessageGroupId`値を割り当てます。

Amazon SQS受信リクエスト試行 ID の使用

受信リクエスト試行 ID は、Amazon SQS での`ReceiveMessage`呼び出しの重複排除に使用される一意のトークンです。アプリケーションと Amazon SQS 間のネットワーク停止や接続の問題が発生した場合は、次のことがベストプラクティスです。

- `ReceiveMessage` 通話を行うときに、受信リクエストの試行 ID を指定します。
- オペレーションが失敗した場合、同じ受信リクエスト試行 ID を使用して再試行します。

Amazon SQS のメッセージ処理とタイミング

このトピックでは、Amazon SQS でのメッセージ処理の速度と効率を最適化するための包括的なガイダンスを提供します。これには、タイムリーなメッセージ処理、最適なポーリングモードの選択、パフォーマンスを向上させるためのロングポーリングの設定などが含まれます。

トピック

- Amazon SQS でのタイムリーな方法でのメッセージ処理
- Amazon SQS でのロングポーリングの設定
- Amazon SQS での適切なポーリングモードの使用

Amazon SQS でのタイムリーな方法でのメッセージ処理

可視性タイムアウトの設定は、アプリケーションがメッセージを処理し、削除するのにかかる時間によって異なります。たとえば、アプリケーションでメッセージを処理するには 10 秒必要だが、可視性タイムアウトを 15 分に設定した場合、前のメッセージ処理に失敗すると、再度処理されるまでに長時間待つ必要があります。または、アプリケーションでメッセージを処理するには 10 秒必要だが、可視性タイムアウトを 2 秒に設定した場合は、元のコンシューマーがメッセージを処理している間に別のコンシューマーより重複メッセージが送信されます。

メッセージの処理に十分な時間を確保するために、次のいずれかの方法を使用します:

- メッセージの処理にかかる時間がわかっている (または合理的に見積もることができる) 場合は、メッセージの可視性タイムアウトを、メッセージの処理と削除にかかる最大時間に拡張します。詳細については、{可視性タイムアウトの構成 } を参照してください。
- メッセージの処理にかかる時間がわからない場合は、コンシューマプロセスについてハートビートを作成:初期可視性タイムアウト (2 分など) を特定し、コンシューマがメッセージで作業している限り、可視性タイムアウトを 1 分ごとに 2 分延長します。

> **Important**
>
> 最大可視性タイムアウトは、`ReceiveMessage`Amazon SQS がメッセージを受信してから 12 時間です。可視性タイムアウトを延長しても、12 時間の上限はリセットされません。
>
> さらに、`ReceiveMessage` リクエストによってタイマーが開始されるため、個々のメッセージのタイムアウトは上限の 12 時間 (43,200 秒) に設定できない場合があります。例えば、メッセージを受信して、すぐに `VisibilityTimeout` を 43,200 秒に等しく設定し

て ChangeMessageVisibility コールを送信すると、失敗する場合があります。ただし、値として 43,195 秒を使用すると、ReceiveMessage 経由でメッセージをリクエストしてから可視性タイムアウトを更新するまでに大幅な遅延がない限り、成功します。コンシューマーが 12 時間以上必要とする場合は、Step Functionsの使用を検討してください。

Amazon SQS でのロングポーリングの設定

待ち時間がいつになるかReceiveMessageAPI アクションが0より大きい場合ロングポーリングが有効です。ロングポーリングの最大待機時間は 20 秒です。ロングポーリングは、空のレスポンス (ReceiveMessage リクエストに対して利用可能なメッセージがない場合) や偽の空のレスポンス (メッセージが利用可能でもレスポンスに含まれない場合) の数を減らすことにより、Amazon SQS の使用コストの削減に役立ちます。詳細については、「Amazon SQSショートポーリングとロングポーリング」を参照してください。

最適なメッセージ処理を行うには、次の方法を使用します。

- ほとんどの場合、ReceiveMessage 待機時間を 20 秒に設定します。アプリケーションの設定時間として 20 秒が長すぎる場合は、ReceiveMessage 待機時間の値を小さくします (最小 1 秒)。AWS SDK を使用して Amazon SQS にアクセスしない場合、または AWS SDK に短い待機時間を設定する場合は、長いリクエストを許可するか、長いポーリングに短い待機時間を使用するように Amazon SQS クライアントを変更する必要があります。
- 複数のキューにロングポーリングを実行する場合は、すべてのキューに単一スレッドを使用せずに、キューごとに 1 つのスレッドを使用します。キューごとに 1 つのスレッドを使用した場合は、メッセージが使用可能になると、アプリケーションはキューごとにメッセージを処理できるのに対し、複数のキューを単一スレッドでポーリングすると、使用可能なメッセージがないキューを待機 (最大 20 秒) している間、アプリケーションは他のキューで使用可能なメッセージを処理できません。

> **Important**
>
> HTTP エラーを回避するには、ReceiveMessage リクエストの HTTP レスポンスタイムアウトが WaitTimeSeconds パラメータよりも長いことを確認してください。詳細については、「ReceiveMessage」を参照してください。

Amazon SQS での適切なポーリングモードの使用

- ロングポーリングにより、Amazon SQSキューからメッセージが利用可能になるとすぐに処理することができます。
  - Amazon SQS の使用でコストを削減し、空のキューでの空の受信数 (メッセージを返さない `ReceiveMessage` アクションへのレスポンス) を減らすには、ロングポーリングを有効にします。詳細については、「Amazon SQS ロングポーリング」を参照してください。
  - 複数の受信により、複数のスレッドをポーリングする際の効率を高めるには、スレッド数を減らします。
  - ロングポーリングは、ほとんどの場合にショートポーリングよりも推奨されます。
- ショートポーリングは、ポーリングされたAmazon SQSキューが空の場合でも、すぐにレスポンスを返します。
  - `ReceiveMessage` リクエストへの即時のレスポンスが期待されるアプリケーションの要件を満たすには、ショートポーリングを使用します。
  - ショートポーリングはロングポーリングと同じように請求されます。

Amazon SQS Java SDKの例

AWS SDK for Java を使用すると、Amazon SQS やその他の とやり取りする Java アプリケーションを構築できます AWS のサービス。

- SDK をインストールしてセットアップする開始方法のAWS SDK for Java 2.x デベロッパーガイドを参照してください。
- キューの作成やメッセージの送信などの基本的なキュー操作については、「 AWS SDK for Java 2.x デベロッパーガイド」のAmazon SQS メッセージキューの使用」を参照してください。
- このガイドには、次のような追加の Amazon SQS 機能の例も含まれています。
  - Amazon SQS キューにおけるサーバー側の暗号化の使用
  - Amazon SQS キューのタグの設定
  - Amazon SQS キューへのメッセージ属性の送信

Amazon SQS キューにおけるサーバー側の暗号化の使用

を使用して AWS SDK for Java 、Amazon SQS キューにサーバー側の暗号化 (SSE) を追加します。各キューは AWS Key Management Service (AWS KMS) KMS キーを使用してデータ暗号化キーを生成します。この例では AWS Amazon SQS のマネージド KMS キーを使用します。

SSE および KMS キーのロールの使用における詳しい情報については、「Amazon SQS での保管中の暗号化」をご参照ください。

既存のキューに SSE を追加

既存のキューのサーバーサイドの暗号化を有効にするには、`SetQueueAttributes`メソッドを使用して`KmsMasterKeyId`属性を設定します。

次のコード例では、 を Amazon SQS の AWS マネージド KMS キー AWS KMS key として設定します。また、この例では、AWS KMS key 再利用期間を 140 秒に設定します。

サンプルコードを実行する前に、 AWS 認証情報が設定されていることを確認してください。詳細については、「 AWS SDK for Java 2.x 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。

```
    public static void addEncryption(String queueName, String kmsMasterKeyAlias) {
```

```
        SqsClient sqsClient = SqsClient.create();

        GetQueueUrlRequest urlRequest = GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build();

        GetQueueUrlResponse getQueueUrlResponse;
        try {
            getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest);
        } catch (QueueDoesNotExistException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        }
        String queueUrl = getQueueUrlResponse.queueUrl();


        Map<QueueAttributeName, String> attributes = Map.of(
                QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias,
                QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140" // Set the
data key reuse period to 140 seconds.
        );                                                                  // This is
how long SQS can reuse the data key before requesting a new one from KMS.

        SetQueueAttributesRequest attRequest = SetQueueAttributesRequest.builder()
                .queueUrl(queueUrl)
                .attributes(attributes)
                .build();
        try {
            sqsClient.setQueueAttributes(attRequest);
            LOGGER.info("The attributes have been applied to {}", queueName);
        } catch (InvalidAttributeNameException | InvalidAttributeValueException e) {
            LOGGER.error(e.getMessage(), e);
            throw new RuntimeException(e);
        } finally {
            sqsClient.close();
        }
    }
```

キューのSSEの無効化

既存のキューに対してサーバーサイドの暗号化を無効にするには、SetQueueAttributes メソッドを使用して、KmsMasterKeyId属性を空の文字列に設定します。

> **Important**
>
> `null` は、の有効な値ではありません。`KmsMasterKeyId`

SSEを使用してキューを作成する

キューの作成時にSSEを有効にするには、API メソッド`KmsMasterKeyId`に属性`CreateQueue`を追加します。

以下の例では、SSE を有効にして新しいキューを作成する方法を示します。このキューは、Amazon SQS の AWS マネージド KMS キーを使用します。また、この例では、AWS KMS key 再利用期間を 160 秒に設定します。

サンプルコードを実行する前に、 AWS 認証情報が設定されていることを確認してください。詳細については、「 AWS SDK for Java 2.x 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。

```
// Create an SqsClient for the specified Region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Create a hashmap for the attributes. Add the key alias and reuse period to the
 hashmap.
HashMap<QueueAttributeName, String> attributes = new HashMap<QueueAttributeName,
 String>();
final String kmsMasterKeyAlias = "alias/aws/sqs";  // the alias of the AWS managed KMS
 key for Amazon SQS.
attributes.put(QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias);
attributes.put(QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140");

// Add the attributes to the CreateQueueRequest.
CreateQueueRequest createQueueRequest =
                CreateQueueRequest.builder()
                        .queueName(queueName)
                        .attributes(attributes)
                        .build();
sqsClient.createQueue(createQueueRequest);
```

SSE属性の取得

キュー属性の取得の詳細については、Amazon Simple キューサービス API リファレンスの例をご参照ください。

特定のキューの KMS キー ID またはデータキーの再利用期間を取得する場合、GetQueueAttributes メソッドを実行して KmsMasterKeyId および KmsDataKeyReusePeriodSeconds 値を取得します。

Amazon SQS キューのタグの設定

コスト配分タグを使用して、Amazon SQSキューの整理および識別するようにします。 AWS SDK for Javaを使用してタグを設定する方法を以下の例に示します。詳細については、「Amazon SQSコスト配分タグ」を参照してください。

サンプルコードを実行する前に、 AWS 認証情報が設定されていることを確認してください。詳細については、「 AWS SDK for Java 2.x 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。

タグを一覧表にする

キューのタグを一覧表示するには、ListQueueTagsの方法を使用します。

```
// Create an SqsClient for the specified region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Get the queue URL.
String queueName = "MyStandardQ1";
GetQueueUrlResponse getQueueUrlResponse =

 sqsClient.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
String queueUrl = getQueueUrlResponse.queueUrl();

// Create the ListQueueTagsRequest.
final ListQueueTagsRequest listQueueTagsRequest =

 ListQueueTagsRequest.builder().queueUrl(queueUrl).build();

// Retrieve the list of queue tags and print them.
final ListQueueTagsResponse listQueueTagsResponse =
                                    sqsClient.listQueueTags(listQueueTagsRequest);
```

```
System.out.println(String.format("ListQueueTags: \tTags for queue %s are %s.\n",
                queueName, listQueueTagsResponse.tags() ));
```

タグの追加または更新するには

キューのタグ値を追加または更新するには、TagQueueの方法を使用します。

```
 // Create an SqsClient for the specified Region.
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Get the queue URL.
String queueName = "MyStandardQ1";
GetQueueUrlResponse getQueueUrlResponse =

 sqsClient.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
String queueUrl = getQueueUrlResponse.queueUrl();

// Build a hashmap of the tags.
final HashMap<String, String> addedTags = new HashMap<>();
        addedTags.put("Team", "Development");
        addedTags.put("Priority", "Beta");
        addedTags.put("Accounting ID", "456def");

//Create the TagQueueRequest and add them to the queue.
final TagQueueRequest tagQueueRequest = TagQueueRequest.builder()
        .queueUrl(queueUrl)
        .tags(addedTags)
        .build();
sqsClient.tagQueue(tagQueueRequest);
```

タグの削除

キューから1つ以上のタグを削除するには、UntagQueueの方法を使用します。次の例では、Accounting ID タグを 削除しています。

```

// Create the UntagQueueRequest.
final UntagQueueRequest untagQueueRequest = UntagQueueRequest.builder()
        .queueUrl(queueUrl)
        .tagKeys("Accounting ID")
```

```
        .build();

// Remove the tag from this queue.
sqsClient.untagQueue(untagQueueRequest);
```

Amazon SQS キューへのメッセージ属性の送信

メッセージ属性をメッセージに使用すると、構造化メタデータ (タイムスタンプ、地理空間データ、署名、識別子) などを指定することができます。詳細については、「Amazon SQSメッセージ属性」を参照してください。

サンプルコードを実行する前に、 AWS 認証情報が設定されていることを確認してください。詳細については、「 AWS SDK for Java 2.x 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。

属性の定義

メッセージの属性を定義するには、`MessageAttributeValue` データタイプを使用する以下のコードを追加します。詳細については、「メッセージ属性コンポーネント」および「メッセージ属性のデータ型」を参照してください。

はメッセージ本文とメッセージ属性のチェックサム AWS SDK for Java を自動的に計算し、Amazon SQS が返すデータと比較します。詳細については、AWS SDK for Java 2.x デベロッパーガイドを参照してください。他のプログラミング言語については、「メッセージ属性のMD5メッセージダイジェストの計算」を参照してください。

String

この例では、値が`String`である、`Name`という名前の`Jane`属性を定義します。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("Name", new MessageAttributeValue()
.withDataType("String")
.withStringValue("Jane"));
```

Number

この例では、値が`Number`である、`AccurateWeight`という名前の`230.0000000000000000001`属性を定義します。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccurateWeight", new MessageAttributeValue()
.withDataType("Number")
.withStringValue("230.000000000000000001"));
```

Binary

この例では、初期化されていない10バイト配列の値を持つ、Binaryという名前のByteArray属性を定義します。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("ByteArray", new MessageAttributeValue()
.withDataType("Binary")
.withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

String (custom)

この例では、値がString.EmployeeIdである、EmployeeIdという名前のカスタム属性ABC123456を定義します。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("EmployeeId", new MessageAttributeValue()
.withDataType("String.EmployeeId")
.withStringValue("ABC123456"));
```

Number (custom)

この例では、値がNumber.AccountIdである、AccountIdという名前のカスタム属性000123456を定義します。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccountId", new MessageAttributeValue()
.withDataType("Number.AccountId")
.withStringValue("000123456"));
```

> **Note**
>
> ベースデータタイプはNumberであるため、ReceiveMessageメソッドは123456を返します。

Binary (custom)

この例では、初期化されていない10バイト配列の値を持つ、Binary.JPEGという名前のカスタム属性ApplicationIconを定義します。

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("ApplicationIcon", new MessageAttributeValue()
.withDataType("Binary.JPEG")
.withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

属性を含むメッセージの送信

この例では、SendMessageRequestメッセージを送信する前に属性を追加します。

```
// Send a message with an attribute.
final SendMessageRequest sendMessageRequest = new SendMessageRequest();
sendMessageRequest.withMessageBody("This is my message text.");
sendMessageRequest.withQueueUrl(myQueueUrl);
sendMessageRequest.withMessageAttributes(messageAttributes);
sqs.sendMessage(sendMessageRequest);
```

Important

First-In-First-Out(FIFO)キューにメッセージを送信する場合は、メッセージグループIDを提供したsendMessage後で、 メソッドが実行されることを確認します。SendMessageの代わりにSendMessageBatch を使用する場合は、バッチの各メッセージに対してメッセージ属性を指定する必要があります。

Amazon SQS での API の使用

このトピックでは、Amazon SQS エンドポイントの構築、GET メソッドと POST メソッドを使用したクエリ API リクエストの実行、バッチ API アクションの使用について説明します。パラメータ、エラー、例、およびデータ型を含む Amazon SQS アクションの詳細については、「Amazon Simple Queue Service API リファレンス」を参照してください。

さまざまなプログラミング言語を使用して Amazon SQS にアクセスするには、以下の自動機能が含まれた AWS SDK を使用することもできます。

- サービスリクエストに暗号署名する
- リクエストを再試行する
- エラーレスポンスの処理をする

詳細については、「the section called “ AWS SDKs の使用”」を参照してください。

コマンドラインツール情報については、「AWS CLI コマンドリファレンス」の「Amazon SQS セクション」、および「AWS Tools for PowerShell Cmdlet リファレンス」を参照してください。

JSON プロトコルを使用した Amazon SQS APIs AWS

Amazon SQS は、指定された SDK AWS バージョン上のすべての Amazon SQS APIs のトランスポートメカニズムとして JSON プロトコルを使用します。 AWS JSON プロトコルは、スループットが高く、レイテンシーが低く、application-to-application通信が高速化されます。 AWS JSON プロトコルは、 AWS クエリプロトコルと比較して、リクエストとレスポンスのシリアル化/逆シリアル化がより効率的です。 AWSSQS API で AWS クエリプロトコルを使用する場合、Amazon SQS AWS クエリプロトコルをサポートする AWS SDK バージョンAmazon SQS API で使用される AWS JSON プロトコルではどの言語がサポートされていますか。については、「」を参照してください。 APIs

Amazon SQS は AWS 、JSON プロトコルを使用して AWS SDK クライアント (Java、Python、Golang、JavaScript など) と Amazon SQS サーバー間の通信を行います。Amazon SQS API オペレーションの HTTP リクエストは、JSON 形式の入力を受け付けます。Amazon SQS オペレーションが実行され、実行レスポンスが JSON 形式で SDK クライアントに送り返されます。AWS クエリと比較して、 AWS JSON はクライアントとサーバー間でデータを転送するのがより簡単、迅速、効率的です。

- AWS JSON プロトコルは、Amazon SQS クライアントとサーバー間のメディエーターとして機能します。

- サーバーは Amazon SQS オペレーションが作成されるプログラミング言語を理解していませんが、JSON AWS プロトコルを理解しています。
- AWS JSON プロトコルは、Amazon SQS クライアントとサーバー間のシリアル化 (オブジェクトを JSON 形式に変換) とシリアル化解除 (JSON 形式をオブジェクトに変換) を使用します。

Amazon SQS での AWS JSON プロトコルの詳細については、「」を参照してくださいAmazon SQS AWS JSON プロトコルFAQs。

AWS JSON プロトコルは、指定された AWS SDK バージョンで使用できます。さまざまな言語バージョンの SDK バージョンとリリース日を確認するには、「AWS SDK およびツールリファレンスガイド」の「AWS SDK とツールのバージョンサポートマトリックス」を参照してください。

Amazon SQS での AWS JSON プロトコルを使用したクエリ API リクエストの実行 Amazon SQS

このトピックでは、Amazon SQS エンドポイントの構築、POST リクエストの作成、レスポンスの解釈方法について説明します。

> **Note**
>
> AWS JSON プロトコルは、ほとんどの言語バリアントでサポートされています。サポートされている言語のリストについては、「Amazon SQS API で使用される AWS JSON プロトコルではどの言語がサポートされていますか。」を参照してください。

エンドポイントの構築

Amazon SQS キューを使用するには、エンドポイントを構築する必要があります。Amazon SQS エンドポイントの詳細については、「Amazon Web Services 全般のリファレンス」の以下のページを参照してください。

- リージョンエンドポイント
- Amazon Simple Queue Serviceエンドポイントとクォータ

Amazon SQSエンドポイントはそれぞれ独立しています。例えば、2 つのキューの名前が MyQueue で、一方にはエンドポイント `sqs.us-east-2.amazonaws.com` があり、もう一方にはエンドポイ

ント sqs.eu-west-2.amazonaws.com がある場合、2 つのキューは互いにどのデータも共有しません。

キュー作成のクエストを行うエンドポイントの例を次に示します。

```
POST / HTTP/1.1
Host: sqs.us-west-2.amazonaws.com
X-Amz-Target: AmazonSQS.CreateQueue
X-Amz-Date: <Date>
Content-Type: application/x-amz-json-1.0
Authorization: <AuthParams>
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
{
    "QueueName":"MyQueue",
    "Attributes": {
        "VisibilityTimeout": "40"
    },
    "tags": {
        "QueueType": "Production"
    }
}
```

> **Note**
>
> キュー名とキュー URL では、大文字と小文字が区別されます。
> *AUTHPARAMS*の構造はAPIリクエストの署名によって異なります。詳細については、Amazon Web Services 全般のリファレンスの AWS API リクエストの署名を参照してください。

POST リクエストの作成

Amazon SQS の POST リクエストは、クエリパラメータを HTTP リクエストボディの形で送信します。

以下は、X-Amz-Target が AmazonSQS.<operationName> に設定された HTTP ヘッダーと、Content-Type が application/x-amz-json-1.0 に設定された HTTP ヘッダーの例です。

```
POST / HTTP/1.1
Host: sqs.<region>.<domain>
X-Amz-Target: AmazonSQS.SendMessage
```

```
X-Amz-Date: <Date>
Content-Type: application/x-amz-json-1.0
Authorization: <AuthParams>
Content-Length: <PayloadSizeBytes>
Connection: Keep-Alive
{
    "QueueUrl": "https://sqs.<region>.<domain>/<awsAccountId>/<queueName>/",
    "MessageBody": "This is a test message"
}
```

この HTTP POST リクエストは、Amazon SQS キューにメッセージを送信します。

> **Note**
>
> HTTP ヘッダー `X-Amz-Target` と `Content-Type` は両方とも必須です。
> HTTPクライアントは、クライアントのHTTP バージョンによっては、他の項目を HTTPリクエストに追加する可能性があります。

Amazon SQS JSON API レスポンスの解釈

Amazon SQS にリクエストを送信すると、結果を含む JSON レスポンスが返されます。レスポンス構造は、使用した API アクションによって異なります。

これらのレスポンスの詳細については、以下を参照してください。

- Amazon Simple Queue Service API リファレンスの API アクションの特定の API アクション
- Amazon SQS AWS JSON プロトコルFAQs

正常な JSON レスポンス構造

リクエストが成功した場合、主なレスポンス要素は `x-amzn-RequestId` です。これには、リクエストの汎用一意識別子 (UUID) と、その他の付加されたレスポンスフィールドが含まれます。例えば、`CreateQueue` レスポンスには `QueueUrl` フィールドが含まれています。そのフィールドには、作成されたキューの URL が含まれています。

```
HTTP/1.1 200 OK
x-amzn-RequestId: <requestId>
Content-Length: <PayloadSizeBytes>
Date: <Date>
```

```
Content-Type: application/x-amz-json-1.0
{
    "QueueUrl":"https://sqs.us-east-1.amazonaws.com/111122223333/MyQueue"
}
```

JSON エラーレスポンス構造

リクエストが正常に行われなかった場合、Amazon SQS は HTTP ヘッダーとボディを含む主なレスポンスを返信します。

HTTP ヘッダーで、`x-amzn-RequestId` にはリクエストの UUID が含まれています。`x-amzn-query-error` には、エラーのタイプと、エラーがプロデューサーエラーかコンシューマーエラーかという 2 つの情報が含まれています。

レスポンスボディで、`"__type"` はその他のエラーの詳細を示し、`Message` は読みやすい形式でエラー状況を示します。

以下は、JSON 形式のエラーレスポンスの例です。

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: 66916324-67ca-54bb-a410-3f567a7a0571
x-amzn-query-error: AWS.SimpleQueueService.NonExistentQueue;Sender
Content-Length: <PayloadSizeBytes>
Date: <Date>
Content-Type: application/x-amz-json-1.0
{
    "__type": "com.amazonaws.sqs#QueueDoesNotExist",
    "message": "The specified queue does not exist."
}
```

Amazon SQS AWS JSON プロトコルFAQs

このトピックでは、Amazon SQS での AWS JSON プロトコルの使用に関するよくある質問について説明します。

AWS JSON プロトコルとは何ですか。また、既存の Amazon SQS API リクエストとレスポンスとどのように異なりますか？

JSON は、異種システム間の通信で最も広く使用され、受け入れられている配線方法の 1 つです。Amazon SQS は JSON を通信媒体として使用して、 AWS SDK クライアント (Java、Python、Golang、JavaScript など) と Amazon SQS サーバー間の通信を行います。Amazon

SQS API オペレーションの HTTP リクエストは、JSON 形式の入力を受け付けます。Amazon SQS オペレーションが実行され、実行レスポンスが JSON 形式で SDK クライアントに共有されます。AWS クエリと比較して、JSON はクライアントとサーバー間でより効率的にデータ転送できます。

- Amazon SQS AWS JSON プロトコルは、Amazon SQS クライアントとサーバー間のメディエーターとして機能します。
- サーバーは Amazon SQS オペレーションが作成されるプログラミング言語を理解していませんが、JSON AWS プロトコルを理解しています。
- Amazon SQS AWS JSON プロトコルは、Amazon SQS クライアントとサーバー間のシリアル化 (オブジェクトを JSON 形式に変換) と逆シリアル化 (JSON 形式をオブジェクトに変換) を使用します。

Amazon SQS の AWS JSON プロトコルの使用を開始するにはどうすればよいですか？

Amazon SQS のメッセージングを高速化するために最新の AWS SDK バージョンの使用を開始するには、 AWS SDK を指定されたバージョンまたはそれ以降のバージョンにアップグレードします。SDK クライアントの詳細については、以下の表の「ガイド」列を参照してください。

以下は、Amazon SQS API で使用する JSON AWS プロトコルの言語バリアント間の SDK バージョンのリストです。 Amazon SQS APIs

言語	SDK クライアントリポジトリ	必要な SDK クライアントバージョン	ガイド
C++	aws/aws-sdk-cpp	1.11.98	AWS SDK for C++
Golang 1.x	aws/aws-sdk-go	v1.47.7	AWS SDK for Go
Golang 2.x	aws/aws-sdk-go-v2	v1.28.0	AWS SDK for Go V2
Java 1.x	aws/aws-sdk-java	1.12.585	AWS SDK for Java
Java 2.x	aws/aws-sdk-java-v2	2.21.19	AWS SDK for Java

言語	SDK クライアントリポジトリ	必要な SDK クライアントバージョン	ガイド
JavaScript v2.x	aws/aws-sdk-js	での JavaScript AWS	
JavaScript v3.x	aws/aws-sdk-js-v3	v3.447.0	での JavaScript AWS
.NET	aws/aws-sdk-net	3.7.681.0	AWS SDK for .NET
PHP	aws/aws-sdk-php	3.285.2	AWS SDK for PHP
Python-boto3	boto/boto3	1.28.82	AWS SDK for Python (Boto3)
Python-botocore	boto/botocore	1.31.82	AWS SDK for Python (Boto3)
awscli	AWS CLI	1.29.82	AWSコマンドラインインターフェイス
Ruby	aws/aws-sdk-ruby	1.67.0	AWS SDK for Ruby

Amazon SQS ワークロードで JSON プロトコルを有効にすることにはどのようなリスクがありますか。

AWS SDK のカスタム実装、またはカスタムクライアントと AWS SDK の組み合わせを使用して、クエリベースの (XML ベースの) AWS レスポンスを生成する Amazon SQS とやり取りしている場合、JSON AWS プロトコルと互換性がない可能性があります。問題が発生した場合は、 AWS サポートにお問い合わせください。

すでに最新の AWS SDK バージョンを使用しているが、オープンソースソリューションが JSON をサポートしていない場合はどうなりますか？

SDK のバージョンを、使用中のバージョンより前のバージョンに変更する必要があります。詳細についてはAmazon SQS の AWS JSON プロトコルの使用を開始するにはどうすればよいですか？、「」を参照してください。「」にリストされている AWS SDK バージョンAmazon SQS の AWS JSON プロトコルの使用を開始するにはどうすればよいですか？は、Amazon SQS API の JSON ワイヤプロトコルを使用します。 APIs AWS SDK を以前のバージョンに変更すると、Amazon SQS APIsは AWS クエリを使用します。

Amazon SQS API で使用される AWS JSON プロトコルではどの言語がサポートされていますか。

Amazon SQS は、 AWS SDKsされているすべての言語バリアント (GA) をサポートしています。現在、Kotlin、Rust、Swift はサポートしていません。他の言語バリアントについて詳しくは、「AWS での構築ツール」を参照してください。

Amazon SQS API で使用される AWS JSON プロトコルではどのリージョンがサポートされていますか。

Amazon SQS は AWS 、Amazon SQS が利用可能なすべてのAWS リージョンで JSON プロトコルをサポートしています。 Amazon SQS

AWS JSON プロトコルを使用して Amazon SQS の指定された AWS SDK バージョンにアップグレードすると、どのようなレイテンシーの改善が期待できますか？

AWS JSON プロトコルは、 AWS クエリプロトコルと比較して、リクエストとレスポンスのシリアル化と逆シリアル化がより効率的です。5 KB のメッセージペイロード AWS のパフォーマンステストに基づいて、Amazon SQS の JSON プロトコルはend-to-endのメッセージ処理レイテンシーを最大 23% 削減し、アプリケーションクライアント側の CPU とメモリの使用量を削減します。

AWS クエリプロトコルは廃止されますか？

AWS クエリプロトコルは引き続きサポートされます。 AWS SDK バージョンが、「Amazon SQS の JSON プロトコルの開始方法」に記載されているもの以外の以前のバージョンに設定されている限り、 AWS クエリプロトコルを引き続き使用できます。 AWS Amazon SQS

AWS JSON プロトコルの詳細情報はどこで入手できますか。

JSON プロトコルの詳細については、Smithy ドキュメントの「AWS JSON 1.0 プロトコル」を参照してください。 AWS JSON プロトコルを使用する Amazon SQS API リクエストの詳細については、「Amazon SQS での AWS JSON プロトコルを使用したクエリ API リクエストの実行 Amazon SQS」を参照してください。

Amazon SQS でのクエリプロトコルを使用した AWS クエリ API リクエストの実行 Amazon SQS

このトピックでは、Amazon SQS エンドポイントの構築、GET および POST リクエストの作成、レスポンスの解釈方法について説明します。

エンドポイントの構築

Amazon SQSキューを使用するには、エンドポイントを構築する必要があります。Amazon SQS エンドポイントの詳細については、「Amazon Web Services 全般のリファレンス」の以下のページを参照してください。

- リージョンエンドポイント
- Amazon Simple Queue Serviceエンドポイントとクォータ

Amazon SQSエンドポイントはそれぞれ独立しています。例えば、2 つのキューの名前が MyQueue で、一方にはエンドポイント `sqs.us-east-2.amazonaws.com` があり、もう一方にはエンドポイント `sqs.eu-west-2.amazonaws.com` がある場合、2 つのキューは互いにどのデータも共有しません。

キューを作成するリクエストを行うエンドポイントの例を次に示します。

```
https://sqs.eu-west-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> キュー名とキュー URL では、大文字と小文字が区別されます。
> *AUTHPARAMS*の構造はAPIリクエストの署名によって異なります。詳細については、Amazon Web Services 全般のリファレンスの AWS API リクエストの署名を参照してください。

GETリクエストの作成

Amazon SQS の GET リクエストは、以下を要素とする URLとして構成します。

- エンドポイント-リクエストが作用するリソース (キュー名と URL)。たとえば: `https://sqs.us-east-2.amazonaws.com/`*`123456789012`*`/MyQueue`
- アクション-エンドポイントに実行したい アクション。疑問符 (?) によってエンドポイントとアクションが区切られます。たとえば`?Action=SendMessage&MessageBody=Your%20Message%20Text`など:
- パラメータ – 任意のリクエストパラメータ。パラメータごとにアンパサンド (&) で区切ります。例: `&Version=2012-11-05&`*`AUTHPARAMS`*

メッセージを Amazon SQS キューに送信する GET リクエストの例を以下に示します。

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
?Action=SendMessage&MessageBody=Your%20message%20text
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> キュー名とキュー URL では、大文字と小文字が区別されます。
> GET リクエストは URL であるため、すべてのパラメータ値を URL エンコードする必要があります。URL にはスペースを使用できないため、各スペースは `%20` として URL エンコードします。例の残りについては、読みやすくするために URL エンコードしていません。

POST リクエストの作成

Amazon SQS の POST リクエストは、クエリパラメータを HTTP リクエストボディの形で送信します。

`Content-Type` を `application/x-www-form-urlencoded` に設定した HTTP ヘッダーの例を次に示します。

```
POST /123456789012/MyQueue HTTP/1.1
Host: sqs.us-east-2.amazonaws.com
Content-Type: application/x-www-form-urlencoded
```

ヘッダーの後には、メッセージを Amazon SQS キューに送信する `form-urlencoded` GET リクエストを続けます。各パラメータは、アンパサンド (&)で区切られています。

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Expires=2020-10-15T12%3A00%3A00Z
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> `Content-Type`HTTPヘッダーのみが必須です。*`AUTHPARAMS`* は、GET リクエストの場合と同じです。
> HTTPクライアントは、クライアントのHTTP バージョンによっては、他の項目を HTTPリクエストに追加する可能性があります。

Amazon SQS の XML API レスポンスの解釈

Amazon SQS にリクエストを送信すると、リクエストの結果を含む XML レスポンスが返されます。これらのレスポンスの構造と詳細を理解するには、Amazon Simple Queue Service API リファレンスの特定の API アクションを参照してください。

成功した XML レスポンス構造

リクエストが成功すると、メインのレスポンス要素は、アクション名の後に `Response` を続けた名前になります (例: *`ActionName`*`Response`)。

この要素には、以下の子要素が含まれています。

- ***ActionName*Result**-アクション固有の要素が含まれています。たとえば、CreateQueueResult 要素には QueueUrl 要素が含まれています。つまり、作成されたキューのURLが含まれています。
- **ResponseMetadata** – RequestId を含み、この ID 内にリクエストのユニバーサルユニーク識別子 (UUID) が含まれています。

以下に、XML形式の正常なレスポンスの例を示します。

```
<CreateQueueResponse
   xmlns=https://sqs.us-east-2.amazonaws.com/doc/2012-11-05/
   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:type=CreateQueueResponse>
   <CreateQueueResult>
      <QueueUrl>https://sqs.us-east-2.amazonaws.com/770098461991/queue2</QueueUrl>
   </CreateQueueResult>
   <ResponseMetadata>
      <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
   </ResponseMetadata>
</CreateQueueResponse>
```

XML エラーレスポンス構造

リクエストが正常に行われなかった場合、 Amazon SQSは常に主な応答要素を返信しますErrorResponse。この要素には Error 要素と RequestId 要素が含まれます。

Error 要素には、以下の子要素が含まれています。

- **Type**-エラーがプロデューサーかコンシューマーのどちらかを特定します。
- **Code**-エラーのタイプを特定します。
- **Message**-読み取り可能な条件でエラー状況を特定します。
- **Detail**-(オプション) エラーに関する追加の詳細を特定します。

RequestId要素には、リクエストのUUIDが含まれています。

以下に、XML形式のエラーレスポンスの例を示します。

```
<ErrorResponse>
```

```
    <Error>
       <Type>Sender</Type>
       <Code>InvalidParameterValue</Code>
       <Message>
          Value (quename_nonalpha) for parameter QueueName is invalid.
          Must be an alphanumeric String of 1 to 80 in length.
       </Message>
    </Error>
    <RequestId>42d59b56-7407-4c4a-be0f-4c88daeea257</RequestId>
</ErrorResponse>
```

Amazon SQS のリクエストの認証

署名認証とは、リクエストの送信者を特定し、検証するプロセスです。認証の最初の段階で、 AWS はプロデューサーのアイデンティティと、プロデューサーが AWSを使用するように登録されているかどうかを確認します (詳細については、「ステップ 1: AWS アカウント と IAM ユーザーを作成する」を参照してください)。次に、以下の手順 AWS に従います。

1. プロデューサー (送信者) が必要な証明書を取得します。
2. プロデューサーがリクエストと認証情報をコンシューマー (受信者) に送信します。
3. コンシューマーは証明書を使用して、プロデューサーが本当にリクエストを送信したか検証します。
4. 次のいずれかの結果になります。
   - 認証が成功すると、コンシューマーはリクエストを処理します。
   - 認証に失敗すると、コンシューマーによってリクエストが却下され、エラーが返されます。

HMAC-SHA による基本的な認証のプロセス

クエリAPIを使用してAmazon SQSにアクセスする場合は、リクエストの認証のために、以下の項目を準備する必要があります。

- を識別するAWS アクセスキー ID。 AWS は AWS アカウントを使用してシークレットアクセスキーを検索します。
- HMAC-SHAリクエスト署名、シークレットアクセスキーを使用して計算されます(共有シークレットはお客様と AWSだけが知っています。詳細については、RFC2104 を参照してください)。AWS SDK によって署名プロセスが処理されますが、HTTP または HTTPS 経由でクエリリクエストを送信した場合、各クエリリクエストに署名を含める必要があります。

1. 署名バージョン4の署名キーを取得します。詳細については、「Java を使用して署名キーを取得」を参照してください。

> **Note**
>
> Amazon SQSは署名バージョン4をサポートしています。署名バージョン4では、SHA256により以前のバージョンよりもセキュリティとパフォーマンスが向上しています。Amazon SQSを使用する新規のアプリケーションを作成する場合、署名バージョン4を使用します。

2. リクエスト署名をBase64 エンコードします。以下のサンプルJavaコードによってこの処理が行われます。

```
package amazon.webservices.common;

// Define common routines for encoding data in AWS requests.
public class Encoding {

    /* Perform base64 encoding of input bytes.
     * rawData is the array of bytes to be encoded.
     * return is the base64-encoded string representation of rawData.
     */
    public static String EncodeBase64(byte[] rawData) {
        return Base64.encodeBytes(rawData);
    }
}
```

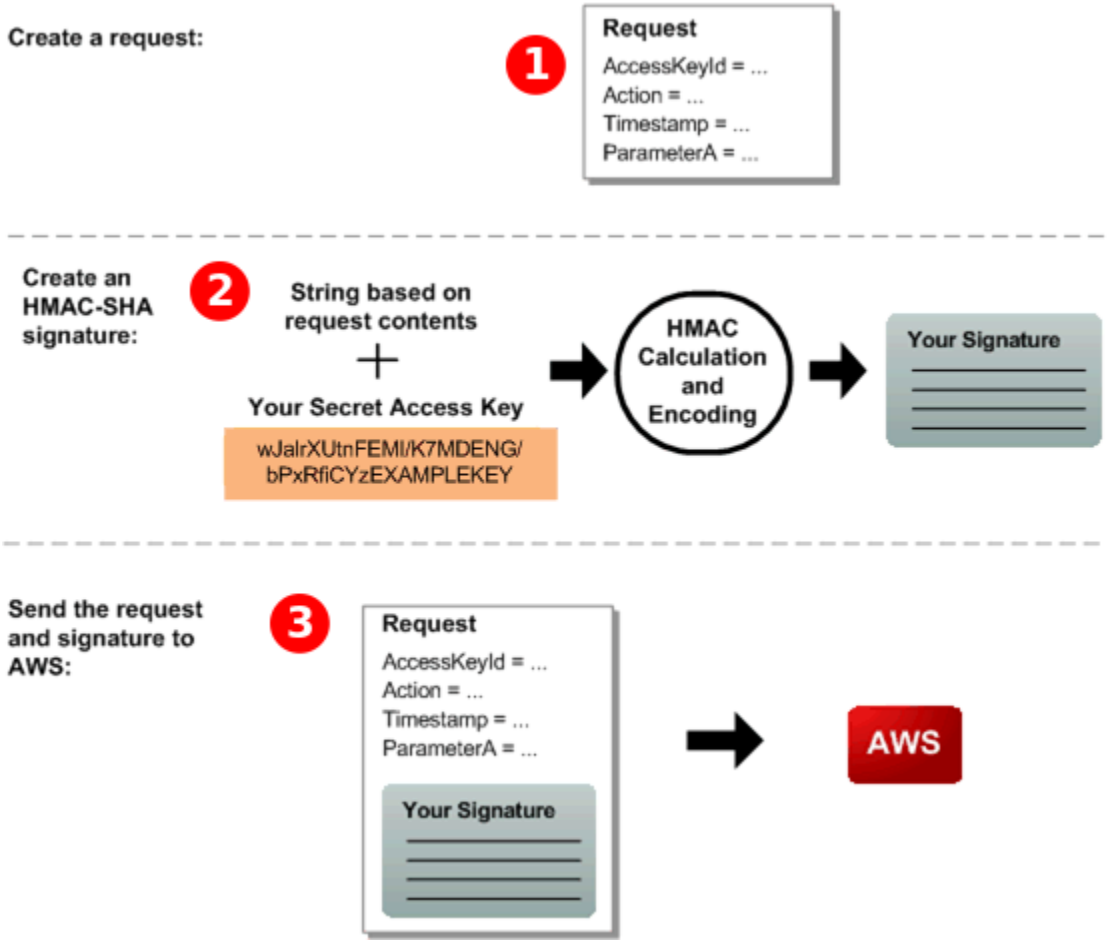
- リクエストのタイムスタンプ (または有効期限)。リクエストで使用するタイムスタンプは、`dateTime` オブジェクトでなければなりません。時、分、秒を含む詳細な日時が必要です。たとえば、`2007-01-31T23:59:59Z` などです。必須ではありませんが、協定世界時 (グリニッジ標準時)を使用してオブジェクトを提供することが推奨されています。

> **Note**
>
> あなたのサーバー時刻が正しく設定されていることを確認してください。 (有効期限ではなく) タイムスタンプを指定すると、リクエストは指定された時刻から 15 分後に自動的に期限切れになります (サーバーの現在の時刻 AWS より 15 AWS 分以上前のタイムスタンプを持つリクエストは処理されません) 。

.NET を使用する場合、過度に厳密なタイムスタンプ情報を送信しないようご注意ください (タイムスタンプの処理プロセスの違いにより不具合が発生する可能性があるためです)。この場合、精度が 1ミリ秒以内のdateTimeオブジェクトを手動で作成してください。

パート 1: ユーザーからのリクエスト

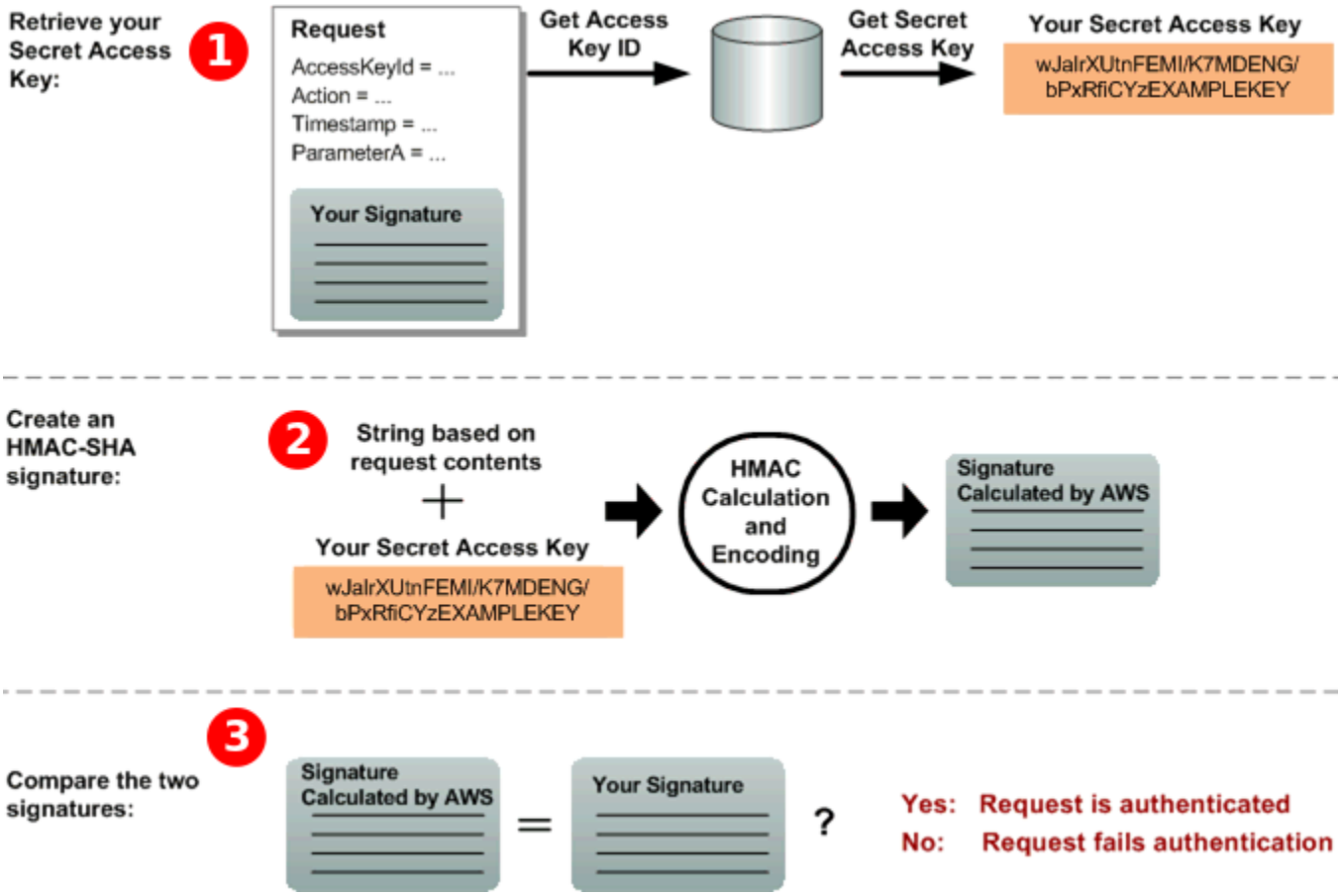
HMAC-SHA AWS リクエスト署名を使用してリクエストを認証するには、次のプロセスに従う必要があります。



1. へのリクエストを作成します AWS。
2. シークレットアクセスキーを使用して、キー付きハッシュメッセージ認証コード (HMAC-SHA) 署名を計算します。
3. リクエストに署名とアクセスキー ID を含めてから、リクエストを送信します AWS。

パート 2: からのレスポンス AWS

AWS は応答として次のプロセスを開始します。



1. AWS はアクセスキー ID を使用してシークレットアクセスキーを検索します。
2. AWS は、リクエストで送信した署名の計算に使用したのと同じアルゴリズムを使用して、リクエストデータとシークレットアクセスキーから署名を生成します。
3. 次のいずれかの結果になります。
   - が AWS 生成する署名がリクエストで送信した署名と一致する場合、 はリクエストが本物である AWS と見なします。
   - 比較が失敗すると、リクエストは破棄され、エラー AWS が返されます。

Amazon SQSのバッチアクション

Amazon SQS のバッチアクションを使用すると、コストを削減し、1 つのアクションで最大 10 件のメッセージを操作できます。バッチアクションは以下のとおりです。

- SendMessageBatch
- DeleteMessageBatch
- ChangeMessageVisibilityBatch

バッチアクションを使用すると、1 回の API コールで複数のオペレーションを実行できるため、パフォーマンスを最適化してコストを削減できます。クエリ API または Amazon SQS バッチアクションをサポートする任意の AWS SDK を使用して、バッチ機能を活用できます。

重要な詳細

- メッセージサイズ制限： 1 回のSendMessageBatch呼び出しで送信されるすべてのメッセージの合計サイズは、1,048,576 バイト (1 MiB) を超えることはできません
- アクセス許可: SendMessageBatch、DeleteMessageBatch、または ChangeMessageVisibilityBatch のアクセス許可を明示的に設定することはできません。SendMessage、DeleteMessage、または ChangeMessageVisibility のアクセス許可を設定すると、代わりに、アクションの対応するバッチバージョンのアクセス許可が設定されます。
- コンソールのサポート: Amazon SQS コンソールは、バッチアクションをサポートしていません。バッチオペレーションを実行するには、クエリ API または AWS SDK を使用する必要があります。

メッセージアクションのバッチ処理

コストと効率をさらに最適化するには、メッセージアクションのバッチ処理に関する以下のベストプラクティスを検討してください。

- バッチ API アクション: 1 つのアクションで複数のメッセージを送信、受信、削除したり、複数のメッセージのメッセージ可視性タイムアウトを変更したりするには、Amazon SQS のバッチ API アクションを使用します。これにより、API コールの数と関連コストを削減できます。
- クライアント側のバッファリングとロングポーリング: ロングポーリングと AWS SDK for Javaに含まれるバッファリング非同期クライアントを併用することで、クライアント側のバッファリングとリクエストバッチ処理を組み合わせます。このアプローチは、リクエスト数を最小限に抑え、大量のメッセージ処理を最適化するのに役立ちます。

> **Note**
>
> Amazon SQSバッファリング非同期クライアントは現在 FIFOキューをサポートしていません。

Amazon SQS でのクライアント側のバッファリングとリクエストのバッチ処理の有効化

AWS SDK for Javaを含む`AmazonSQSBufferedAsyncClient`Amazon SQSにアクセスするもの。このクライアントにより、クライアント側のバッファリングを使用したシンプルなリクエストバッチ処理が可能になります。クライアントから行われた呼び出しは、最初にバッファされ、バッチリクエストとして Amazon SQS に送信されます。

クライアント側のバッファリングは最大10個のリクエストをバッファリングし、バッチリクエストとして送信できるため、Amazon SQSの利用コストを削減して、送信リクエストの数を減らすことができます。バッファリングは同期および非同期コールの両方を`AmazonSQSBufferedAsyncClient` バッファします。バッチ処理されたリクエストとロングポーリングのサポートによって、スループットを向上させることもできます。詳細については、「Amazon SQS での水平スケーリングとアクションのバッチ処理を使用したスループットの向上」を参照してください。

`AmazonSQSBufferedAsyncClient`は`AmazonSQSAsyncClient`と同じインターフェイスを実行するため、`AmazonSQSAsyncClient` から `AmazonSQSBufferedAsyncClient` に移行するには通常既存のコードを最小限変更するだけです。

> **Note**
>
> Amazon SQS バッファリング非同期クライアントは現在 FIFOキューをサポートしていません。

AmazonSQSBufferedAsyncClient の使用

開始する前に、「Amazon SQSのセットアップ」のステップを完了します。

AWS SDK for Java 1.x

AWS SDK for Java 1.x では、次の例AmazonSQSBufferedAsyncClientに基づいて新しい を作成できます。

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

新規作成したらAmazonSQSBufferedAsyncClient、これを使用してAmazon SQSに複数のリクエストを送信できます (AmazonSQSAsyncClient と同様です)。次に例を示します。

```
final CreateQueueRequest createRequest = new
 CreateQueueRequest().withQueueName("MyQueue");

final CreateQueueResult res = bufferedSqs.createQueue(createRequest);

final SendMessageRequest request = new SendMessageRequest();
final String body = "Your message text" + System.currentTimeMillis();
request.setMessageBody( body );
request.setQueueUrl(res.getQueueUrl());

final Future<SendMessageResult> sendResult = bufferedSqs.sendMessageAsync(request);

final ReceiveMessageRequest receiveRq = new ReceiveMessageRequest()
    .withMaxNumberOfMessages(1)
    .withQueueUrl(queueUrl);
final ReceiveMessageResult rx = bufferedSqs.receiveMessage(receiveRq);
```

AmazonSQSBufferedAsyncClient の設定

AmazonSQSBufferedAsyncClient は、ほとんどのユースケースに合うように事前に設定されています。たとえば、AmazonSQSBufferedAsyncClient をさらに設定できます。'

1. 必要な設定パラメータを使用して、QueueBufferConfig クラスのインスタンスを作成します。
2. AmazonSQSBufferedAsyncClient コンストラクタにインスタンスを指定します。

```
// Create the basic Amazon SQS async client
```

```
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

final QueueBufferConfig config = new QueueBufferConfig()
    .withMaxInflightReceiveBatches(5)
    .withMaxDoneReceiveBatches(15);

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync, config);
```

QueueBufferConfig の設定パラメータ

パラメータ	デフォルト値	説明
longPoll	true	longPoll が true に設定されている場合、AmazonSQSBufferedAsyncClient はメッセージの処理時にロングポーリングを使用しようとします。
longPollWaitTimeoutSeconds	20 s	空の受信結果を返すまでに、キュー内へのメッセージの出現をサーバーが待機するのを ReceiveMessage 呼び出しがブロックする最大秒数。 **Note** ロングポーリングが無効になっている場合、この設定に効果はありません。
maxBatchOpenMs	200ms	送信呼び出しが、同じタイプのメッセージをバッチ処理す

パラメータ	デフォルト値	説明
		る他の呼び出しを待機する最大ミリ秒。 設定を大きくすればするほど、同じ量の処理を実行するのに必要なバッチが少なくなります (ただし、バッチ内の最初の呼び出しは待機時間が長くなります)。 このパラメータを `0` に設定した場合、送信されたリクエストは他のリクエストを待機しないため、バッチ処理が事実上無効になります。
`maxBatchSize`	バッチあたり 10 個のリクエスト	1 つのバッチリクエストでまとめてバッチ処理されるメッセージの最大数。設定を大きくするほど、全体数が同じリクエストの処理に要するバッチ数が減ります。 **Note** バッチあたり 10 個のリクエストはAmazon SQSの最大許容値です。

パラメータ	デフォルト値	説明
`maxBatchSizeBytes`	1 MiB	クライアントがAmazon SQSに送信しようとするメッセージバッチの最大サイズ、バイト単位。 **Note** 1 MiB は Amazon SQS の最大許容値です。

パラメータ	デフォルト値	説明
`maxDoneReceiveBatches`	10 個のバッチ	`AmazonSQSBufferedAsyncClient` がプリフェッチし、クライアント側に保存する受信バッチの最大数。 設定を大きくすればするほど、Amazon SQSを呼び出さなくても多くの受信リクエストを満たすことができます (ただし、プリフェッチされるメッセージが多くなるほど、バッファにとどまる時間が長くなるため、それ自体の可視性タイムアウトが発生する可能性があります)。 **Note** `0` は、すべてのメッセージのプリフェッチが無効になっていて、メッセージはオンデマンドでのみ消費されることを示します。

パラメータ	デフォルト値	説明
`maxInflightOutboundBatches`	5 個のバッチ	同時に処理できるアクティブな送信バッチの最大数。 設定を大きくすればするほど、送信バッチの送信速度が速くなり (CPU や帯域幅などの他のクォータの影響を受けます)、`AmazonSQSBufferedAsyncClient` により消費されるスレッドが増えます。
`maxInflightReceiveBatches`	10 個のバッチ	同時に処理できるアクティブな受信バッチの最大数。 設定を大きくすればするほど、受信するメッセージが増え (CPU や帯域幅などの他のクォータの影響を受けます)、`AmazonSQSBufferedAsyncClient` により消費されるスレッドが増えます。 **Note** `0` は、すべてのメッセージのプリフェッチが無効になっていて、メッセージはオンデマンドでのみ消費されることを示します。

パラメータ	デフォルト値	説明
`visibilityTimeoutSeconds`	-1	このパラメータが 0 以外の正の値に設定されている場合、ここで設定した可視性タイムアウトにより、メッセージの処理元のキューで設定された可視性タイムアウトが上書きされます。 **Note** -1 は、キューのデフォルト設定が選択されていることを示します。 可視性タイムアウトを `0` に設定することはできません。

AWS SDK for Java 2.x

AWS SDK for Java 2.x では、次の例`SqsAsyncBatchManager`に基づいて新しい を作成できます。

```
// Create the basic Sqs Async Client
SqsAsyncClient sqs = SqsAsyncClient.builder()
    .region(Region.US_EAST_1)
    .build();

// Create the batch manager
SqsAsyncBatchManager sqsAsyncBatchManager = sqs.batchManager();
```

新規作成したら`SqsAsyncBatchManager`、これを使用してAmazon SQSに複数のリクエストを送信できます (`SqsAsyncClient` と同様です)。次に例を示します。

```
final String queueName = "MyAsyncBufferedQueue" + UUID.randomUUID();
final CreateQueueRequest request =
 CreateQueueRequest.builder().queueName(queueName).build();
final String queueUrl = sqs.createQueue(request).join().queueUrl();
System.out.println("Queue created: " + queueUrl);


// Send messages
CompletableFuture<SendMessageResponse> sendMessageFuture;
for (int i = 0; i < 10; i++) {
    final int index = i;
    sendMessageFuture = sqsAsyncBatchManager.sendMessage(
            r -> r.messageBody("Message " + index).queueUrl(queueUrl));
    SendMessageResponse response= sendMessageFuture.join();
    System.out.println("Message " + response.messageId() + " sent!");
}

// Receive messages with customized configurations
CompletableFuture<ReceiveMessageResponse> receiveResponseFuture =
 customizedBatchManager.receiveMessage(
        r -> r.queueUrl(queueUrl)
                .waitTimeSeconds(10)
                .visibilityTimeout(20)
                .maxNumberOfMessages(10)
);
System.out.println("You have received " +
 receiveResponseFuture.join().messages().size() + " messages in total.");

// Delete messages
DeleteQueueRequest deleteQueueRequest =
 DeleteQueueRequest.builder().queueUrl(queueUrl).build();
int code = sqs.deleteQueue(deleteQueueRequest).join().sdkHttpResponse().statusCode();
System.out.println("Queue is deleted, with statusCode " + code);
```

SqsAsyncBatchManager の設定

SqsAsyncBatchManager は、ほとんどのユースケースに合うように事前に設定されています。たとえば、SqsAsyncBatchManager をさらに設定できます。'

を使用したカスタム設定の作成SqsAsyncBatchManager.Builder：

```
SqsAsyncBatchManager customizedBatchManager = SqsAsyncBatchManager.builder()
    .client(sqs)
```

```
    .scheduledExecutor(Executors.newScheduledThreadPool(5))
    .overrideConfiguration(b -> b
        .maxBatchSize(10)
        .sendRequestFrequency(Duration.ofMillis(200))
        .receiveMessageMinWaitDuration(Duration.ofSeconds(10))
        .receiveMessageVisibilityTimeout(Duration.ofSeconds(20))
        .receiveMessageAttributeNames(Collections.singletonList("*"))

 .receiveMessageSystemAttributeNames(Collections.singletonList(MessageSystemAttributeName.ALL))
    .build();
```

`BatchOverrideConfiguration` 個のパラメータ

パラメータ	デフォルト値	説明
`maxBatchSize`	バッチあたり 10 個のリクエスト	1 つのバッチリクエストでまとめてバッチ処理されるメッセージの最大数。設定を大きくするほど、全体数が同じリクエストの処理に要するバッチ数が減ります。 **Note** Amazon SQS の最大許容値は、バッチあたり 10 リクエストです。
`sendRequestFrequency`	200ms	送信呼び出しが、同じタイプのメッセージをバッチ処理する他の呼び出しを待機する最大ミリ秒。 設定を大きくすればするほど、同じ量の処理を実行するのに必要なバッチが少なくなります (ただし、バッチ内の最

パラメータ	デフォルト値	説明
		初の呼び出しは待機時間が長くなります)。 このパラメータを `0` に設定した場合、送信されたリクエストは他のリクエストを待機しないため、バッチ処理が事実上無効になります。
`receiveMessageVisibilityTimeout`	-1	このパラメータが 0 以外の正の値に設定されている場合、ここで設定した可視性タイムアウトにより、メッセージの処理元のキューで設定された可視性タイムアウトが上書きされます。 **Note** 1 は、キューのデフォルト設定が選択されていることを示します。可視性タイムアウトを `0` に設定することはできません。
`receiveMessageMinWaitDuration`	50 ミリ秒	`receiveMessage` 呼び出しが使用可能なメッセージの取得を待機する最小時間 (ミリ秒単位) 。設定が大きいほど、同じ数のリクエストを実行するために必要なバッチが少なくなります。

Amazon SQS での水平スケーリングとアクションのバッチ処理を使用したスループットの向上

Amazon SQS は高スループットメッセージングをサポートしています。スループット制限の詳細については、「」を参照してくださいAmazon SQS のメッセージキュー。

スループットを最大化するには：

- プロデューサーとコンシューマーを水平方向にスケールするには、それぞれのインスタンスを追加します。
- アクションバッチ処理を使用して、1 つのリクエストで複数のメッセージを送受信し、API コールのオーバーヘッドを削減します。

水平スケーリング

Amazon SQSは HTTP リクエストレスポンスプロトコルを通じてアクセスするため、リクエストのレイテンシー (リクエストの開始からレスポンスの受信までの時間) により1回の接続を使用して1つのスレッドを処理した場合のスループットは制限されます。たとえば、Amazon EC2ベースのクライアントから同じリージョンにあるAmazon SQSへのレイテンシーが平均 20 ミリ秒の場合、1回の接続で1つのスレッドを処理した場合の最大スループットは平均で 50 TPS になります。

水平スケーリングには、全体的なキュースループットを高めるために、メッセージのプロデューサー (`SendMessage` リクエストを生成) とコンシューマー (`ReceiveMessage` リクエストと `DeleteMessage` リクエストを生成) の数を増やすことが必要です。水平スケーリングを行うには 3 つの方法があります。

- クライアントあたりのスレッドの数を増やす
- クライアントを追加する
- クライアントあたりのスレッドの数を増やし、クライアントを追加する

クライアントを追加すると、基本的にはキューのスループットが直線的に向上します。たとえば、クライアントの数を 2 倍にした場合、スループットも 2 倍になります。

アクションバッチ処理

バッチ処理では、サービスへの各ラウンドトリップでより多くの処理が実行されます (たとえば、1 回の `SendMessageBatch` リクエストで複数のメッセージを送信する場合な

ど)。Amazon SQS バッチ アクションは、SendMessageBatch、DeleteMessageBatch、および ChangeMessageVisibilityBatch です。プロデューサーやコンシューマーを変更することなくバッチ処理を活用するには、Amazon SQSバファリング非同期クライアント を使用します。

> **Note**
>
> ReceiveMessage は、一度に 10 件のメッセージを処理できるため、ReceiveMessageBatch アクションはありません。

バッチ処理では、1 件のメッセージのレイテンシー全体が受け入れられるのではなく、1 回のバッチリクエストの複数のメッセージにまたがるバッチアクションのレイテンシーが分散されます (たとえば、SendMessage リクエストなど)。各ラウンドトリップがより多くの処理を実行するため、バッチリクエストがスレッドと接続をより効率的に使用するようになり、スループットが向上します。

マッチ処理と水平スケーリングを組み合わせて、個々のメッセージリクエストより少ないスレッド、接続、リクエストで一定のスループットを実現できます。バッチ処理されたAmazon SQSアクションを使用して、最大 10 通のメッセージを一度に送信、受信、または削除できます。Amazon SQSではリクエスト単位で課金されるため、バッチ処理はコストを大幅に削減できます。

バッチ処理により、アプリケーションがいくらか複雑になる可能性はあります (たとえば、アプリケーションはメッセージを送信前に累積する必要があります。または、レスポンスを長時間待機する必要が生じることもときどきあります)。しかし、それでもバッチ処理は次の場合に効果的です:

- アプリケーションが短い時間で多くのメッセージを生成するため、遅延が大幅に長くなることはない。
- 一般的なメッセージプロデューサーが自身でコントロールしていないイベントに応答してメッセージを送信する必要があるのと異なり、メッセージコンシューマーは自身の判断でキューからメッセージを取得する。

> **Important**
>
> バッチ内の個々のメッセージが失敗しても、バッチリクエストは成功することがあります。バッチリクエストの後、必ず個々のメッセージのエラーがないか確認し、必要に応じてアクションを再試行してください。

1 回のオペレーションおよびバッチリクエストでの Java の使用例

前提条件

`aws-java-sdk-sqs.jar`パッケージ、`aws-java-sdk-ec2.jar` パッケージおよび `commons-logging.jar` パッケージを Java ビルドクラスパスに追加します。以下の例は、Maven プロジェクトの `pom.xml` ファイルにあるこれらの依存関係を示しています。

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-ec2</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>LATEST</version>
    </dependency>
</dependencies>
```

SimpleProducerConsumer.java

次の Java コード例では、簡単なプロデューサー-コンシューマーパターンが実行されています。メインスレッドでは、指定された時間に 1 KB のメッセージを処理するプロデューサーおよびコンシューマースレッドが多数発生します。この例には、単一オペレーションリクエストを生成するプロデューサーおよびコンシューマーと、バッチ処理リクエストを生成するプロデューサーおよびコンシューマーが含まれています。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
```

```
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

import com.amazonaws.AmazonClientException;
import com.amazonaws.ClientConfiguration;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Start a specified number of producer and consumer threads, and produce-consume
 * for the least of the specified duration and 1 hour. Some messages can be left
 * in the queue because producers and consumers might not be in exact balance.
 */
public class SimpleProducerConsumer {

    // The maximum runtime of the program.
    private final static int MAX_RUNTIME_MINUTES = 60;
    private final static Log log = LogFactory.getLog(SimpleProducerConsumer.class);

    public static void main(String[] args) throws InterruptedException {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the number of producers: ");
```

```
        final int producerCount = input.nextInt();

        System.out.print("Enter the number of consumers: ");
        final int consumerCount = input.nextInt();

        System.out.print("Enter the number of messages per batch: ");
        final int batchSize = input.nextInt();

        System.out.print("Enter the message size in bytes: ");
        final int messageSizeByte = input.nextInt();

        System.out.print("Enter the run time in minutes: ");
        final int runTimeMinutes = input.nextInt();

        /*
         * Create a new instance of the builder with all defaults (credentials
         * and region) set automatically. For more information, see Creating
         * Service Clients in the AWS SDK for Java Developer Guide.
         */
        final ClientConfiguration clientConfiguration = new ClientConfiguration()
                .withMaxConnections(producerCount + consumerCount);

        final AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()
                .withClientConfiguration(clientConfiguration)
                .build();

        final String queueUrl = sqsClient
                .getQueueUrl(new GetQueueUrlRequest(queueName)).getQueueUrl();

        // The flag used to stop producer, consumer, and monitor threads.
        final AtomicBoolean stop = new AtomicBoolean(false);

        // Start the producers.
        final AtomicInteger producedCount = new AtomicInteger();
        final Thread[] producers = new Thread[producerCount];
        for (int i = 0; i < producerCount; i++) {
            if (batchSize == 1) {
                producers[i] = new Producer(sqsClient, queueUrl, messageSizeByte,
                        producedCount, stop);
            } else {
                producers[i] = new BatchProducer(sqsClient, queueUrl, batchSize,
                        messageSizeByte, producedCount,
                        stop);
            }
```

```
            producers[i].start();
        }

        // Start the consumers.
        final AtomicInteger consumedCount = new AtomicInteger();
        final Thread[] consumers = new Thread[consumerCount];
        for (int i = 0; i < consumerCount; i++) {
            if (batchSize == 1) {
                consumers[i] = new Consumer(sqsClient, queueUrl, consumedCount,
                        stop);
            } else {
                consumers[i] = new BatchConsumer(sqsClient, queueUrl, batchSize,
                        consumedCount, stop);
            }
            consumers[i].start();
        }

        // Start the monitor thread.
        final Thread monitor = new Monitor(producedCount, consumedCount, stop);
        monitor.start();

        // Wait for the specified amount of time then stop.
        Thread.sleep(TimeUnit.MINUTES.toMillis(Math.min(runTimeMinutes,
                MAX_RUNTIME_MINUTES)));
        stop.set(true);

        // Join all threads.
        for (int i = 0; i < producerCount; i++) {
            producers[i].join();
        }

        for (int i = 0; i < consumerCount; i++) {
            consumers[i].join();
        }

        monitor.interrupt();
        monitor.join();
    }

    private static String makeRandomString(int sizeByte) {
        final byte[] bs = new byte[(int) Math.ceil(sizeByte * 5 / 8)];
        new Random().nextBytes(bs);
        bs[0] = (byte) ((bs[0] | 64) & 127);
        return new BigInteger(bs).toString(32);
```

```
    }

    /**
     * The producer thread uses {@code SendMessage}
     * to send messages until it is stopped.
     */
    private static class Producer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        Producer(AmazonSQS sqsQueueBuffer, String queueUrl, int messageSizeByte,
                 AtomicInteger producedCount, AtomicBoolean stop) {
            this.sqsClient = sqsQueueBuffer;
            this.queueUrl = queueUrl;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /*
         * The producedCount object tracks the number of messages produced by
         * all producer threads. If there is an error, the program exits the
         * run() method.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    sqsClient.sendMessage(new SendMessageRequest(queueUrl,
                            theMessage));
                    producedCount.incrementAndGet();
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("Producer: " + e.getMessage());
                System.exit(1);
            }
        }
    }
```

```
    /**
     * The producer thread uses {@code SendMessageBatch}
     * to send messages until it is stopped.
     */
    private static class BatchProducer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        BatchProducer(AmazonSQS sqsQueueBuffer, String queueUrl, int batchSize,
                      int messageSizeByte, AtomicInteger producedCount,
                      AtomicBoolean stop) {
            this.sqsClient = sqsQueueBuffer;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        public void run() {
            try {
                while (!stop.get()) {
                    final SendMessageBatchRequest batchRequest =
                            new SendMessageBatchRequest().withQueueUrl(queueUrl);

                    final List<SendMessageBatchRequestEntry> entries =
                            new ArrayList<SendMessageBatchRequestEntry>();
                    for (int i = 0; i < batchSize; i++)
                        entries.add(new SendMessageBatchRequestEntry()
                                .withId(Integer.toString(i))
                                .withMessageBody(theMessage));
                    batchRequest.setEntries(entries);

                    final SendMessageBatchResult batchResult =
                            sqsClient.sendMessageBatch(batchRequest);
                    producedCount.addAndGet(batchResult.getSuccessful().size());

                    /*
                     * Because SendMessageBatch can return successfully, but
```

```
                     * individual batch items fail, retry the failed batch items.
                     */
                    if (!batchResult.getFailed().isEmpty()) {
                        log.warn("Producer: retrying sending "
                                + batchResult.getFailed().size() + " messages");
                        for (int i = 0, n = batchResult.getFailed().size();
                             i < n; i++) {
                            sqsClient.sendMessage(new
                                    SendMessageRequest(queueUrl, theMessage));
                            producedCount.incrementAndGet();
                        }
                    }
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("BatchProducer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * The consumer thread uses {@code ReceiveMessage} and {@code DeleteMessage}
     * to consume messages until it is stopped.
     */
    private static class Consumer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;

        Consumer(AmazonSQS sqsClient, String queueUrl, AtomicInteger consumedCount,
                 AtomicBoolean stop) {
            this.sqsClient = sqsClient;
            this.queueUrl = queueUrl;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /*
         * Each consumer thread receives and deletes messages until the main
```

```
         * thread stops the consumer thread. The consumedCount object tracks the
         * number of messages that are consumed by all consumer threads, and the
         * count is logged periodically.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    try {
                        final ReceiveMessageResult result = sqsClient
                                .receiveMessage(new
                                        ReceiveMessageRequest(queueUrl));

                        if (!result.getMessages().isEmpty()) {
                            final Message m = result.getMessages().get(0);
                            sqsClient.deleteMessage(new
                                    DeleteMessageRequest(queueUrl,
                                    m.getReceiptHandle()));
                            consumedCount.incrementAndGet();
                        }
                    } catch (AmazonClientException e) {
                        log.error(e.getMessage());
                    }
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("Consumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * The consumer thread uses {@code ReceiveMessage} and {@code
     * DeleteMessageBatch} to consume messages until it is stopped.
     */
    private static class BatchConsumer extends Thread {
        final AmazonSQS sqsClient;
        final String queueUrl;
        final int batchSize;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;
```

```
        BatchConsumer(AmazonSQS sqsClient, String queueUrl, int batchSize,
                      AtomicInteger consumedCount, AtomicBoolean stop) {
            this.sqsClient = sqsClient;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        public void run() {
            try {
                while (!stop.get()) {
                    final ReceiveMessageResult result = sqsClient
                            .receiveMessage(new ReceiveMessageRequest(queueUrl)
                                    .withMaxNumberOfMessages(batchSize));

                    if (!result.getMessages().isEmpty()) {
                        final List<Message> messages = result.getMessages();
                        final DeleteMessageBatchRequest batchRequest =
                                new DeleteMessageBatchRequest()
                                        .withQueueUrl(queueUrl);

                        final List<DeleteMessageBatchRequestEntry> entries =
                                new ArrayList<DeleteMessageBatchRequestEntry>();
                        for (int i = 0, n = messages.size(); i < n; i++)
                            entries.add(new DeleteMessageBatchRequestEntry()
                                    .withId(Integer.toString(i))
                                    .withReceiptHandle(messages.get(i)
                                            .getReceiptHandle()));
                        batchRequest.setEntries(entries);

                        final DeleteMessageBatchResult batchResult = sqsClient
                                .deleteMessageBatch(batchRequest);
                        consumedCount.addAndGet(batchResult.getSuccessful().size());

                        /*
                         * Because DeleteMessageBatch can return successfully,
                         * but individual batch items fail, retry the failed
                         * batch items.
                         */
                        if (!batchResult.getFailed().isEmpty()) {
                            final int n = batchResult.getFailed().size();
                            log.warn("Producer: retrying deleting " + n
```

```
                                    + " messages");
                            for (BatchResultErrorEntry e : batchResult
                                    .getFailed()) {

                                sqsClient.deleteMessage(
                                        new DeleteMessageRequest(queueUrl,
                                                messages.get(Integer
                                                        .parseInt(e.getId()))
                                                        .getReceiptHandle()));

                                consumedCount.incrementAndGet();
                            }
                        }
                    }
                }
            } catch (AmazonClientException e) {
                /*
                 * By default, AmazonSQSClient retries calls 3 times before
                 * failing. If this unlikely condition occurs, stop.
                 */
                log.error("BatchConsumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * This thread prints every second the number of messages produced and
     * consumed so far.
     */
    private static class Monitor extends Thread {
        private final AtomicInteger producedCount;
        private final AtomicInteger consumedCount;
        private final AtomicBoolean stop;

        Monitor(AtomicInteger producedCount, AtomicInteger consumedCount,
                AtomicBoolean stop) {
            this.producedCount = producedCount;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        public void run() {
            try {
```

```
                while (!stop.get()) {
                    Thread.sleep(1000);
                    log.info("produced messages = " + producedCount.get()
                            + ", consumed messages = " + consumedCount.get());
                }
            } catch (InterruptedException e) {
                // Allow the thread to exit.
            }
        }
    }
}
```

サンプル実行からのボリュームメトリクスのモニタリング

Amazon SQSは、メッセージの送信、受信、削除のボリュームメトリクスを自動的に生成します。これらのメトリックスと他のメトリックスにアクセスするには、キューの { モニタリング } タブを使用するか、[CloudWatchコンソール]を使用します。

> **Note**
>
> メトリクスを、キューが開始してから参照可能になるまで最大 15 分かかる場合があります。

AWS SDK での Amazon SQS 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利用できます。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する 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

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

可用性の例

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

Amazon SQS での JMS の使用

Amazon SQS Java Messaging Library は、Amazon SQS 用の Java Message Service (JMS) インターフェイスで、既に JMS を使用しているアプリケーションで Amazon SQS を利用できます。このインターフェイスにより、最小限のコードの変更で、 Amazon SQS をJMSプロバイダーとして使用できます。 AWS SDK for Javaと一緒に Amazon SQS Java Messaging Library を使用すると、JMS 接続およびセッション、Amazon SQS キューとの間でメッセージを送受信するプロデューサーおよびコンシューマーを作成することができます。

ライブラリは、JMS1.1の仕様に規定されている、キューとのメッセージの送受信 (JMSポイントツーポイントモデル) をサポートしています。ライブラリは、テキスト、バイト、または Amazon SQS キューへのオブジェクトメッセージの同期的な送信をサポートしています。また、オブジェクトの同期的または非同期的な受信もサポートしています。

JMS 1.1 仕様をサポートする Amazon SQS Java Messaging Library の機能の詳細については、「Amazon SQS でサポートされている JMS 1.1 実装」および「Amazon SQS に関するよくある質問」を参照してください。

JMS と Amazon SQS を使用するための前提条件

始めるには以下の前提条件を満たす必要があります:

- SDK for Java

  プロジェクトにSDK for Javaを含めるには以下の2つの方法があります:

  - SDK for Java をダウンロードしてインストールします。
  - Maven を使用して Amazon SQS Java Messaging Library を入手します。

  > **Note**
  >
  > SDK for Javaは依存関係として含められます。
  > SDK for Java と Java 用 Amazon SQS 拡張クライアントライブラリには、J2SE Development Kit 8.0 以降が必要です。

  SDK for Java のダウンロードの詳細については、「 SDK for Java」を参照してください。

- Amazon SQS Java Messaging Library

Mavenを使用しない場合は、`amazon-sqs-java-messaging-lib.jar`パッケージをJavaクラスパスに追加する必要があります。ライブラリのダウンロードの詳細については、「Amazon SQS Java Messaging Library」を参照してください。

> **Note**
>
> Amazon SQS Java Messaging Library には、Mavenと Spring フレームワークのサポートが含まれます。
> Maven、Spring フレームワーク、Amazon SQS Java Messaging Library を使用するサンプルコードについては、「Amazon SQS 標準キューで JMS を使用するための実用的な Java の例」を参照してください。
>
> ```
> <dependency>
>  <groupId>com.amazonaws</groupId>
>  <artifactId>amazon-sqs-java-messaging-lib</artifactId>
>  <version>1.0.4</version>
>  <type>jar</type>
> </dependency>
> ```

- Amazon SQS キュー

  AWS Management Console for Amazon SQS、 `CreateQueue` API、または Amazon SQS Java Messaging Library に含まれているラップされた Amazon SQS クライアントを使用してキューを作成します。

  - Amazon SQS AWS Management Console または`CreateQueue`の APIを使用してキューを作成する方法については、「キューの作成」を参照してください。
  - Amazon SQS Java Messaging Library の使用の詳細については、「Amazon SQS Java Messaging Library の使用」を参照してください。

Amazon SQS Java Messaging Library の使用

Amazon SQSでJava Message Service (JMS)の使用を開始するには、このセクションのコード例を使用します。以降のセクションでは、JMS接続およびセッションの作成方法や、メッセージの送受信方法を説明します。

Amazon SQS Java Messaging Library に含まれるラップされた Amazon SQS クライアントオブジェクトは、Amazon SQS キューが存在するかどうかをチェックします。キューが存在しない場合は、クライアントがキューを作成します。

JMS接続の作成

開始する前に、「JMS と Amazon SQS を使用するための前提条件」の前提条件を参照してください。

1. 接続ファクトリを作成し、そのファクトリに対してcreateConnection メソッドを呼び出します。

```
// Create a new connection factory with all defaults (credentials and region) set
 automatically
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        AmazonSQSClientBuilder.defaultClient()
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

   SQSConnectionクラスは javax.jms.Connection を拡張します。JMSのスタンダード接続メソッドと共に、SQSConnectionは、getAmazonSQSClientやgetWrappedAmazonSQSClientなどの追加のメソッドも提供します。どちらのメッソドでも、新しいキューの作成などのJMS仕様には含まれていない管理操作を実行できます。ただし、getWrappedAmazonSQSClientメソッドは現在接続で使用されているAmazon SQSクライアントのラップされたバージョンも提供します。ラッパーはクライアントからのすべての例外を JMSException に変換するので、JMSException を想定する既存のコードでより容易に使用できます。

2. getAmazonSQSClientやgetWrappedAmazonSQSClientから返されるクライアントオブジェクトを使用して、JMS 仕様には含まれていない管理操作 (Amazon SQS キューの作成など) を実行できます。

   既存のコードで JMS 例外を想定している場合は、getWrappedAmazonSQSClient を使用する必要があります:

   - getWrappedAmazonSQSClientを使用する場合、返されるクライアントオブジェクトはすべての例外をJMS例外に変換します。

- `getAmazonSQSClient`を使用する場合、例外はすべてAmazon SQS例外になります。

Amazon SQSキューを作成する

ラップされたクライアントオブジェクトは、 Amazon SQSキューが存在するかどうかを確認します。

キューが存在しない場合は、クライアントがキューを作成します。キューが存在する場合、関数からは何も返されません。詳細については、TextMessageSender.java にある「必要に応じてキューを作成する」セクションを参照してください。

標準キューを作成するには

```
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an SQS queue named MyQueue, if it doesn't already exist
if (!client.queueExists("MyQueue")) {
    client.createQueue("MyQueue");
}
```

FIFOキューを作成するには

```
// Get the wrapped client
AmazonSQSMessagingClientWrapper client = connection.getWrappedAmazonSQSClient();

// Create an Amazon SQS FIFO queue named MyQueue.fifo, if it doesn't already exist
if (!client.queueExists("MyQueue.fifo")) {
    Map<String, String> attributes = new HashMap<String, String>();
    attributes.put("FifoQueue", "true");
    attributes.put("ContentBasedDeduplication", "true");
    client.createQueue(new
 CreateQueueRequest().withQueueName("MyQueue.fifo").withAttributes(attributes));
}
```

> **Note**
>
> FIFO キュー名は`.fifo`のサフィックスで終わる必要があります。

ContentBasedDeduplication属性の詳細については、「Amazon SQS の 1 回のみ処理」を参照してください。

同期的なメッセージの送信

1. 接続と基になる Amazon SQS キューの準備ができたら、 AUTO_ACKNOWLEDGEモードで非トランザクションJMSセッションを作成します。

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

2. テキストメッセージをキューに送信するには、JMS キュー ID とメッセージプロデューサーを作成します。

```
// Create a queue identity and specify the queue name to the session
Queue queue = session.createQueue("MyQueue");

// Create a producer for the 'MyQueue'
MessageProducer producer = session.createProducer(queue);
```

3. テキストメッセージを作成し、キューに送信します。

   - メッセージを標準キューに送信するために、追加のパラメータを設定する必要はありません。

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

   - メッセージをFIFOキューに送信するには、メッセージのグループ IDを設定する必要があります。メッセージ重複排除IDを設定することもできます。詳細については、「Amazon SQS の FIFO キューの主要な用語」を参照してください。

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Set the message group ID
message.setStringProperty("JMSXGroupID", "Default");
```

```
// You can also set a custom message deduplication ID
// message.setStringProperty("JMS_SQS_DeduplicationId", "hello");
// Here, it's not needed because content-based deduplication is enabled for the
 queue

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
System.out.println("JMS Message Sequence Number " +
 message.getStringProperty("JMS_SQS_SequenceNumber"));
```

同期的なメッセージの受信

1. メッセージを受信するには、同じキューにコンシューマーを作成し、startメソッドを呼び出します。

   接続でのstartメソッドはいつでも呼び出すことができます。ただし、コンシューマーは呼び出されるまでメッセージの受信を開始しません。

   ```
   // Create a consumer for the 'MyQueue'
   MessageConsumer consumer = session.createConsumer(queue);
   // Start receiving incoming messages
   connection.start();
   ```

2. コンシューマーで、タイムアウトを1秒に設定してreceiveメソッドを呼び出し、受信メッセージの内容を出力します。

   - 標準キューからメッセージを受信したら、メッセージの内容にアクセスできます。

     ```
     // Receive a message from 'MyQueue' and wait up to 1 second
     Message receivedMessage = consumer.receive(1000);

     // Cast the received message as TextMessage and display the text
     if (receivedMessage != null) {
         System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
     }
     ```

   - FIFOキューからメッセージを受信したら、メッセージの内容や、その他のFIFO 固有のメッセージ属性 (メッセージグループ ID、メッセージ重複排除 ID、シーケンス番号など) にアクセ

スできます。詳細については、「Amazon SQS の FIFO キューの主要な用語」を参照してください。

```
// Receive a message from 'MyQueue' and wait up to 1 second
Message receivedMessage = consumer.receive(1000);

// Cast the received message as TextMessage and display the text
if (receivedMessage != null) {
    System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
    System.out.println("Group id: " +
 receivedMessage.getStringProperty("JMSXGroupID"));
    System.out.println("Message deduplication id: " +
 receivedMessage.getStringProperty("JMS_SQS_DeduplicationId"));
    System.out.println("Message sequence number: " +
 receivedMessage.getStringProperty("JMS_SQS_SequenceNumber"));
}
```

3. 接続とセッションを閉じます。

```
// Close the connection (and the session).
connection.close();
```

出力は次の例のようになります:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

> **Note**
>
> Spring Framework を使用してこれらのオブジェクトを初期化できます。追加情報については、「SpringExampleConfiguration.xml」、SpringExample.java、およびExampleConfiguration.javaのほか、ExampleCommon.javaおよびAmazon SQS 標準キューで JMS を使用するための実用的な Java の例 に含まれる他のヘルパークラスを参照してください。

オブジェクトの送受信の完全な例については、TextMessageSender.javaおよびSyncMessageReceiver.javaを参照してください。

非同期的なメッセージの受信

「Amazon SQS Java Messaging Library の使用」の例では、メッセージはMyQueueに送信され、同期的に受信されます。

以下の例では、リスナーを介してメッセージを非同期的に受信する方法を示します。

1. MessageListenerインターフェイスを実装します。

```
class MyListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        try {
            // Cast the received message as TextMessage and print the text to
 screen.
            System.out.println("Received: " + ((TextMessage) message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

   onMessageインターフェイスのMessageListenerメソッドは、メッセージを受信すると呼び出されます。このリスナーの実装内で、メッセージに格納されたテキストが出力されます。

2. コンシューマーで明示的にreceiveメソッドを呼び出す代わりに、コンシューマーのメッセージリスナーをMyListener実装のインスタンスに設定します。メインスレッドは1秒間待機します。

```
// Create a consumer for the 'MyQueue'.
MessageConsumer consumer = session.createConsumer(queue);

// Instantiate and set the message listener for the consumer.
consumer.setMessageListener(new MyListener());

// Start receiving incoming messages.
connection.start();

// Wait for 1 second. The listener onMessage() method is invoked when a message is
 received.
Thread.sleep(1000);
```

残りの手順は、「Amazon SQS Java Messaging Library の使用」の例の手順と同じです。非同期コンシューマーの完全な例については、「AsyncMessageReceiver.java」のAmazon SQS 標準キューで JMS を使用するための実用的な Java の例を参照してください。

この例の出力は以下の例のようになります:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

クライアント確認モードの使用

「Amazon SQS Java Messaging Library の使用」の例ではAUTO_ACKNOWLEDGE モードを使用しています。この場合、受信したすべてのメッセージは自動的に確認されます (このため、基になるAmazon SQSキューから削除されます)。

1. メッセージが処理された後にメッセージを明示的に確認するには、CLIENT_ACKNOWLEDGEモードでセッションを作成する必要があります。

   ```
   // Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
   Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
   ```

2. メッセージが受信されたら、メッセージを表示して、その後明示的に確認します。

   ```
   // Cast the received message as TextMessage and print the text to screen. Also
    acknowledge the message.
   if (receivedMessage != null) {
       System.out.println("Received: " + ((TextMessage) receivedMessage).getText());
       receivedMessage.acknowledge();
       System.out.println("Acknowledged: " + message.getJMSMessageID());
   }
   ```

   > **Note**
   >
   > このモードでは、メッセージが確認されると、そのメッセージ以前に受信されたすべてのメッセージも暗黙的に確認されます。たとえば、10通のメッセージが受信され、(受信した順序で)10番目のメッセージのみが確認された場合、前の9通のメッセージもすべて確認されます。

残りの手順は、「Amazon SQS Java Messaging Library の使用」の例の手順と同じです。クライアント確認モードの同期コンシューマーの完全な例については、「SyncMessageReceiverClientAcknowledge.java」のAmazon SQS 標準キューで JMS を使用するための実用的な Java の例を参照してください。

この例の出力は以下の例のようになります:

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5
Received: Hello World!
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

順不同確認モードの使用

CLIENT_ACKNOWLEDGEモードを使用すると、明示的に確認されたメッセージの前に受信されたすべてのメッセージが自動的に確認されます。詳細については、「クライアント確認モードの使用」を参照してください。

Amazon SQS Java Messaging Library には別の確認モードも用意されています。UNORDERED_ACKNOWLEDGE モードを使用する場合は、受信した順序に関係なく、すべての受信メッセージを個別かつ明示的にクライアントが確認する必要があります。これには、UNORDERED_ACKNOWLEDGEモードでセッションを作成します。

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

残りの手順は、「クライアント確認モードの使用」の例の手順と同じです。UNORDERED_ACKNOWLEDGEモードの同期コンシューマーの完全な例については、SyncMessageReceiverUnorderedAcknowledge.javaを参照してください。

この例の出力は以下のようになります:

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

Java Message Service を他の Amazon SQS クライアントで使用する

AWS SDK で Amazon SQS Java Message Service (JMS) クライアントを使用すると、Amazon SQS メッセージサイズは 256 KB に制限されます。ただし、任意のAmazon SQSクライアントを使用してJMSプロバイダを作成することができます。たとえば、Java用の Amazon SQS 拡張クライアントライブラリ とJMSクライアントを使用する場合、Amazon S3内のメッセージペイロード (最大2GB) への参照を含むAmazon SQSメッセージを送信することができます。詳細については、「Java と Amazon S3 を使用した大量の Amazon SQS メッセージの管理」を参照してください。

次の Java コード例では、拡張クライアントライブラリの JMS プロバイダーを作成します。

この例をテストする前に、「JMS と Amazon SQS を使用するための前提条件」の前提条件を参照してください。

```
AmazonS3 s3 = new AmazonS3Client(credentials);
Region s3Region = Region.getRegion(Regions.US_WEST_2);
s3.setRegion(s3Region);

// Set the Amazon S3 bucket name, and set a lifecycle rule on the bucket to
// permanently delete objects a certain number of days after each object's creation
 date.
// Next, create the bucket, and enable message objects to be stored in the bucket.
BucketLifecycleConfiguration.Rule expirationRule = new
 BucketLifecycleConfiguration.Rule();
expirationRule.withExpirationInDays(14).withStatus("Enabled");
BucketLifecycleConfiguration lifecycleConfig = new
 BucketLifecycleConfiguration().withRules(expirationRule);

s3.createBucket(s3BucketName);
s3.setBucketLifecycleConfiguration(s3BucketName, lifecycleConfig);
System.out.println("Bucket created and configured.");

// Set the SQS extended client configuration with large payload support enabled.
ExtendedClientConfiguration extendedClientConfig = new ExtendedClientConfiguration()
    .withLargePayloadSupportEnabled(s3, s3BucketName);

AmazonSQS sqsExtended = new AmazonSQSExtendedClient(new AmazonSQSClient(credentials),
 extendedClientConfig);
Region sqsRegion = Region.getRegion(Regions.US_WEST_2);
sqsExtended.setRegion(sqsRegion);
```

次のJavaコード例は、接続ファクトリを作成しています:

```
// Create the connection factory using the environment variable credential provider.
// Pass the configured Amazon SQS Extended Client to the JMS connection factory.
SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
        new ProviderConfiguration(),
        sqsExtended
        );

// Create the connection.
SQSConnection connection = connectionFactory.createConnection();
```

Amazon SQS 標準キューで JMS を使用するための実用的な Java の例

以下のコード例では、Java Message Service (JMS) を Amazon SQS の標準キューで使用する方法を示します。FIFO キューの操作方法の詳細については、「FIFOキューを作成するには」、「同期的なメッセージの送信」、および「同期的なメッセージの受信」を参照してください。(メッセージを同期して受信することは、標準キューでも FIFO キューでも同じです。ただし、FIFO キューのメッセージには、より多くの属性が含まれます)。

以下の例をテストする前に、「JMS と Amazon SQS を使用するための前提条件」の前提条件を参照してください。

ExampleConfiguration.java

次の Java コード例では、他の Java の例で使用するデフォルトのキュー名、リージョン、認証情報を設定します。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
```

```
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleConfiguration {
    public static final String DEFAULT_QUEUE_NAME = "SQSJMSClientExampleQueue";

    public static final Region DEFAULT_REGION = Region.getRegion(Regions.US_EAST_2);

    private static String getParameter( String args[], int i ) {
        if( i + 1 >= args.length ) {
            throw new IllegalArgumentException( "Missing parameter for " + args[i] );
        }
        return args[i+1];
    }

    /**
     * Parse the command line and return the resulting config. If the config parsing
 fails
     * print the error and the usage message and then call System.exit
     *
     * @param app the app to use when printing the usage string
     * @param args the command line arguments
     * @return the parsed config
     */
    public static ExampleConfiguration parseConfig(String app, String args[]) {
        try {
            return new ExampleConfiguration(args);
        } catch (IllegalArgumentException e) {
            System.err.println( "ERROR: " + e.getMessage() );
            System.err.println();
            System.err.println( "Usage: " + app + " [--queue <queue>] [--region
 <region>] [--credentials <credentials>] ");
            System.err.println( "  or" );
            System.err.println( "       " + app + " <spring.xml>" );
            System.exit(-1);
            return null;
        }
    }

    private ExampleConfiguration(String args[]) {
        for( int i = 0; i < args.length; ++i ) {
            String arg = args[i];
```

```
            if( arg.equals( "--queue" ) ) {
                setQueueName(getParameter(args, i));
                i++;
            } else if( arg.equals( "--region" ) ) {
                String regionName = getParameter(args, i);
                try {
                    setRegion(Region.getRegion(Regions.fromName(regionName)));
                } catch( IllegalArgumentException e ) {
                    throw new IllegalArgumentException( "Unrecognized region " +
regionName );
                }
                i++;
            } else if( arg.equals( "--credentials" ) ) {
                String credsFile = getParameter(args, i);
                try {
                    setCredentialsProvider( new
PropertiesFileCredentialsProvider(credsFile) );
                } catch (AmazonClientException e) {
                    throw new IllegalArgumentException("Error reading credentials from
" + credsFile, e );
                }
                i++;
            } else {
                throw new IllegalArgumentException("Unrecognized option " + arg);
            }
        }
    }

    private String queueName = DEFAULT_QUEUE_NAME;
    private Region region = DEFAULT_REGION;
    private AWSCredentialsProvider credentialsProvider = new
DefaultAWSCredentialsProviderChain();

    public String getQueueName() {
        return queueName;
    }

    public void setQueueName(String queueName) {
        this.queueName = queueName;
    }

    public Region getRegion() {
        return region;
    }
```

```
    public void setRegion(Region region) {
        this.region = region;
    }

    public AWSCredentialsProvider getCredentialsProvider() {
        return credentialsProvider;
    }

    public void setCredentialsProvider(AWSCredentialsProvider credentialsProvider) {
        // Make sure they're usable first
        credentialsProvider.getCredentials();
        this.credentialsProvider = credentialsProvider;
    }
}
```

TextMessageSender.java

次のJavaコード例では、テキストメッセージプロデューサーを作成しています。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class TextMessageSender {
    public static void main(String args[]) throws JMSException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("TextMessageSender", args);

        ExampleCommon.setupLogging();
```

```
        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        MessageProducer producer =
 session.createProducer( session.createQueue( config.getQueueName() ) );

        sendMessages(session, producer);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }

    private static void sendMessages( Session session, MessageProducer producer ) {
        BufferedReader inputReader = new BufferedReader(
            new InputStreamReader( System.in, Charset.defaultCharset() ) );

        try {
            String input;
            while( true ) {
                System.out.print( "Enter message to send (leave empty to exit): " );
                input = inputReader.readLine();
                if( input == null || input.equals("" ) ) break;

                TextMessage message = session.createTextMessage(input);
                producer.send(message);
                System.out.println( "Send message " + message.getJMSMessageID() );
            }
        } catch (EOFException e) {
            // Just return on EOF
        } catch (IOException e) {
```

```
            System.err.println( "Failed reading input: " + e.getMessage() );
        } catch (JMSException e) {
            System.err.println( "Failed sending message: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

SyncMessageReceiver.java

次のJavaコード例では、同期メッセージコンシューマーを作成しています。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class SyncMessageReceiver {
public static void main(String args[]) throws JMSException {
    ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiver", args);

    ExampleCommon.setupLogging();

    // Create the connection factory based on the config
    SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
            new ProviderConfiguration(),
            AmazonSQSClientBuilder.standard()
                    .withRegion(config.getRegion().getName())
                    .withCredentials(config.getCredentialsProvider())
            );
```

```
    // Create the connection
    SQSConnection connection = connectionFactory.createConnection();

    // Create the queue if needed
    ExampleCommon.ensureQueueExists(connection, config.getQueueName());

    // Create the session
    Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
    MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

    connection.start();

    receiveMessages(session, consumer);

    // Close the connection. This closes the session automatically
    connection.close();
    System.out.println( "Connection closed" );
}

private static void receiveMessages( Session session, MessageConsumer consumer ) {
    try {
        while( true ) {
            System.out.println( "Waiting for messages");
            // Wait 1 minute for a message
            Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
            if( message == null ) {
                System.out.println( "Shutting down after 1 minute of silence" );
                break;
            }
            ExampleCommon.handleMessage(message);
            message.acknowledge();
            System.out.println( "Acknowledged message " + message.getJMSMessageID() );
        }
    } catch (JMSException e) {
        System.err.println( "Error receiving from SQS: " + e.getMessage() );
        e.printStackTrace();
    }
}
}
```

AsyncMessageReceiver.java

次のJavaコード例では、非同期メッセージコンシューマーを作成しています。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class AsyncMessageReceiver {
    public static void main(String args[]) throws JMSException, InterruptedException {
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("AsyncMessageReceiver", args);

        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

```
        MessageConsumer consumer =
 session.createConsumer( session.createQueue( config.getQueueName() ) );

        // No messages are processed until this is called
        connection.start();

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );

        callback.waitForOneMinuteOfSilence();
        System.out.println( "Returning after one minute of silence" );

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println( "Connection closed" );
    }


    private static class ReceiverCallback implements MessageListener {
        // Used to listen for message silence
        private volatile long timeOfLastMessage = System.nanoTime();

        public void waitForOneMinuteOfSilence() throws InterruptedException {
            for(;;) {
                long timeSinceLastMessage = System.nanoTime() - timeOfLastMessage;
                long remainingTillOneMinuteOfSilence =
                    TimeUnit.MINUTES.toNanos(1) - timeSinceLastMessage;
                if( remainingTillOneMinuteOfSilence < 0 ) {
                    break;
                }
                TimeUnit.NANOSECONDS.sleep(remainingTillOneMinuteOfSilence);
            }
        }


        @Override
        public void onMessage(Message message) {
            try {
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message " +
 message.getJMSMessageID() );
                timeOfLastMessage = System.nanoTime();
            } catch (JMSException e) {
```

```
                System.err.println( "Error processing message: " + e.getMessage() );
                e.printStackTrace();
            }
        }
    }
}
```

SyncMessageReceiverClientAcknowledge.java

次のJavaコード例では、クライアント確認モードの同期コンシューマーを作成しています。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

/**
 * An example class to demonstrate the behavior of CLIENT_ACKNOWLEDGE mode for received
 messages. This example
 * complements the example given in {@link SyncMessageReceiverUnorderedAcknowledge} for
 UNORDERED_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that no message is returned for this
 attempt since in CLIENT_ACKNOWLEDGE mode,
 * as expected, all the messages prior to the acknowledged messages are also
 acknowledged.
 *
```

```
 * This ISN'T the behavior for UNORDERED_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverUnorderedAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverClientAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverClientAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with client acknowledge mode
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
 session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
 session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();
```

```
        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
receiving no messages since
        // we have acknowledged the second message. Although we didn't explicitly
acknowledge the first message,
        // in the CLIENT_ACKNOWLEDGE mode, all the messages received prior to the
explicitly acknowledged message
        // are also acknowledged. Therefore, we have implicitly acknowledged the first
message.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }
```

```
    /**
     * Receives a message through the consumer synchronously with the default timeout
 (TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
 "Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
 acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
 throws JMSException {
        // Receive a message
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

SyncMessageReceiverUnorderedAcknowledge.java

次のJavaコード例では、順不同確認モードの同期コンシューマーを作成しています。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
```

```
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

/**
 * An example class to demonstrate the behavior of UNORDERED_ACKNOWLEDGE mode for
 received messages. This example
 * complements the example given in {@link SyncMessageReceiverClientAcknowledge} for
 CLIENT_ACKNOWLEDGE mode.
 *
 * First, a session, a message producer, and a message consumer are created. Then, two
 messages are sent. Next, two messages
 * are received but only the second one is acknowledged. After waiting for the
 visibility time out period, an attempt to
 * receive another message is made. It's shown that the first message received in the
 prior attempt is returned again
 * for the second attempt. In UNORDERED_ACKNOWLEDGE mode, all the messages must be
 explicitly acknowledged no matter what
 * the order they're received.
 *
 * This ISN'T the behavior for CLIENT_ACKNOWLEDGE mode. Please see {@link
 SyncMessageReceiverClientAcknowledge}
 * for an example.
 */
public class SyncMessageReceiverUnorderedAcknowledge {

    // Visibility time-out for the queue. It must match to the one set for the queue
 for this example to work.
    private static final long TIME_OUT_SECONDS = 1;

    public static void main(String args[]) throws JMSException, InterruptedException {
        // Create the configuration for the example
        ExampleConfiguration config =
 ExampleConfiguration.parseConfig("SyncMessageReceiverUnorderedAcknowledge", args);

        // Setup logging for the example
        ExampleCommon.setupLogging();

        // Create the connection factory based on the config
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
```

```
                new ProviderConfiguration(),
                AmazonSQSClientBuilder.standard()
                        .withRegion(config.getRegion().getName())
                        .withCredentials(config.getCredentialsProvider())
                );

        // Create the connection
        SQSConnection connection = connectionFactory.createConnection();

        // Create the queue if needed
        ExampleCommon.ensureQueueExists(connection, config.getQueueName());

        // Create the session  with unordered acknowledge mode
        Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

        // Create the producer and consume
        MessageProducer producer =
session.createProducer(session.createQueue(config.getQueueName()));
        MessageConsumer consumer =
session.createConsumer(session.createQueue(config.getQueueName()));

        // Open the connection
        connection.start();

        // Send two text messages
        sendMessage(producer, session, "Message 1");
        sendMessage(producer, session, "Message 2");

        // Receive a message and don't acknowledge it
        receiveMessage(consumer, false);

        // Receive another message and acknowledge it
        receiveMessage(consumer, true);

        // Wait for the visibility time out, so that unacknowledged messages reappear
in the queue
        System.out.println("Waiting for visibility timeout...");
        Thread.sleep(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        // Attempt to receive another message and acknowledge it. This results in
receiving the first message since
        // we have acknowledged only the second message. In the UNORDERED_ACKNOWLEDGE
mode, all the messages must
```

```
        // be explicitly acknowledged.
        receiveMessage(consumer, true);

        // Close the connection. This closes the session automatically
        connection.close();
        System.out.println("Connection closed.");
    }

    /**
     * Sends a message through the producer.
     *
     * @param producer Message producer
     * @param session Session
     * @param messageText Text for the message to be sent
     * @throws JMSException
     */
    private static void sendMessage(MessageProducer producer, Session session, String
 messageText) throws JMSException {
        // Create a text message and send it
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives a message through the consumer synchronously with the default timeout
 (TIME_OUT_SECONDS).
     * If a message is received, the message is printed. If no message is received,
 "Queue is empty!" is
     * printed.
     *
     * @param consumer Message consumer
     * @param acknowledge If true and a message is received, the received message is
 acknowledged.
     * @throws JMSException
     */
    private static void receiveMessage(MessageConsumer consumer, boolean acknowledge)
 throws JMSException {
        // Receive a message
        Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(TIME_OUT_SECONDS));

        if (message == null) {
            System.out.println("Queue is empty!");
        } else {
```

```
            // Since this queue has only text messages, cast the message object and
 print the text
            System.out.println("Received: " + ((TextMessage) message).getText());

            // Acknowledge the message if asked
            if (acknowledge) message.acknowledge();
        }
    }
}
```

SpringExampleConfiguration.xml

次のXMLコード例は、SpringExample.javaのBean構成ファイルです。

```
<!--
    Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.

    Licensed under the Apache License, Version 2.0 (the "License").
    You may not use this file except in compliance with the License.
    A copy of the License is located at

    https://aws.amazon.com/apache2.0

    or in the "license" file accompanying this file. This file is distributed
    on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
    express or implied. See the License for the specific language governing
    permissions and limitations under the License.
-->

<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/util http://www.springframework.org/
schema/util/spring-util-3.0.xsd
    ">
```

```
    <bean id="CredentialsProviderBean"
 class="com.amazonaws.auth.DefaultAWSCredentialsProviderChain"/>

    <bean id="ClientBuilder" class="com.amazonaws.services.sqs.AmazonSQSClientBuilder"
 factory-method="standard">
        <property name="region" value="us-east-2"/>
        <property name="credentials" ref="CredentialsProviderBean"/>
    </bean>

    <bean id="ProviderConfiguration"
 class="com.amazon.sqs.javamessaging.ProviderConfiguration">
        <property name="numberOfMessagesToPrefetch" value="5"/>
    </bean>

    <bean id="ConnectionFactory"
 class="com.amazon.sqs.javamessaging.SQSConnectionFactory">
        <constructor-arg ref="ProviderConfiguration" />
        <constructor-arg ref="ClientBuilder" />
    </bean>

    <bean id="Connection" class="javax.jms.Connection"
        factory-bean="ConnectionFactory"
        factory-method="createConnection"
        init-method="start"
        destroy-method="close" />

    <bean id="QueueName" class="java.lang.String">
        <constructor-arg value="SQSJMSClientExampleQueue"/>
    </bean>
</beans>
```

SpringExample.java

次のJavaコード例では、Bean構成ファイルを使用してオブジェクトを初期化しています。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
```

```
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class SpringExample {
    public static void main(String args[]) throws JMSException {
        if( args.length != 1 || !args[0].endsWith(".xml")) {
            System.err.println( "Usage: " + SpringExample.class.getName() + " <spring
 config.xml>" );
            System.exit(1);
        }

        File springFile = new File( args[0] );
        if( !springFile.exists() || !springFile.canRead() ) {
            System.err.println( "File " + args[0] + " doesn't exist or isn't
 readable.");
            System.exit(2);
        }

        ExampleCommon.setupLogging();

        FileSystemXmlApplicationContext context =
            new FileSystemXmlApplicationContext( "file://" +
 springFile.getAbsolutePath() );

        Connection connection;
        try {
            connection = context.getBean(Connection.class);
        } catch( NoSuchBeanDefinitionException e ) {
            System.err.println( "Can't find the JMS connection to use: " +
 e.getMessage() );
            System.exit(3);
            return;
        }

        String queueName;
        try {
            queueName = context.getBean("QueueName", String.class);
        } catch( NoSuchBeanDefinitionException e ) {
```

```
            System.err.println( "Can't find the name of the queue to use: " +
 e.getMessage() );
            System.exit(3);
            return;
        }

        if( connection instanceof SQSConnection ) {
            ExampleCommon.ensureQueueExists( (SQSConnection) connection, queueName );
        }

        // Create the session
        Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
        MessageConsumer consumer =
 session.createConsumer( session.createQueue( queueName) );

        receiveMessages(session, consumer);

        // The context can be setup to close the connection for us
        context.close();
        System.out.println( "Context closed" );
    }

    private static void receiveMessages( Session session, MessageConsumer consumer ) {
        try {
            while( true ) {
                System.out.println( "Waiting for messages");
                // Wait 1 minute for a message
                Message message = consumer.receive(TimeUnit.MINUTES.toMillis(1));
                if( message == null ) {
                    System.out.println( "Shutting down after 1 minute of silence" );
                    break;
                }
                ExampleCommon.handleMessage(message);
                message.acknowledge();
                System.out.println( "Acknowledged message" );
            }
        } catch (JMSException e) {
            System.err.println( "Error receiving from SQS: " + e.getMessage() );
            e.printStackTrace();
        }
    }
}
```

ExampleCommon.java

次のJavaコード例では、Amazon SQSキューが存在するかどうかを確認し、存在しない場合はキューを作成します。このコード例にはログ記録コードの例も含まれています。

```
/*
 * Copyright 2010-2024 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *  https://aws.amazon.com/apache2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 *
 */

public class ExampleCommon {
    /**
     * A utility function to check the queue exists and create it if needed. For most
     * use cases this is usually done by an administrator before the application is
 run.
     */
    public static void ensureQueueExists(SQSConnection connection, String queueName)
 throws JMSException {
        AmazonSQSMessagingClientWrapper client =
 connection.getWrappedAmazonSQSClient();

        /**
         * In most cases, you can do this with just a createQueue call, but
 GetQueueUrl
         * (called by queueExists) is a faster operation for the common case where the
 queue
         * already exists. Also many users and roles have permission to call
 GetQueueUrl
         * but don't have permission to call CreateQueue.
         */
        if( !client.queueExists(queueName) ) {
            client.createQueue( queueName );
```

```
        }
    }

    public static void setupLogging() {
        // Setup logging
        BasicConfigurator.configure();
        Logger.getRootLogger().setLevel(Level.WARN);
    }

    public static void handleMessage(Message message) throws JMSException {
        System.out.println( "Got message " + message.getJMSMessageID() );
        System.out.println( "Content: ");
        if( message instanceof TextMessage ) {
            TextMessage txtMessage = ( TextMessage ) message;
            System.out.println( "\t" + txtMessage.getText() );
        } else if( message instanceof BytesMessage ){
            BytesMessage byteMessage = ( BytesMessage ) message;
            // Assume the length fits in an int - SQS only supports sizes up to 256k so
 that
            // should be true
            byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
            byteMessage.readBytes(bytes);
            System.out.println( "\t" +  Base64.encodeAsString( bytes ) );
        } else if( message instanceof ObjectMessage ) {
            ObjectMessage objMessage = (ObjectMessage) message;
            System.out.println( "\t" + objMessage.getObject() );
        }
    }
}
```

Amazon SQS でサポートされている JMS 1.1 実装

Amazon SQS Java Messaging Library は、以下の JMS 1.1 実装をサポートしています。Amazon SQS Java Messaging Library でサポートされている機能と性能の詳細については、「Amazon SQS に関するよくある質問」を参照してください。

サポートされている共通インターフェース

- `Connection`
- `ConnectionFactory`
- `Destination`

- `Session`
- `MessageConsumer`
- `MessageProducer`

サポートされているメッセージタイプ

- `ByteMessage`
- `ObjectMessage`
- `TextMessage`

サポートされているメッセージ確認モード

- `AUTO_ACKNOWLEDGE`
- `CLIENT_ACKNOWLEDGE`
- `DUPS_OK_ACKNOWLEDGE`
- `UNORDERED_ACKNOWLEDGE`

> **Note**
>
> `UNORDERED_ACKNOWLEDGE`モードはJMS1.1仕様パートには含まれていません。このモードにより、JMSクライアントによるメッセージの明示的な確認を Amazon SQS が許可できるようになります。

JMS定義ヘッダーと予約プロパティ

メッセージの送信用

メッセージを送信する場合は、各メッセージに以下のヘッダーおよびプロパティを設定できます:

- `JMSXGroupID` (FIFOキューの場合は必須で、スタンドダードキューには許可されません)
- `JMS_SQS_DeduplicationId`(FIFO キューではオプション、標準キューでは許可されません)

メッセージを送信すると、Amazon SQSにより各メッセージに以下のヘッダーおよびプロパティが設定されます:

- `JMSMessageID`
- `JMS_SQS_SequenceNumber`(FIFO キューの場合のみ)

メッセージの受信用

メッセージを受信すると、Amazon SQSにより各メッセージに以下のヘッダーおよびプロパティが設定されます:

- `JMSDestination`
- `JMSMessageID`
- `JMSRedelivered`
- `JMSXDeliveryCount`
- `JMSXGroupID`(FIFO キューの場合のみ)
- `JMS_SQS_DeduplicationId`(FIFO キューの場合のみ)
- `JMS_SQS_SequenceNumber`(FIFO キューの場合のみ)

Amazon SQS のチュートリアル

このトピックでは、Amazon SQS の機能と機能を調べるためのチュートリアルを提供します。

チュートリアル

- を使用した Amazon SQS キューの作成 AWS CloudFormation
- チュートリアル:Amazon Virtual Private Cloud から Amazon SQSキューにメッセージを送信する

を使用した Amazon SQS キューの作成 AWS CloudFormation

AWS CloudFormation コンソールを JSON または YAML テンプレートとともに使用して、Amazon SQS キューを作成します。詳細については、「 AWS CloudFormation ユーザーガイド」の「テンプレートとリソースの使用 AWS CloudFormation」を参照してください。 `AWS::SQS::Queue`

AWS CloudFormation を使用して Amazon SQS キューを作成するには。

1. 次のJSON コードを`MyQueue.json`という名前のファイルにコピーします。標準キューを作成するには、`FifoQueue` および`ContentBasedDeduplication`プロパティを省略します。コンテンツベースの重複排除の詳細については、「Amazon SQS の 1 回のみ処理」を参照してください。

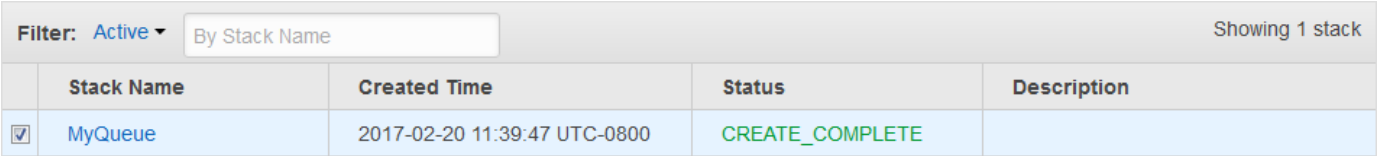
   > **Note**
   >
   > FIFOキューの名前は`.fifo`のサフィックスで終わる必要があります。

```
{
   "AWSTemplateFormatVersion": "2010-09-09",
   "Resources": {
      "MyQueue": {
         "Properties": {
            "QueueName": "MyQueue.fifo",
            "FifoQueue": true,
            "ContentBasedDeduplication": true
             },
         "Type": "AWS::SQS::Queue"
         }
      },
```

```
    "Outputs": {
       "QueueName": {
          "Description": "The name of the queue",
          "Value": {
             "Fn::GetAtt": [
                "MyQueue",
                "QueueName"
             ]
          }
       },
       "QueueURL": {
          "Description": "The URL of the queue",
          "Value": {
             "Ref": "MyQueue"
          }
       },
       "QueueARN": {
          "Description": "The ARN of the queue",
          "Value": {
             "Fn::GetAtt": [
                "MyQueue",
                "Arn"
             ]
          }
       }
    }
}
```

2. AWS CloudFormation コンソール にサインインし、続いて [スタックの作成] を選択します。
3. [Specify Template] パネルで、[Upload a template file]、`MyQueue.json` ファイル、[次へ] の順に選択します。
4. {詳細を指定する} ページで、[`MyQueue`スタック名] に 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。

   AWS CloudFormation は`MyQueue`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、 AWS CloudFormation に [CREATE_COMPLETE] ステータスが表示されます。



7. (オプション) キューの名前、URL、ARN を表示するには、スタックの名前を選択し、次のページで [出力] セクションを展開します。

チュートリアル:Amazon Virtual Private Cloud から Amazon SQS キューにメッセージを送信する

このチュートリアルでは、安全なプライベートネットワーク経由で Amazon SQS キューにメッセージを送信する方法を示します。ネットワークには以下が含まれます。

- Amazon EC2 インスタンスを含む VPC。
- Amazon EC2 インスタンスがパブリックインターネットを使用せずに Amazon SQS に接続できるようにするインターフェイス VPC エンドポイント。

完全プライベートネットワークでも、Amazon EC2 インスタンスに接続し、Amazon SQS キューにメッセージを送信できます。詳細については、「Amazon SQSのAmazon Virtual Private Cloud エンドポイント」を参照してください。

Important

- Amazon Virtual Private Cloud は HTTPS Amazon SQSエンドポイントでのみ使用できます。
- Amazon VPC からメッセージを送信するように Amazon SQS を設定する場合は、プライベート DNS を有効にし、デュアルスタックエンドポイント`sqs.us-east-2.api.aws`の`sqs.us-east-2.amazonaws.com`または 形式でエンドポイントを指定する必要があります。
- Amazon SQS は、 エンドポイントサービスを使用した PrivateLink を介した FIPS `com.amazonaws.region.sqs-fips`エンドポイントもサポートしています。形式の FIPS エンドポイントに接続できます`sqs-fips.region.amazonaws.com`。
- Amazon Virtual Private Cloud でデュアルスタックエンドポイントを使用する場合、リクエストは IPv4 と IPv6 を使用して送信されます。

- 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。

ステップ 1: Amazon EC2 キーペアを作成する

キーペアを使用すると、Amazon EC2 インスタンスに接続することができます。これは、ログイン情報を暗号化す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。

1. Amazon EC2 コンソールにサインインします。
2. ナビゲーションメニューの [ネットワーク & セキュリティ] で、[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. [キーペア作成] ダイアログボックスの [キーペア名] に SQS-VPCE-Tutorial-Key-Pair を入力し、[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SQS-VPCE-Tutorial-Key-Pair.pemが自動的にダウンロードされます。

   > **Important**
   > このファイルを安全な場所に保存します。EC2は、2回目に同じキーペアに対して.pemファイルを生成しません。

6. SSHクライアントに EC2 インスタンスへの接続を許可するには、相手のユーザーのみが読み取り権限を持つことができるように、プライベートキーファイルのアクセス許可を設定します。例:

   ```
   chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
   ```

ステップ 2: AWS リソースを作成する

必要なインフラストラクチャを設定するには、 AWS CloudFormation template を使用する必要があります。これは、Amazon EC2 インスタンスや Amazon SQS キューなどの AWS リソースで構成されるスタックを作成するための設計図です。

このチュートリアルのスタックには、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サブネット内に起動されたAmazon EC2インスタンス
- Amazon SQSキュー

1. GitHub SQS-VPCE-Tutorial-CloudFormation.yamlから という名前の AWS CloudFormation テンプレートをダウンロードします。
2. AWS 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、SQS-VPCE-SQS-Tutorial-CloudFormation.yamlファイルを選択して [次へ] を選択します。
5. [詳細の指定] ページで、以下の作業を行います。
   a. [スタック名] に「SQS-VPCE-Tutorial-Stack」と入力します。
   b. [KeyName] には、[SQS-VPCE-Tutorial-Key-Pair] を選択します。
   c. [次へ] を選択します。
6. [Options(オプション)] ページで、[Next(次へ)] を選択します。
7. 「確認」ページの「機能」セクションで、 AWS CloudFormation がカスタム名で IAM リソースを作成する可能性があることを承認します」を選択し、「作成」を選択します。

AWS CloudFormation は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、 AWS CloudFormation に [CREATE_COMPLETE]ステータスが表示されます。

ステップ 3: EC2 インスタンスがパブリックアクセス可能ではないことを確認する

AWS CloudFormation テンプレートは、VPC SQS-VPCE-Tutorial-EC2-Instanceに という名前の EC2 インスタンスを起動します。このEC2 インスタンスはアウトバウンドトラフィックを許可せず、 Amazon SQS にメッセージを送信することができません。これを確認するには、インスタンスに接続し、パブリックエンドポイントへの接続を試行してからメッセージAmazon SQSを送信してみる必要があります。

1. Amazon EC2 コンソールにサインインします。
2. ナビゲーションメニューで、[インスタンス] の下にある [インスタンス] を選択します。
3. [SQS-VPCE-Tutorial-EC2Instance] を選択します。
4. ec2-203-0-113-0.us-west-2.compute.amazonaws.com など、パブリック DNS の下にホスト名をコピーします。
5. 先に作成したキーペアが格納されているディレクトリから次のコマンドを使用してインスタンスに接続します、例:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

6. パブリックエンドポイントに接続を試みます、例:

```
ping amazon.com
```

接続の試行は予期したとおりに失敗します。

7. Amazon SQSコンソールにサインインします。
8. キューのリストから、VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK など、 AWS CloudFormation テンプレートによって作成されたキューを選択します。
9. 詳細 テーブルで、URL をコピーします、例、 https://sqs.us-east-2.amazonaws.com/123456789012/。
10. EC2 インスタンスから、次のコマンドを使用して、キューにメッセージを発行してみます、例:

```
aws sqs send-message --region us-east-2 --endpoint-url https://sqs.us-
east-2.amazonaws.com/ --queue-url https://sqs.us-east-2.amazonaws.com/123456789012/
 --message-body "Hello from Amazon SQS."
```

送信の試行は予期したとおりに失敗します。

> **Important**
>
> 後で Amazon SQSのVPC エンドポイントを作成するときに、送信の試行は成功します。

ステップ4: Amazon SQSの Amazon VPC エンドポイントを作成する

VPCを Amazon SQS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後、VPC内のEC2 インスタンスから Amazon SQS API を使用できます。これにより、パブリックインターネットを経由せずに、 AWS ネットワーク内のキューにメッセージを送信できます。

> **Note**
> EC2 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は引き続きアクセスできません。

1. Amazon VPC コンソールにサインインします。
2. 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 [サービス名] で、 Amazon SQS のサービス名を選択します。

   > **Note**
   > サービス名は、現在の AWS リージョンによって異なります。たとえば、米国東部 (オハイオ) にいる場合、サービス名は `com.amazonaws.`us-east-2.sqs になります。

5. [VPC] には、SQS-VPCE-Tutorial-VPC を選択します。
6. [サブネット] には、[サブネット ID] にSQS-VPCE-Tutorial-Subnetを含むサブネットを選択します。
7. [セキュリティグループ] の場合は [セキュリティグループの選択] を選択し、[グループ名] にSQS VPCE Tutorial Security Groupを含むセキュリティグループを選択します。
8. [エンドポイントの作成] を選択します。

   インターフェイス VPC エンドポイントが作成され、その ID が表示されます。例: vpce-0ab1cdef2ghi3j456k。
9. [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。

Amazon VPCがエンドポイントの作成を開始し、{保留中} ステータスが表示されます。プロセスが完了すると、Amazon VPCに {利用可能} ステータスが表示されます。

ステップ5:Amazon SQSキューにメッセージを送信する

これで VPCに Amazon SQSのエンドポイントが含まれたので、EC2 インスタンスに接続して、キューにメッセージを送信できます。

1. EC2インスタンスに再接続します、例:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

2. 次のコマンドを使用して、もう一度キューにメッセージを発行してみます、例:

```
aws sqs send-message --region us-east-2 --endpoint-url https://sqs.us-
east-2.amazonaws.com/ --queue-url https://sqs.us-east-2.amazonaws.com/123456789012/
 --message-body "Hello from Amazon SQS."
```

   送信の試行が成功し、メッセージ本文のMD5ダイジェストとメッセージ ID が表示されます、例:

```
{
 "MD5OfMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
 "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

AWS CloudFormation テンプレートによって作成されたキュー (VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK など) からのメッセージの受信と削除については、「」を参照してください[Amazon SQS でのメッセージの受信と削除]。

リソースの削除の詳細については、以下を参照してください。

- [VPC エンドポイントの削除]のAmazon VPC User Guide
- [Amazon SQSキューの削除]
- 「Amazon EC2 ユーザーガイド」の「[インスタンスの終了]」
- 「Amazon VPC ユーザーガイド」の「[VPC を削除する]」
- AWS CloudFormation ユーザーガイド[の「 AWS CloudFormation コンソールでのスタックの削除]」

- 「Amazon EC2 ユーザーガイド」の「キーペアの削除」

SDK を使用した Amazon SQS のコード例 AWS SDKs

次のコード例は、 AWS Software Development Kit (SDK) で Amazon SQS 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を呼び出す方法を示していますが、コンテキスト内のアクションは、関連するシナリオで確認できます。

「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 SQS 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。

開始方法

Hello Amazon SQS

次のコード例は、Amazon SQS 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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
```

```
    {
        var sqsClient = new AmazonSQSClient();

        Console.WriteLine($"Hello Amazon SQS! Following are some of your
 queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });

        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\tQueue Url: {queue}");
            Console.WriteLine();
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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 sqs)

# Set this project's name.
project("hello_sqs")

# 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_sqs.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_sqs.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ListQueuesRequest.h>
#include <iostream>

/*
 *  A "Hello SQS" starter application that initializes an Amazon Simple Queue
 Service
 *  (Amazon SQS) client and lists the SQS queues in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_sqs'
 *
 */

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::SQS::SQSClient sqsClient(clientConfig);

        Aws::Vector<Aws::String> allQueueUrls;
        Aws::String nextToken; // Next token is used to handle a paginated
 response.
        do {
            Aws::SQS::Model::ListQueuesRequest request;

            Aws::SQS::Model::ListQueuesOutcome outcome =
 sqsClient.ListQueues(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::String> &pageOfQueueUrls =
 outcome.GetResult().GetQueueUrls();
                if (!pageOfQueueUrls.empty()) {
                    allQueueUrls.insert(allQueueUrls.cend(),
 pageOfQueueUrls.cbegin(),
```

```
                                        pageOfQueueUrls.cend());
                }
            }
            else {
                std::cerr << "Error with SQS::ListQueues. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
            nextToken = outcome.GetResult().GetNextToken();
        } while (!nextToken.empty());


        std::cout << "Hello Amazon SQS! You have " << allQueueUrls.size() << "
 queue"
                  << (allQueueUrls.size() == 1 ? "" : "s") << " in your account."
                  << std::endl;

        if (!allQueueUrls.empty()) {
            std::cout << "Here are your queue URLs." << std::endl;
            for (const Aws::String &queueUrl: allQueueUrls) {
                std::cout << "  * " << queueUrl << 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「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues 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
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
```

```
    log.Printf("Couldn't get queues. Here's why: %v\n", err)
    break
   } else {
    queueUrls = append(queueUrls, output.QueueUrls...)
   }
  }
  if len(queueUrls) == 0 {
   fmt.Println("You don't have any queues!")
  } else {
   for _, queueUrl := range queueUrls {
    fmt.Printf("\t%v\n", queueUrl)
   }
  }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.SqsException;
import software.amazon.awssdk.services.sqs.paginators.ListQueuesIterable;

/**
 * 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 HelloSQS {
    public static void main(String[] args) {
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listQueues(sqsClient);
        sqsClient.close();
    }

    public static void listQueues(SqsClient sqsClient) {
        try {
            ListQueuesIterable listQueues = sqsClient.listQueuesPaginator();
            listQueues.stream()
                    .flatMap(r -> r.queueUrls().stream())
                    .forEach(content -> System.out.println(" Queue URL: " +
 content.toLowerCase()));

        } catch (SqsException 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「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS クライアントを初期化し、キューを一覧表示します。

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = 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 SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your
 account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
package com.kotlin.sqs

import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.paginators.listQueuesPaginated
import kotlinx.coroutines.flow.transform

suspend fun main() {
    listTopicsPag()
}

suspend fun listTopicsPag() {
    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient
            .listQueuesPaginated { }
            .transform { it.queueUrls?.forEach { queue -> emit(queue) } }
            .collect { queue ->
                println("The Queue URL is $queue")
            }
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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: "sqs-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/awslabs/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: "sqs-basics",
            dependencies: [
                .product(name: "AWSSQS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
```

```
            path: "Sources")

    ]
)
```

Swift ソースコード `entry.swift`。

```
import ArgumentParser
import AWSClientRuntime
import AWSSQS
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: "sqs-basics",
        abstract: """
        This example shows how to list all of your available Amazon SQS queues.
        """,
        discussion: """
        """
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        var queues: [String] = []
        let outputPages = sqsClient.listQueuesPaginated(
            input: ListQueuesInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let urls = output.queueUrls else {
                print("No queues found.")
                return
            }
```

```
            // Iterate over the queue URLs listed on this page, adding them
            // to the `queues` array.

            for queueUrl in urls {
                queues.append(queueUrl)
            }
        }

        print("You have \(queues.count) queues:")
        for queue in queues {
            print("   \(queue)")
        }
    }
}

/// 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ListQueues」を参照してください。

コードの例

- SDK を使用した Amazon SQS の基本的な例 AWS SDKs
  - Hello Amazon SQS
  - SDK を使用した Amazon SQS のアクション AWS SDKs
    - CLI で AddPermission を使用する

- AWS SDK または CLI ChangeMessageVisibilityで を使用する
- CLI で ChangeMessageVisibilityBatch を使用する
- AWS SDK または CLI CreateQueueで を使用する
- AWS SDK または CLI DeleteMessageで を使用する
- AWS SDK または CLI DeleteMessageBatchで を使用する
- AWS SDK または CLI DeleteQueueで を使用する
- AWS SDK または CLI GetQueueAttributesで を使用する
- AWS SDK または CLI GetQueueUrlで を使用する
- CLI で ListDeadLetterSourceQueues を使用する
- AWS SDK または CLI ListQueuesで を使用する
- CLI で PurgeQueue を使用する
- AWS SDK または CLI ReceiveMessageで を使用する
- CLI で RemovePermission を使用する
- AWS SDK または CLI SendMessageで を使用する
- AWS SDK または CLI SendMessageBatchで を使用する
- AWS SDK または CLI SetQueueAttributesで を使用する

- SDK を使用した Amazon SQS のシナリオ AWS SDKs
  - Amazon SQS を使用してメッセージを送受信するウェブアプリケーションを作成する
  - ステップ関数でメッセンジャーアプリケーションを作成する
  - 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 で Amazon S3 を使用して大規模な Amazon SQS Amazon S3 メッセージを管理する
  - AWS SDK を使用して Amazon S3 イベント通知を受信して処理する
  - 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
  - AWS SDK を使用して Amazon SQS でメッセージのバッチを送受信する
  - Message AWS Processing Framework for .NET を使用して Amazon SQS メッセージを公開および受信する
  - Amazon SQS Java Messaging Library を使用して Amazon SQS の Java Message Service (JMS) インターフェイスを操作する
  - AWS SDK を使用してキュータグと Amazon SQS を操作する

- Amazon SQS のサーバーレスの例
  - Amazon SQS トリガーから Lambda 関数を呼び出す
  - Amazon SQS トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

SDK を使用した Amazon SQS の基本的な例 AWS SDKs

以下のコード例は、 AWS SDK で Amazon Simple Queue Service の基本的な機能を使用する方法を示しています。

例

- Hello Amazon SQS
- SDK を使用した Amazon SQS のアクション AWS SDKs
  - CLI で AddPermission を使用する
  - AWS SDK または CLI ChangeMessageVisibilityで を使用する
  - CLI で ChangeMessageVisibilityBatch を使用する
  - AWS SDK または CLI CreateQueueで を使用する
  - AWS SDK または CLI DeleteMessageで を使用する
  - AWS SDK または CLI DeleteMessageBatchで を使用する
  - AWS SDK または CLI DeleteQueueで を使用する
  - AWS SDK または CLI GetQueueAttributesで を使用する
  - AWS SDK または CLI GetQueueUrlで を使用する
  - CLI で ListDeadLetterSourceQueues を使用する
  - AWS SDK または CLI ListQueuesで を使用する
  - CLI で PurgeQueue を使用する
  - AWS SDK または CLI ReceiveMessageで を使用する
  - CLI で RemovePermission を使用する
  - AWS SDK または CLI SendMessageで を使用する
  - AWS SDK または CLI SendMessageBatchで を使用する
  - AWS SDK または CLI SetQueueAttributesで を使用する

Hello Amazon SQS

次のコード例は、Amazon SQS 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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSActions;

public static class HelloSQS
{
    static async Task Main(string[] args)
    {
        var sqsClient = new AmazonSQSClient();

        Console.WriteLine($"Hello Amazon SQS! Following are some of your
 queues:");
        Console.WriteLine();

        // You can use await and any of the async methods to get a response.
        // Let's get the first five queues.
        var response = await sqsClient.ListQueuesAsync(
            new ListQueuesRequest()
            {
                MaxResults = 5
            });

        foreach (var queue in response.QueueUrls)
        {
            Console.WriteLine($"\tQueue Url: {queue}");
            Console.WriteLine();
```

```
        }
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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 sqs)

# Set this project's name.
project("hello_sqs")

# 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_sqs.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_sqs.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/sqs/SQSClient.h>
#include <aws/sqs/model/ListQueuesRequest.h>
#include <iostream>

/*
 *  A "Hello SQS" starter application that initializes an Amazon Simple Queue
 Service
 *  (Amazon SQS) client and lists the SQS queues in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_sqs'
 *
 */

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::SQS::SQSClient sqsClient(clientConfig);

        Aws::Vector<Aws::String> allQueueUrls;
        Aws::String nextToken; // Next token is used to handle a paginated
 response.
        do {
            Aws::SQS::Model::ListQueuesRequest request;

            Aws::SQS::Model::ListQueuesOutcome outcome =
 sqsClient.ListQueues(request);

            if (outcome.IsSuccess()) {
                const Aws::Vector<Aws::String> &pageOfQueueUrls =
 outcome.GetResult().GetQueueUrls();
                if (!pageOfQueueUrls.empty()) {
                    allQueueUrls.insert(allQueueUrls.cend(),
 pageOfQueueUrls.cbegin(),
                                        pageOfQueueUrls.cend());
                }
            }
            else {
                std::cerr << "Error with SQS::ListQueues. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
            nextToken = outcome.GetResult().GetNextToken();
        } while (!nextToken.empty());


        std::cout << "Hello Amazon SQS! You have " << allQueueUrls.size() << "
 queue"
                  << (allQueueUrls.size() == 1 ? "" : "s") << " in your account."
                  << std::endl;

        if (!allQueueUrls.empty()) {
```

```
            std::cout << "Here are your queue URLs." << std::endl;
            for (const Aws::String &queueUrl: allQueueUrls) {
                std::cout << "  * " << queueUrl << 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「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues 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
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.SqsException;
import software.amazon.awssdk.services.sqs.paginators.ListQueuesIterable;

/**
 * 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 HelloSQS {
    public static void main(String[] args) {
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listQueues(sqsClient);
        sqsClient.close();
    }

    public static void listQueues(SqsClient sqsClient) {
        try {
            ListQueuesIterable listQueues = sqsClient.listQueuesPaginator();
            listQueues.stream()
                    .flatMap(r -> r.queueUrls().stream())
                    .forEach(content -> System.out.println(" Queue URL: " +
 content.toLowerCase()));
```

```
        } catch (SqsException 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「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS クライアントを初期化し、キューを一覧表示します。

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";

export const helloSqs = 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 SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
 `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }
```

```
  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your
 account.`,
  );
  console.log(queues.map((t) => `  * ${t}`).join("\n"));
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
package com.kotlin.sqs

import aws.sdk.kotlin.services.sqs.SqsClient
import aws.sdk.kotlin.services.sqs.paginators.listQueuesPaginated
import kotlinx.coroutines.flow.transform

suspend fun main() {
    listTopicsPag()
}

suspend fun listTopicsPag() {
    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient
            .listQueuesPaginated { }
            .transform { it.queueUrls?.forEach { queue -> emit(queue) } }
            .collect { queue ->
                println("The Queue URL is $queue")
```

```
            }
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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: "sqs-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/awslabs/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: "sqs-basics",
            dependencies: [
                .product(name: "AWSSQS", package: "aws-sdk-swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

Swift ソースコード `entry.swift`。

```
import ArgumentParser
import AWSClientRuntime
import AWSSQS
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: "sqs-basics",
        abstract: """
        This example shows how to list all of your available Amazon SQS queues.
        """,
        discussion: """
        """
    )

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        var queues: [String] = []
        let outputPages = sqsClient.listQueuesPaginated(
            input: ListQueuesInput()
```

```
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let urls = output.queueUrls else {
                print("No queues found.")
                return
            }

            // Iterate over the queue URLs listed on this page, adding them
            // to the `queues` array.

            for queueUrl in urls {
                queues.append(queueUrl)
            }
        }

        print("You have \(queues.count) queues:")
        for queue in queues {
            print("   \(queue)")
        }
    }
}

/// 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ListQueues」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 SQS 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 SQS のアクション AWS SDKs

次のコード例は、 AWS SDKs で個々の Amazon SQS 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。

これらの抜粋は Amazon SQS 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 SQS 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。詳細なリストについては、「Amazon Simple Queue Service API リファレンス」を参照してください。

例

- CLI で AddPermission を使用する
- AWS SDK または CLI ChangeMessageVisibilityで を使用する
- CLI で ChangeMessageVisibilityBatch を使用する
- AWS SDK または CLI CreateQueueで を使用する
- AWS SDK または CLI DeleteMessageで を使用する
- AWS SDK または CLI DeleteMessageBatchで を使用する
- AWS SDK または CLI DeleteQueueで を使用する
- AWS SDK または CLI GetQueueAttributesで を使用する
- AWS SDK または CLI GetQueueUrlで を使用する
- CLI で ListDeadLetterSourceQueues を使用する
- AWS SDK または CLI ListQueuesで を使用する
- CLI で PurgeQueue を使用する
- AWS SDK または CLI ReceiveMessageで を使用する
- CLI で RemovePermission を使用する
- AWS SDK または CLI SendMessageで を使用する
- AWS SDK または CLI SendMessageBatchで を使用する
- AWS SDK または CLI SetQueueAttributesで を使用する

CLI で **AddPermission** を使用する

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

CLI

AWS CLI

キューにアクセス許可を追加するには

この例では、指定された AWS アカウントが指定されたキューにメッセージを送信できるようにします。

コマンド:

```
aws sqs add-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue --aws-account-ids 12345EXAMPLE --actions SendMessage
```

出力:

```
None.
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定された が指定されたキューからメッセージを送信 AWS アカウント することを許可します。

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 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AddPermission」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定された が指定されたキューからメッセージを送信 AWS アカウント することを許可します。

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 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AddPermission」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 SQS 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 **ChangeMessageVisibility**で を使用する

次のサンプルコードは、ChangeMessageVisibility 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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! Changes the visibility timeout of a message in an Amazon Simple Queue Service
//! (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageReceiptHandle: A message receipt handle.
  \param visibilityTimeoutSeconds: Visibility timeout in seconds.
  \param clientConfiguration: AWS client configuration.
```

```
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::changeMessageVisibility(
        const Aws::String &queue_url,
        const Aws::String &messageReceiptHandle,
        int visibilityTimeoutSeconds,
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ChangeMessageVisibilityRequest request;
    request.SetQueueUrl(queue_url);
    request.SetReceiptHandle(messageReceiptHandle);
    request.SetVisibilityTimeout(visibilityTimeoutSeconds);

    auto outcome = sqsClient.ChangeMessageVisibility(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully changed visibility of message " <<
                  messageReceiptHandle << " from queue " << queue_url <<
 std::endl;
    }
    else {
        std::cout << "Error changing visibility of message from queue "
                  << queue_url << ": " <<
                  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「ChangeMessageVisibility」を参照してください。

CLI

AWS CLI

メッセージのタイムアウトの可視性を変更するには

この例は、指定されたメッセージのタイムアウトの可視性を 10 時間 (10 時間 × 60 分 × 60 秒) に変更します。

コマンド:

```
aws sqs change-message-visibility --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --
visibility-timeout 36000
```

出力:

```
None.
```

- API の詳細については、AWS CLI コマンドリファレンスの「ChangeMessageVisibility」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS メッセージを受信し、可視性タイムアウトを変更します。

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 1,
```

```
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ChangeMessageVisibility」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS メッセージを受信し、可視性タイムアウトを変更します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region to us-west-2
AWS.config.update({ region: "us-west-2" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "https://sqs.REGION.amazonaws.com/ACCOUNT-ID/QUEUE-NAME";

var params = {
  AttributeNames: ["SentTimestamp"],
```

```
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else {
    // Make sure we have a message
    if (data.Messages != null) {
      var visibilityParams = {
        QueueUrl: queueURL,
        ReceiptHandle: data.Messages[0].ReceiptHandle,
        VisibilityTimeout: 20, // 20 second timeout
      };
      sqs.changeMessageVisibility(visibilityParams, function (err, data) {
        if (err) {
          console.log("Delete Error", err);
        } else {
          console.log("Timeout Changed", data);
        }
      });
    } else {
      console.log("No messages to change");
    }
  }
});
```

- 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「ChangeMessageVisibility」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: この例では、指定したキュー内の指定した受信ハンドルを持つメッセージの可視性タイムアウトを 10 時間 (10 時間 × 60 分 × 60 秒 = 36,000 秒) に変更します。

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- 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ChangeMessageVisibility」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定したキュー内の指定した受信ハンドルを持つメッセージの可視性タイムアウトを 10 時間 (10 時間 × 60 分 × 60 秒 = 36,000 秒) に変更します。

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- 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ChangeMessageVisibility」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
require 'aws-sdk-sqs' # v2: require 'aws-sdk'
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
sqs = Aws::SQS::Client.new(region: 'us-west-2')

begin
  queue_name = 'my-queue'
  queue_url = sqs.get_queue_url(queue_name: queue_name).queue_url

  # Receive up to 10 messages
  receive_message_result_before = sqs.receive_message({
```

```
                                                                  queue_url: queue_url,
                                                                  max_number_of_messages:
 10
                                                              })

  puts "Before attempting to change message visibility timeout: received
 #{receive_message_result_before.messages.count} message(s)."

  receive_message_result_before.messages.each do |message|
    sqs.change_message_visibility({
                                      queue_url: queue_url,
                                      receipt_handle: message.receipt_handle,
                                      visibility_timeout: 30 # This message will
 not be visible for 30 seconds after first receipt.
                                    })
  end

  # Try to retrieve the original messages after setting their visibility timeout.
  receive_message_result_after = sqs.receive_message({
                                                        queue_url: queue_url,
                                                        max_number_of_messages: 10
                                                      })

  puts "\nAfter attempting to change message visibility timeout: received
 #{receive_message_result_after.messages.count} message(s)."
rescue Aws::SQS::Errors::NonExistentQueue
  puts "Cannot receive messages for a queue named '#{queue_name}', as it does not
 exist."
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ChangeMessageVisibility」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 SQS 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。

CLI で **ChangeMessageVisibilityBatch** を使用する

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

CLI

AWS CLI

複数のメッセージの可視性タイムアウトをバッチとして変更するには

この例では、2 件の指定されたメッセージの可視性タイムアウトを 10 時間 (10 時間 × 60 分 × 60 秒) に変更します。

コマンド:

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

入力ファイル (change-message-visibility-batch.json):

```
[
  {
    "Id": "FirstMessage",
        "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
        "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBkTUH...HifSnw==",
        "VisibilityTimeout": 36000
  }
]
```

出力:

```
{
  "Successful": [
    {
      "Id": "SecondMessage"
    },
    {
      "Id": "FirstMessage"
    }
  ]
}
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定したキュー内の指定した受信ハンドルを持つ 2 つのメッセージの可視性タイムアウトを変更します。最初のメッセージの可視性タイムアウトを 10 時間 (10 時間 × 60 分 × 60 秒 = 36,000 秒) に変更します。2 番目のメッセージの可視性タイムアウトを 5 時間 (5 時間 × 60 分 × 60 秒 = 18,000 秒) に変更します。

```
$changeVisibilityRequest1 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6gl8Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000

$changeVisibilityRequest2 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqww=="
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
 $changeVisibilityRequest2
```

出力:

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- 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ChangeMessageVisibilityBatch」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定したキュー内の指定した受信ハンドルを持つ 2 つのメッセージの可視性タイムアウトを変更します。最初のメッセージの可視性タイムアウトを 10 時間 (10 時間 × 60 分 × 60 秒 = 36,000 秒) に変更します。2 番目のメッセージの可視性タイムアウトを 5 時間 (5 時間 × 60 分 × 60 秒 = 18,000 秒) に変更します。

```
$changeVisibilityRequest1 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest1.Id = "Request1"
$changeVisibilityRequest1.ReceiptHandle = "AQEBd329...v6gl8Q=="
$changeVisibilityRequest1.VisibilityTimeout = 36000

$changeVisibilityRequest2 = New-Object
 Amazon.SQS.Model.ChangeMessageVisibilityBatchRequestEntry
$changeVisibilityRequest2.Id = "Request2"
$changeVisibilityRequest2.ReceiptHandle = "AQEBgGDh...J/Iqww=="
$changeVisibilityRequest2.VisibilityTimeout = 18000

Edit-SQSMessageVisibilityBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $changeVisibilityRequest1,
 $changeVisibilityRequest2
```

出力:

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- 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ChangeMessageVisibilityBatch」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 SQS 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 **CreateQueue**で を使用する

次のサンプルコードは、`CreateQueue` 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
- バッチメッセージを送受信する
- Amazon SQS Java Messaging Library を使用して JMS インターフェイスを操作する

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

特定の名前を持つキューを作成します。

```
    /// <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;
    }
```

Amazon SQS キューを作成して、そこにメッセージを送信します。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon;
    using Amazon.SQS;
    using Amazon.SQS.Model;

    public class CreateSendExample
    {
        // Specify your AWS Region (an example Region is shown).
        private static readonly string QueueName = "Example_Queue";
        private static readonly RegionEndpoint ServiceRegion =
 RegionEndpoint.USWest2;
        private static IAmazonSQS client;

        public static async Task Main()
        {
```

```
            client = new AmazonSQSClient(ServiceRegion);
            var createQueueResponse = await CreateQueue(client, QueueName);

            string queueUrl = createQueueResponse.QueueUrl;

            Dictionary<string, MessageAttributeValue> messageAttributes = new
 Dictionary<string, MessageAttributeValue>
            {
                { "Title",   new MessageAttributeValue { DataType = "String",
 StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
 StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
 StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
 bestseller for week of 12/11/2016.";

            var sendMsgResponse = await SendMessage(client, queueUrl,
 messageBody, messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS
 client, string queueName)
        {
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
```

```
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! Create an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueName: An Amazon SQS queue name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::createQueue(const Aws::String &queueName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::CreateQueueRequest request;
    request.SetQueueName(queueName);

    const Aws::SQS::Model::CreateQueueOutcome outcome =
 sqsClient.CreateQueue(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created queue " << queueName << " with a queue
 URL "
                  << outcome.GetResult().GetQueueUrl() << "." << std::endl;
    }
    else {
        std::cerr << "Error creating queue " << queueName << ": " <<
                  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「CreateQueue」を参照してください。

CLI

AWS CLI

キューを作成するには

この例は、指定された名前のキューを作成して、メッセージの保持期間を 3 日間 (3 日 × 24 時間 × 60 分 × 60 秒) に設定します。またキューのデッドレターキューを、最大受信数 1,000 件のメッセージを含む指定されたキューに設定します。

コマンド:

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

入力ファイル (create-queue.json):

```
{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "MessageRetentionPeriod": "259200"
}
```

出力:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、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.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.ChangeMessageVisibilityRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlResponse;
import software.amazon.awssdk.services.sqs.model.ListQueuesRequest;
import software.amazon.awssdk.services.sqs.model.ListQueuesResponse;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;
```

```
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 SQSExample {
    public static void main(String[] args) {
        String queueName = "queue" + System.currentTimeMillis();
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        // Perform various tasks on the Amazon SQS queue.
        String queueUrl = createQueue(sqsClient, queueName);
        listQueues(sqsClient);
        listQueuesFilter(sqsClient, queueUrl);
        List<Message> messages = receiveMessages(sqsClient, queueUrl);
        sendBatchMessages(sqsClient, queueUrl);
        changeMessages(sqsClient, queueUrl, messages);
        deleteMessages(sqsClient, queueUrl, messages);
        sqsClient.close();
    }

    public static String createQueue(SqsClient sqsClient, String queueName) {
        try {
            System.out.println("\nCreate Queue");

            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 void listQueues(SqsClient sqsClient) {

        System.out.println("\nList Queues");
        String prefix = "que";

        try {
            ListQueuesRequest listQueuesRequest =
ListQueuesRequest.builder().queueNamePrefix(prefix).build();
            ListQueuesResponse listQueuesResponse =
sqsClient.listQueues(listQueuesRequest);
            for (String url : listQueuesResponse.queueUrls()) {
                System.out.println(url);
            }

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

    public static void listQueuesFilter(SqsClient sqsClient, String queueUrl) {
        // List queues with filters
        String namePrefix = "queue";
        ListQueuesRequest filterListRequest = ListQueuesRequest.builder()
                .queueNamePrefix(namePrefix)
                .build();

        ListQueuesResponse listQueuesFilteredResponse =
sqsClient.listQueues(filterListRequest);
        System.out.println("Queue URLs with prefix: " + namePrefix);
        for (String url : listQueuesFilteredResponse.queueUrls()) {
            System.out.println(url);
        }

        System.out.println("\nSend message");
```

```
        try {
            sqsClient.sendMessage(SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody("Hello world!")
                    .delaySeconds(10)
                    .build());

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

    public static void sendBatchMessages(SqsClient sqsClient, String queueUrl) {

        System.out.println("\nSend multiple messages");
        try {
            SendMessageBatchRequest sendMessageBatchRequest =
SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)

.entries(SendMessageBatchRequestEntry.builder().id("id1").messageBody("Hello
from msg 1").build(),

SendMessageBatchRequestEntry.builder().id("id2").messageBody("msg
2").delaySeconds(10)
                                    .build())
                    .build();
            sqsClient.sendMessageBatch(sendMessageBatchRequest);

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

    public static List<Message> receiveMessages(SqsClient sqsClient, String
queueUrl) {

        System.out.println("\nReceive messages");
        try {
            ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
```

```
                    .maxNumberOfMessages(5)
                    .build();
            return sqsClient.receiveMessage(receiveMessageRequest).messages();

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

    public static void changeMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {

        System.out.println("\nChange Message Visibility");
        try {

            for (Message message : messages) {
                ChangeMessageVisibilityRequest req =
ChangeMessageVisibilityRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .visibilityTimeout(100)
                        .build();
                sqsClient.changeMessageVisibility(req);
            }

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

    public static void deleteMessages(SqsClient sqsClient, String queueUrl,
List<Message> messages) {
        System.out.println("\nDelete Messages");

        try {
            for (Message message : messages) {
                DeleteMessageRequest deleteMessageRequest =
DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .build();
```

```
                sqsClient.deleteMessage(deleteMessageRequest);
            }
        } catch (SqsException 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「CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS 標準キューを作成します。

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
```

```
};
```

ロングポーリングでキューを作成します。

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const response = await client.send(
    new CreateQueueCommand({
      QueueName: queueName,
      Attributes: {
        // When the wait time for the ReceiveMessage API action is greater than
 0,
        // long polling is in effect. The maximum long polling wait time is 20
        // seconds. Long polling helps reduce the cost of using Amazon SQS by,
        // eliminating the number of empty responses and false empty responses.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-short-and-long-polling.html
        ReceiveMessageWaitTimeSeconds: "20",
      },
    }),
  );
  console.log(response);
  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「CreateQueue」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS 標準キューを作成します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    DelaySeconds: "60",
    MessageRetentionPeriod: "86400",
  },
};

sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

メッセージが到着するまで待機する Amazon SQS キューを作成します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
  Attributes: {
    ReceiveMessageWaitTimeSeconds: "20",
  },
};
```

```
sqs.createQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 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「CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun createQueue(queueNameVal: String): String {
    println("Create Queue")
    val createQueueRequest =
        CreateQueueRequest {
            queueName = queueNameVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.createQueue(createQueueRequest)
        println("Get queue url")

        val getQueueUrlRequest =
            GetQueueUrlRequest {
                queueName = queueNameVal
            }

        val getQueueUrlResponse = sqsClient.getQueueUrl(getQueueUrlRequest)
```

```
        return getQueueUrlResponse.queueUrl.toString()
    }
}
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定した名前でキューを作成します。

```
New-SQSQueue -QueueName MyQueue
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 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CreateQueue」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定した名前でキューを作成します。

```
New-SQSQueue -QueueName MyQueue
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 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CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def create_queue(name, attributes=None):
    """
    Creates an Amazon SQS queue.

    :param name: The name of the queue. This is part of the URL assigned to the
 queue.
    :param attributes: The attributes of the queue, such as maximum message size
 or
                       whether it's a FIFO queue.
    :return: A Queue object that contains metadata about the queue and that can
 be used
             to perform queue operations like sending and receiving messages.
    """
    if not attributes:
        attributes = {}

    try:
        queue = sqs.create_queue(QueueName=name, Attributes=attributes)
        logger.info("Created queue '%s' with URL=%s", name, queue.url)
    except ClientError as error:
        logger.exception("Couldn't create queue named '%s'.", name)
        raise error
    else:
        return queue
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
# This code example demonstrates how to create a queue in Amazon Simple Queue
 Service (Amazon SQS).

require 'aws-sdk-sqs'

# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_name [String] The name of the queue.
# @return [Boolean] true if the queue was created; otherwise, false.
# @example
#   exit 1 unless queue_created?(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'my-queue'
#   )
def queue_created?(sqs_client, queue_name)
  sqs_client.create_queue(queue_name: queue_name)
  true
rescue StandardError => e
  puts "Error creating queue: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Creating the queue named '#{queue_name}'..."

  if queue_created?(sqs_client, queue_name)
    puts 'Queue created.'
```

```
  else
    puts 'Queue not created.'
  end
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS 標準キューを作成します。

```
    TRY.
        oo_result = lo_sqs->createqueue( iv_queuename = iv_queue_name ).        "
oo_result is returned for testing purposes. "
        MESSAGE 'SQS queue created.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedeldrecently.
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
queue with the same name.' TYPE 'E'.
      CATCH /aws1/cx_sqsqueuenameexists.
        MESSAGE 'A queue with this name already exists.' TYPE 'E'.
    ENDTRY.
```

メッセージが到着するまで待機する Amazon SQS キューを作成します。

```
    TRY.
        DATA lt_attributes TYPE /aws1/cl_sqsqueueattrmap_w=>tt_queueattributemap.
```

```
        DATA ls_attribute TYPE /aws1/
cl_sqsqueueattrmap_w=>ts_queueattributemap_maprow.
        ls_attribute-key = 'ReceiveMessageWaitTimeSeconds'.               " Time
 in seconds for long polling, such as how long the call waits for a message to
 arrive in the queue before returning. "
        ls_attribute-value = NEW /aws1/cl_sqsqueueattrmap_w( iv_value =
 iv_wait_time ).
        INSERT ls_attribute INTO TABLE lt_attributes.
        oo_result = lo_sqs->createqueue(                  " oo_result is returned
 for testing purposes. "
                iv_queuename = iv_queue_name
                it_attributes = lt_attributes ).
        MESSAGE 'SQS queue created.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedeldrecently.
        MESSAGE 'After deleting a queue, wait 60 seconds before creating another
 queue with the same name.' TYPE 'E'.
      CATCH /aws1/cx_sqsqueuenameexists.
        MESSAGE 'A queue with this name already exists.' 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「CreateQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.createQueue(
            input: CreateQueueInput(
```

```
                queueName: queueName
            )
        )

        guard let queueUrl = output.queueUrl else {
            print("No queue URL returned.")
            return
        }
```

- API の詳細については、 AWS SDK for Swift API リファレンスの「CreateQueue」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 SQS 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 **DeleteMessage**で を使用する

次のサンプルコードは、DeleteMessage 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。

- バッチメッセージを送受信する

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューからメッセージを受信し、メッセージを削除します。

```
        public static async Task Main()
        {
```

```
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
 queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await
 client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</
param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
```

```
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };

            var receiveMessageResponse = await
client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! Delete a message from an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageReceiptHandle: A message receipt handle.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::deleteMessage(const Aws::String &queueUrl,
                                const Aws::String &messageReceiptHandle,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::DeleteMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetReceiptHandle(messageReceiptHandle);

    const Aws::SQS::Model::DeleteMessageOutcome outcome =
 sqsClient.DeleteMessage(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted message from queue " << queueUrl
                  << std::endl;
    }
    else {
        std::cerr << "Error deleting message from queue " << queueUrl << ": " <<
                  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「DeleteMessage」を参照してください。

CLI

AWS CLI

単一のメッセージを削除するには

この例は、指定された単一のメッセージを削除します。

コマンド:

```
aws sqs delete-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

出力:

```
None.
```

- API の詳細については、AWS CLI コマンドリファレンスの「DeleteMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        try {
```

```
            for (Message message : messages) {
                DeleteMessageRequest deleteMessageRequest =
 DeleteMessageRequest.builder()
                        .queueUrl(queueUrl)
                        .receiptHandle(message.receiptHandle())
                        .build();
                sqsClient.deleteMessage(deleteMessageRequest);
            }
        } catch (SqsException 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「DeleteMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS メッセージを受信および削除します。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
```

```
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「DeleteMessage」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS メッセージを受信および削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 10,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  VisibilityTimeout: 20,
  WaitTimeSeconds: 0,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Receive Error", err);
  } else if (data.Messages) {
    var deleteParams = {
      QueueUrl: queueURL,
      ReceiptHandle: data.Messages[0].ReceiptHandle,
    };
    sqs.deleteMessage(deleteParams, function (err, data) {
      if (err) {
        console.log("Delete Error", err);
      } else {
        console.log("Message Deleted", data);
      }
```

```
    });
  }
});
```

- 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「DeleteMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.purgeQueue(purgeRequest)
        println("Messages are successfully deleted from $queueUrlVal")
    }
}

suspend fun deleteQueue(queueUrlVal: String) {
    val request =
        DeleteQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
```

```
        println("$queueUrlVal was deleted!")
    }
}
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定した受信ハンドルを持つメッセージを指定したキューから削除します。

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- 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DeleteMessage」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定した受信ハンドルを持つメッセージを指定したキューから削除します。

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- 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DeleteMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def delete_message(message):
    """
    Delete a message from a queue. Clients must delete messages after they
    are received and processed to remove them from the queue.

    :param message: The message to delete. The message's queue URL is contained
 in
                    the message's metadata.
    :return: None
    """
    try:
        message.delete()
        logger.info("Deleted message: %s", message.message_id)
    except ClientError as error:
        logger.exception("Couldn't delete message: %s", message.message_id)
        raise error
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DeleteMessage」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 SQS 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 **DeleteMessageBatch**で を使用する

次のサンプルコードは、DeleteMessageBatch 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。

- S3 イベント通知の処理
- メッセージをキューに発行する
- バッチメッセージを送受信する

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <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();
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteMessageBatch」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

Aws::SQS::SQSClient sqsClient(clientConfiguration);

        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;
            }
```

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

CLI

AWS CLI

複数のメッセージを一括削除するには

この例は、指定されたメッセージを削除します。

コマンド:

```
aws sqs delete-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-batch.json
```

入力ファイル (delete-message-batch.json):

```
[
  {
        "Id": "FirstMessage",
        "ReceiptHandle": "AQEB1mgl...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

出力:

```
{
  "Successful": [
    {
```

```
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「DeleteMessageBatch」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
}
```

```
// 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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「DeleteMessageBatch」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
```

```
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「DeleteMessageBatch」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: この例では、指定した受信ハンドルを持つ 2 つのメッセージを指定したキューから削除します。

```
$deleteMessageRequest1 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
 $deleteMessageRequest2
```

出力:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- 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DeleteMessageBatch」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定した受信ハンドルを持つ 2 つのメッセージを指定したキューから削除します。

```
$deleteMessageRequest1 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest1.Id = "Request1"
$deleteMessageRequest1.ReceiptHandle = "AQEBX2g4...wtJSQg=="

$deleteMessageRequest2 = New-Object
 Amazon.SQS.Model.DeleteMessageBatchRequestEntry
$deleteMessageRequest2.Id = "Request2"
$deleteMessageRequest2.ReceiptHandle = "AQEBqOVY...KTsLYg=="

Remove-SQSMessageBatch -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue -Entry $deleteMessageRequest1,
 $deleteMessageRequest2
```

出力:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- 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 DeleteMessageBatch」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def delete_messages(queue, messages):
    """
    Delete a batch of messages from a queue in a single request.

    :param queue: The queue from which to delete the messages.
```

```
    :param messages: The list of messages to delete.
    :return: The response from SQS that contains the list of successful and
failed
             message deletions.
    """
    try:
        entries = [
            {"Id": str(ind), "ReceiptHandle": msg.receipt_handle}
            for ind, msg in enumerate(messages)
        ]
        response = queue.delete_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info("Deleted %s",
messages[int(msg_meta["Id"])].receipt_handle)
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Could not delete %s",
messages[int(msg_meta["Id"])].receipt_handle
                )
    except ClientError:
        logger.exception("Couldn't delete messages from queue %s", queue)
    else:
        return response
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DeleteMessageBatch」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        // Create the list of message entries.

        var entries: [SQSClientTypes.DeleteMessageBatchRequestEntry] = []
        var messageNumber = 1

        for handle in handles {
            let entry = SQSClientTypes.DeleteMessageBatchRequestEntry(
                id: "\(messageNumber)",
                receiptHandle: handle
            )
            entries.append(entry)
            messageNumber += 1
        }

        // Delete the messages.

        let output = try await sqsClient.deleteMessageBatch(
            input: DeleteMessageBatchInput(
                entries: entries,
                queueUrl: queue
            )
        )

        // Get the lists of failed and successful deletions from the output.

        guard let failedEntries = output.failed else {
            print("Failed deletion list is missing!")
            return
        }
        guard let successfulEntries = output.successful else {
            print("Successful deletion list is missing!")
            return
        }

        // Display a list of the failed deletions along with their
        // corresponding explanation messages.

        if failedEntries.count != 0 {
```

```
            print("Failed deletions:")

            for entry in failedEntries {
                print("Message #\(entry.id ?? "<unknown>") failed:
\(entry.message ?? "<unknown>")")
            }
        } else {
            print("No failed deletions.")
        }

        // Output a list of the message numbers that were successfully deleted.

        if successfulEntries.count != 0 {
            var successes = ""

            for entry in successfulEntries {
                if successes.count == 0 {
                    successes = entry.id ?? "<unknown>"
                } else {
                    successes = "\(successes), \(entry.id ?? "<unknown>")"
                }
            }
            print("Succeeded: ", successes)
        } else {
            print("No successful deletions.")
        }
```

- API の詳細については、 AWS SDK for Swift API リファレンスのDeleteMessageBatch」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 SQS 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 **DeleteQueue**で を使用する

次のサンプルコードは、DeleteQueue 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
- バッチメッセージを送受信する
- Amazon SQS Java Messaging Library を使用して JMS インターフェイスを操作する

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

URL を使用してキューを削除します。

```
/// <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;
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! Delete an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueURL: An Amazon SQS queue URL.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::deleteQueue(const Aws::String &queueURL,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);
    Aws::SQS::Model::DeleteQueueRequest request;
    request.SetQueueUrl(queueURL);

    const Aws::SQS::Model::DeleteQueueOutcome outcome =
 sqsClient.DeleteQueue(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted queue with url " << queueURL <<
                  std::endl;
    }
    else {
        std::cerr << "Error deleting queue " << queueURL << ": " <<
                  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「DeleteQueue」を参照してください。

CLI

AWS CLI

キューを削除するには

この例は、指定されたキューを削除します。

コマンド:

```
aws sqs delete-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

出力:

```
None.
```

- API の詳細については、AWS CLI コマンドリファレンスの「DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
}



// 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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、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.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
```

```
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

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

                Usage:    <queueName>

                Where:
                   queueName - The name of the Amazon SQS queue to delete.

                """;

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

        String queueName = args[0];
        SqsClient sqs = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();

        deleteSQSQueue(sqs, queueName);
        sqs.close();
    }

    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);

        } catch (SqsException 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「DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューを削除します。

```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });

  const response = await client.send(command);
  console.log(response);
  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「DeleteQueue」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューを削除します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.deleteQueue(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
});
```

- 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「DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.purgeQueue(purgeRequest)
        println("Messages are successfully deleted from $queueUrlVal")
    }
}

suspend fun deleteQueue(queueUrlVal: String) {
    val request =
        DeleteQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
    }
}
```

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

PowerShell

Tools for PowerShell V4

例 1: こので例は、指定したキューを削除します。

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- 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 DeleteQueue」を参照してください。

Tools for PowerShell V5

例 1: こので例は、指定したキューを削除します。

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- 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 DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def remove_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「DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
require 'aws-sdk-sqs' # v2: require 'aws-sdk'
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
sqs = Aws::SQS::Client.new(region: 'us-west-2')

sqs.delete_queue(queue_url: URL)
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    TRY.
        lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
        MESSAGE 'SQS queue deleted' TYPE 'I'.
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「DeleteQueue」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        do {
            _ = try await sqsClient.deleteQueue(
                input: DeleteQueueInput(
                    queueUrl: queueUrl
                )
```

```
        )
    } catch _ as AWSSQS.QueueDoesNotExist {
        print("Error: The specified queue doesn't exist.")
        return
    }
```

- API の詳細については、 AWS SDK for Swift API リファレンスのDeleteQueue」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 SQS 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 **GetQueueAttributes**で を使用する

次のサンプルコードは、`GetQueueAttributes` 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。

- S3 イベント通知の処理
- 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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

    Aws::SQS::SQSClient sqsClient(clientConfiguration);

            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 with SQS::GetQueueAttributes. "
                          << outcome.GetError().GetMessage()
                          << std::endl;


            }
```

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

CLI

AWS CLI

キューの属性を取得するには

この例では、指定されたキューの属性をすべて取得します。

コマンド:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

出力:

```
{
  "Attributes": {
    "ApproximateNumberOfMessagesNotVisible": "0",
    "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":1000}",
    "MessageRetentionPeriod": "345600",
    "ApproximateNumberOfMessagesDelayed": "0",
    "MaximumMessageSize": "262144",
    "CreatedTimestamp": "1442426968",
    "ApproximateNumberOfMessages": "0",
    "ReceiveMessageWaitTimeSeconds": "0",
    "DelaySeconds": "0",
    "VisibilityTimeout": "30",
    "LastModifiedTimestamp": "1442426968",
    "QueueArn": "arn:aws:sqs:us-east-1:80398EXAMPLE:MyNewQueue"
  }
}
```

この例は、指定されたキューの最大メッセージサイズと可視性タイムアウト属性のみを取得します。

コマンド:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-
names MaximumMessageSize VisibilityTimeout
```

出力:

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「GetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
}



// 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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「GetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
  //     extendedRequestId: undefined,
```

```
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: { DelaySeconds: '1' }
  // }
  return response;
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「GetQueueAttributes」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: この例では、指定したキューのすべての属性を一覧表示します。

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

出力:

```
VisibilityTimeout                     : 30
DelaySeconds                          : 0
MaximumMessageSize                    : 262144
MessageRetentionPeriod                : 345600
ApproximateNumberOfMessages           : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed    : 0
CreatedTimestamp                      : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
```

```
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
                                          [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

例 2: この例では、指定したキューの指定した属性のみを別個に一覧表示します。

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

出力:

```
VisibilityTimeout                       : 30
DelaySeconds                            : 0
MaximumMessageSize                      : 262144
MessageRetentionPeriod                  : 345600
ApproximateNumberOfMessages             : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed      : 0
CreatedTimestamp                        : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                   : 12/29/2015 2:23:17 PM
QueueARN                                : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                  :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
```

```
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- 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GetQueueAttributes」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定したキューのすべての属性を一覧表示します。

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

出力:

```
VisibilityTimeout                       : 30
DelaySeconds                            : 0
MaximumMessageSize                      : 262144
MessageRetentionPeriod                  : 345600
ApproximateNumberOfMessages             : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed      : 0
CreatedTimestamp                        : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                   : 12/29/2015 2:23:17 PM
QueueARN                                : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                  :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                              : {[QueueArn, arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue], [ApproximateNumberOfMessages, 0],
```

```
                                          [ApproximateNumberOfMessagesNotVisible,
 0], [ApproximateNumberOfMessagesDelayed, 0]...}
```

例 2: この例では、指定したキューの指定した属性のみを別個に一覧表示します。

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

出力:

```
VisibilityTimeout                     : 30
DelaySeconds                          : 0
MaximumMessageSize                    : 262144
MessageRetentionPeriod                : 345600
ApproximateNumberOfMessages           : 0
ApproximateNumberOfMessagesNotVisible : 0
ApproximateNumberOfMessagesDelayed    : 0
CreatedTimestamp                      : 2/11/2015 5:53:35 PM
LastModifiedTimestamp                 : 12/29/2015 2:23:17 PM
QueueARN                              : arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue
Policy                                :
 {"Version":"2008-10-17","Id":"arn:aws:sqs:us-east-1:80398EXAMPLE:MyQueue/
SQSDefaultPolicy","Statement":[{"Sid":"Sid14

 495134224EX","Effect":"Allow","Principal":
{"AWS":"*"},"Action":"SQS:SendMessage","Resource":"arn:aws:sqs:us-east-1:80
                                          398EXAMPLE:MyQueue","Condition":
{"ArnEquals":{"aws:SourceArn":"arn:aws:sns:us-east-1:80398EXAMPLE:MyTopic"}}},
{"Sid":

 "SendMessagesFromMyQueue","Effect":"Allow","Principal":
{"AWS":"80398EXAMPLE"},"Action":"SQS:SendMessage","Resource":"
                                          arn:aws:sqs:us-
east-1:80398EXAMPLE:MyQueue"}]}
Attributes                            : {[MaximumMessageSize, 262144],
 [VisibilityTimeout, 30]}
```

- 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 GetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.getQueueAttributes(
            input: GetQueueAttributesInput(
                attributeNames: [
                    .approximatenumberofmessages,
                    .maximummessagesize
                ],
                queueUrl: url
            )
        )

        guard let attributes = output.attributes else {
            print("No queue attributes returned.")
            return
        }

        for (attr, value) in attributes {
            switch(attr) {
            case "ApproximateNumberOfMessages":
                print("Approximate message count: \(value)")
            case "MaximumMessageSize":
                print("Maximum message size: \(value)kB")
            default:
                continue
            }
        }
```

- API の詳細については、 AWS SDK for Swift API リファレンスのGetQueueAttributes」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 SQS 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 **GetQueueUrl**で を使用する

次のサンプルコードは、`GetQueueUrl` 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。AWS コード例リポジトリ で全く同じ例を見つけて、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.SQS;
using Amazon.SQS.Model;

public class GetQueueUrl
{
    /// <summary>
    /// Initializes the Amazon SQS client object and then calls the
    /// GetQueueUrlAsync method to retrieve the URL of an Amazon SQS
    /// queue.
    /// </summary>
    public static async Task Main()
    {
        // If the Amazon SQS message queue is not in the same AWS Region as
your
        // default user, you need to provide the AWS Region as a parameter to
the
        // client constructor.
        var client = new AmazonSQSClient();
```

```
            string queueName = "New-Example-Queue";

            try
            {
                var response = await client.GetQueueUrlAsync(queueName);

                if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
                {
                    Console.WriteLine($"The URL for {queueName} is:
 {response.QueueUrl}");
                }
            }
            catch (QueueDoesNotExistException ex)
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine($"The queue {queueName} was not found.");
            }
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「GetQueueUrl」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! Get the URL for an Amazon Simple Queue Service (Amazon SQS) queue.
```

```
/*!
  \param queueName: An Amazon SQS queue name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::getQueueUrl(const Aws::String &queueName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::GetQueueUrlRequest request;
    request.SetQueueName(queueName);

    const Aws::SQS::Model::GetQueueUrlOutcome outcome =
 sqsClient.GetQueueUrl(request);
    if (outcome.IsSuccess()) {
        std::cout << "Queue " << queueName << " has url " <<
                  outcome.GetResult().GetQueueUrl() << std::endl;
    }
    else {
        std::cerr << "Error getting url for queue " << queueName << ": " <<
                  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「GetQueueUrl」を参照してください。

CLI

AWS CLI

キューの URL を取得するには

この例は、指定されたキューの URL を取得します。

コマンド:

```
aws sqs get-queue-url --queue-name MyQueue
```

出力:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「GetQueueUrl」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
            GetQueueUrlResponse getQueueUrlResponse = sqsClient

.getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「GetQueueUrl」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューの URL を取得します。

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
  console.log(response);
  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「GetQueueUrl」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューの URL を取得します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  QueueName: "SQS_QUEUE_NAME",
};

sqs.getQueueUrl(params, function (err, data) {
  if (err) {
```

```
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrl);
  }
});
```

- 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「GetQueueUrl」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: この例では、指定した名前のキューの URL を表示します。

```
Get-SQSQueueUrl -QueueName MyQueue
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

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

Tools for PowerShell V5

例 1: この例では、指定した名前のキューの URL を表示します。

```
Get-SQSQueueUrl -QueueName MyQueue
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API の詳細については、「 コマンドレットリファレンス (V5)」のGetQueueUrl」を参照してください。 AWS Tools for PowerShell

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def get_queue(name):
    """
    Gets an SQS queue by name.

    :param name: The name that was used to create the queue.
    :return: A Queue object.
    """
    try:
        queue = sqs.get_queue_by_name(QueueName=name)
        logger.info("Got queue '%s' with URL=%s", name, queue.url)
    except ClientError as error:
        logger.exception("Couldn't get queue named %s.", name)
        raise error
    else:
        return queue
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「GetQueueUrl」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    TRY.
        oo_result = lo_sqs->getqueueurl( iv_queuename = iv_queue_name ).        "
 oo_result is returned for testing purposes. "
        MESSAGE 'Queue URL retrieved.' TYPE 'I'.
      CATCH /aws1/cx_sqsqueuedoesnotexist.
        MESSAGE 'The requested queue 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「GetQueueUrl」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 SQS 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。

CLI で **ListDeadLetterSourceQueues** を使用する

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

CLI

AWS CLI

デッドレターソースキューを一覧表示するには

この例では、指定されたデッドレターソースキューに関連付けられているキューを一覧表示します。

コマンド:

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

出力:

```
{
  "queueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
```

```
  ]
}
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、デッドレターキューとして指定したキューに依存するすべてのキューの URL を一覧表示します。

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- 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ListDeadLetterSourceQueues」を参照してください。

Tools for PowerShell V5

例 1: この例では、デッドレターキューとして指定したキューに依存するすべてのキューの URL を一覧表示します。

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- 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ListDeadLetterSourceQueues」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 SQS 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 **ListQueues**で を使用する

次のサンプルコードは、ListQueues 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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! List the Amazon Simple Queue Service (Amazon SQS) queues within an AWS
 account.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::SQS::listQueues(const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ListQueuesRequest listQueuesRequest;

    Aws::String nextToken; // Used for pagination.
    Aws::Vector<Aws::String> allQueueUrls;

    do {
        if (!nextToken.empty()) {
            listQueuesRequest.SetNextToken(nextToken);
        }
```

```
        const Aws::SQS::Model::ListQueuesOutcome outcome = sqsClient.ListQueues(
                listQueuesRequest);
        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String> &queueUrls =
 outcome.GetResult().GetQueueUrls();
            allQueueUrls.insert(allQueueUrls.end(),
                                queueUrls.begin(),
                                queueUrls.end());

            nextToken = outcome.GetResult().GetNextToken();
        }
        else {
            std::cerr << "Error listing queues: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

    } while (!nextToken.empty());

    std::cout << allQueueUrls.size() << " Amazon SQS queue(s) found." <<
 std::endl;
    for (const auto &iter: allQueueUrls) {
        std::cout << " " << iter << std::endl;
    }

    return true;
}
```

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

CLI

AWS CLI

キューを一覧表示するには

この例は、すべてのキューを一覧表示します。

コマンド:

```
aws sqs list-queues
```

出力:

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/TestQueue1",
        "https://queue.amazonaws.com/80398EXAMPLE/TestQueue2"
  ]
}
```

この例は、「My」で始まるキューのみを一覧表示します。

コマンド:

```
aws sqs list-queues --queue-name-prefix My
```

出力:

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、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/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues 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
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
```

```
    log.Printf("Couldn't get queues. Here's why: %v\n", err)
    break
   } else {
    queueUrls = append(queueUrls, output.QueueUrls...)
   }
  }
  if len(queueUrls) == 0 {
   fmt.Println("You don't have any queues!")
  } else {
   for _, queueUrl := range queueUrls {
    fmt.Printf("\t%v\n", queueUrl)
   }
  }
 }
```

- API の詳細については、AWS SDK for Go API リファレンスの「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        String prefix = "que";

        try {
            ListQueuesRequest listQueuesRequest =
ListQueuesRequest.builder().queueNamePrefix(prefix).build();
            ListQueuesResponse listQueuesResponse =
sqsClient.listQueues(listQueuesRequest);
            for (String url : listQueuesResponse.queueUrls()) {
                System.out.println(url);
            }
```

```
        } catch (SqsException 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「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューを一覧表示します。

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    for (const url of urls) {
      console.log(url);
    }
  }

  return urls;
};
```

- 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「ListQueues」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューを一覧表示します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {};

sqs.listQueues(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.QueueUrls);
  }
});
```

- 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「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun listQueues() {
    println("\nList Queues")

    val prefix = "que"
    val listQueuesRequest =
        ListQueuesRequest {
            queueNamePrefix = prefix
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.listQueues(listQueuesRequest)
        response.queueUrls?.forEach { url ->
            println(url)
        }
    }
}
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、すべてのキューを一覧表示します。

```
Get-SQSQueue
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/AnotherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/DeadLetterQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

例 2: この例では、指定した名前で始まるすべてのキューを一覧表示します。

```
Get-SQSQueue -QueueNamePrefix My
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

- 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 ListQueues」を参照してください。

Tools for PowerShell V5

例 1: この例では、すべてのキューを一覧表示します。

```
Get-SQSQueue
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/AnotherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/DeadLetterQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

例 2: この例では、指定した名前で始まるすべてのキューを一覧表示します。

```
Get-SQSQueue -QueueNamePrefix My
```

出力:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

- 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ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def get_queues(prefix=None):
    """
    Gets a list of SQS queues. When a prefix is specified, only queues with names
    that start with the prefix are returned.

    :param prefix: The prefix used to restrict the list of returned queues.
    :return: A list of Queue objects.
    """
    if prefix:
        queue_iter = sqs.queues.filter(QueueNamePrefix=prefix)
    else:
        queue_iter = sqs.queues.all()
    queues = list(queue_iter)
    if queues:
        logger.info("Got queues: %s", ", ".join([q.url for q in queues]))
    else:
        logger.warning("No queues found.")
    return queues
```

- API の詳細については、 SDK for Python (Boto3) API リファレンスの「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @example
#   list_queue_urls(Aws::SQS::Client.new(region: 'us-west-2'))
def list_queue_urls(sqs_client)
  queues = sqs_client.list_queues

  queues.queue_urls.each do |url|
    puts url
  end
rescue StandardError => e
  puts "Error listing queue URLs: #{e.message}"
end

# Lists the attributes of a queue in Amazon Simple Queue Service (Amazon SQS).
#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @example
#   list_queue_attributes(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
#   )
def list_queue_attributes(sqs_client, queue_url)
  attributes = sqs_client.get_queue_attributes(
    queue_url: queue_url,
    attribute_names: ['All']
  )
```

```
  attributes.attributes.each do |key, value|
    puts "#{key}: #{value}"
  end
rescue StandardError => e
  puts "Error getting queue attributes: #{e.message}"
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'

  sqs_client = Aws::SQS::Client.new(region: region)

  puts 'Listing available queue URLs...'
  list_queue_urls(sqs_client)

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  puts "\nGetting information about queue '#{queue_name}'..."
  list_queue_attributes(sqs_client, queue_url)
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

リージョンにリストされている最初の Amazon SQS キューを取得します。

```
async fn find_first_queue(client: &Client) -> Result<String, Error> {
    let queues = client.list_queues().send().await?;
    let queue_urls = queues.queue_urls();
    Ok(queue_urls
        .first()
        .expect("No queues in this account and Region. Create a queue to
 proceed.")
        .to_string())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    TRY.
        oo_result = lo_sqs->listqueues( ).        " oo_result is returned for
 testing purposes. "
        MESSAGE 'Retrieved list of queues.' TYPE 'I'.
```

```
    ENDTRY.
```

- API の詳細については、「AWS SDK for SAP ABAP API リファレンス」の「ListQueues」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        var queues: [String] = []
        let outputPages = sqsClient.listQueuesPaginated(
            input: ListQueuesInput()
        )

        // Each time a page of results arrives, process its contents.

        for try await output in outputPages {
            guard let urls = output.queueUrls else {
                print("No queues found.")
                return
            }

            // Iterate over the queue URLs listed on this page, adding them
            // to the `queues` array.

            for queueUrl in urls {
                queues.append(queueUrl)
            }
        }
```

- API の詳細については、 AWS SDK for Swift API リファレンスのListQueues」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 SQS 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。

CLI で **PurgeQueue** を使用する

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

CLI

AWS CLI

キューをクリアするには

この例では、指定されたキューのすべてのメッセージを削除します。

コマンド:

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyNewQueue
```

出力:

```
None.
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定したキューからすべてのメッセージを削除します。

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

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

Tools for PowerShell V5

例 1: この例では、指定したキューからすべてのメッセージを削除します。

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API の詳細については、「 コマンドレットリファレンス (V5)」のPurgeQueue」を参照してください。 AWS Tools for PowerShell

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 SQS 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 **ReceiveMessage**で を使用する

次のサンプルコードは、`ReceiveMessage` 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。

- S3 を使用して大きなメッセージを管理する
- S3 イベント通知の処理
- メッセージをキューに発行する
- バッチメッセージを送受信する

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

URL を使用してキューからメッセージを受信します。

```
    /// <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;
    }
```

Amazon SQS キューからメッセージを受信し、メッセージを削除します。

```
        public static async Task Main()
        {
            // If the AWS Region you want to use is different from
            // the AWS Region defined for the default user, supply
            // the specify your AWS Region to the client constructor.
            var client = new AmazonSQSClient();
            string queueName = "Example_Queue";

            var queueUrl = await GetQueueUrl(client, queueName);
            Console.WriteLine($"The SQS queue's URL is {queueUrl}");

            var response = await ReceiveAndDeleteMessage(client, queueUrl);

            Console.WriteLine($"Message: {response.Messages[0]}");
        }

        /// <summary>
        /// Retrieve the queue URL for the queue named in the queueName
```

```
        /// property using the client object.
        /// </summary>
        /// <param name="client">The Amazon SQS client used to retrieve the
        /// queue URL.</param>
        /// <param name="queueName">A string representing  name of the queue
        /// for which to retrieve the URL.</param>
        /// <returns>The URL of the queue.</returns>
        public static async Task<string> GetQueueUrl(IAmazonSQS client, string
 queueName)
        {
            var request = new GetQueueUrlRequest
            {
                QueueName = queueName,
            };

            GetQueueUrlResponse response = await
 client.GetQueueUrlAsync(request);
            return response.QueueUrl;
        }

        /// <summary>
        /// Retrieves the message from the quque at the URL passed in the
        /// queueURL parameters using the client.
        /// </summary>
        /// <param name="client">The SQS client used to retrieve a message.</
param>
        /// <param name="queueUrl">The URL of the queue from which to retrieve
        /// a message.</param>
        /// <returns>The response from the call to ReceiveMessageAsync.</returns>
        public static async Task<ReceiveMessageResponse>
 ReceiveAndDeleteMessage(IAmazonSQS client, string queueUrl)
        {
            // Receive a single message from the queue.
            var receiveMessageRequest = new ReceiveMessageRequest
            {
                AttributeNames = { "SentTimestamp" },
                MaxNumberOfMessages = 1,
                MessageAttributeNames = { "All" },
                QueueUrl = queueUrl,
                VisibilityTimeout = 0,
                WaitTimeSeconds = 0,
            };
```

```
            var receiveMessageResponse = await
 client.ReceiveMessageAsync(receiveMessageRequest);

            // Delete the received message from the queue.
            var deleteMessageRequest = new DeleteMessageRequest
            {
                QueueUrl = queueUrl,
                ReceiptHandle = receiveMessageResponse.Messages[0].ReceiptHandle,
            };

            await client.DeleteMessageAsync(deleteMessageRequest);

            return receiveMessageResponse;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! Receive a message from an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
```

```
bool AwsDoc::SQS::receiveMessage(const Aws::String &queueUrl,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::ReceiveMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetMaxNumberOfMessages(1);

    const Aws::SQS::Model::ReceiveMessageOutcome outcome =
 sqsClient.ReceiveMessage(
            request);
    if (outcome.IsSuccess()) {

        const Aws::Vector<Aws::SQS::Model::Message> &messages =
                outcome.GetResult().GetMessages();
        if (!messages.empty()) {
            const Aws::SQS::Model::Message &message = messages[0];
            std::cout << "Received message:" << std::endl;
            std::cout << "  MessageId: " << message.GetMessageId() << std::endl;
            std::cout << "  ReceiptHandle: " << message.GetReceiptHandle() <<
 std::endl;
            std::cout << "  Body: " << message.GetBody() << std::endl <<
 std::endl;
        }
        else {
            std::cout << "No messages received from queue " << queueUrl <<
                      std::endl;

        }
    }
    else {
        std::cerr << "Error receiving message from queue " << queueUrl << ": "
                  << 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「ReceiveMessage」を参照してください。

CLI

AWS CLI

メッセージを受信するには

この例は、最大 10 件のメッセージを受信し、使用可能なすべての属性を返します。

コマンド:

```
aws sqs receive-message --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All --message-attribute-names All --max-number-of-messages 10
```

出力:

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEBzbVv...fqNzFw==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "9424c491...26bc3ae7",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "ApproximateFirstReceiveTimestamp": "1442428276921",
        "SenderId": "AIDAIAZKMSNQ7TEXAMPLE",
        "ApproximateReceiveCount": "5",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        },
        "City": {
          "DataType": "String",
          "StringValue": "Any City"
        }
      }
    }
  ]
}
```

この例は、次に受信可能なメッセージを受信し、PostalCode メッセージ属性に加えて、SenderId 属性と SentTimestamp 属性のみを返します。

コマンド:

```
aws sqs receive-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-
names SenderId SentTimestamp --message-attribute-names PostalCode
```

出力:

```
{
  "Messages": [
    {
      "Body": "My first message.",
      "ReceiptHandle": "AQEB6nR4...HzlvZQ==",
      "MD5OfBody": "1000f835...a35411fa",
      "MD5OfMessageAttributes": "b8e89563...e088e74f",
      "MessageId": "d6790f8d-d575-4f01-bc51-40122EXAMPLE",
      "Attributes": {
        "SenderId": "AIDAIAZKMSNQ7TEXAMPLE",
        "SentTimestamp": "1442428276921"
      },
      "MessageAttributes": {
        "PostalCode": {
          "DataType": "String",
          "StringValue": "ABC123"
        }
      }
    }
  ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
}



// 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
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
        try {
            ReceiveMessageRequest receiveMessageRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .maxNumberOfMessages(5)
                    .build();
            return sqsClient.receiveMessage(receiveMessageRequest).messages();

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

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューからメッセージを受信します。

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
```

```
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

ロングポーリングサポートを使用して Amazon SQS キューからメッセージを受信します。

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
    // The duration (in seconds) for which the call waits for a message
    // to arrive in the queue before returning. If a message is available,
    // the call returns sooner than WaitTimeSeconds. If no messages are
    // available and the wait time expires, the call returns successfully
    // with an empty list of messages.
    // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
    WaitTimeSeconds: 20,
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、「AWS SDK for JavaScript API リファレンス」の「ReceiveMessage」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

ロングポーリングサポートを使用して Amazon SQS キューからメッセージを受信します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create the SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var queueURL = "SQS_QUEUE_URL";

var params = {
  AttributeNames: ["SentTimestamp"],
  MaxNumberOfMessages: 1,
  MessageAttributeNames: ["All"],
  QueueUrl: queueURL,
  WaitTimeSeconds: 20,
};

sqs.receiveMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data);
  }
```

```
});
```

- 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「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun receiveMessages(queueUrlVal: String?) {
    println("Retrieving messages from $queueUrlVal")

    val receiveMessageRequest =
        ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.receiveMessage(receiveMessageRequest)
        response.messages?.forEach { message ->
            println(message.body)
        }
    }
}
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定したキューで次に受信する 10 件までのメッセージに関する情報を表示します。情報には、指定したメッセージ属性 (存在する場合) の値が含まれます。

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

出力:

```
Attributes             : {[SenderId, AIDAIAZKMSNQ7TEXAMPLE], [SentTimestamp,
 1451495923744]}
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
MessageAttributes      : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
 [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```

- 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[ReceiveMessage](#)」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定したキューで次に受信する 10 件までのメッセージに関する情報を表示します。情報には、指定したメッセージ属性 (存在する場合) の値が含まれます。

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

出力:

```
Attributes             : {[SenderId, AIDAIAZKMSNQ7TEXAMPLE], [SentTimestamp,
 1451495923744]}
Body                   : Information about John Doe's grade.
MD5OfBody              : ea572796e3c231f974fe75d89EXAMPLE
MD5OfMessageAttributes : 48c1ee811f0fe7c4e88fbe0f5EXAMPLE
```

```
MessageAttributes      : {[StudentGrade, Amazon.SQS.Model.MessageAttributeValue],
 [StudentName, Amazon.SQS.Model.MessageAttributeValue]}
MessageId              : 53828c4b-631b-469b-8833-c093cEXAMPLE
ReceiptHandle          : AQEBpfGp...20Q5cg==
```

- 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 ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def receive_messages(queue, max_number, wait_time):
    """
    Receive a batch of messages in a single request from an SQS queue.

    :param queue: The queue from which to receive messages.
    :param max_number: The maximum number of messages to receive. The actual
 number
                        of messages received might be less.
    :param wait_time: The maximum time to wait (in seconds) before returning.
 When
                       this number is greater than zero, long polling is used.
 This
                       can result in reduced costs and fewer false empty
 responses.
    :return: The list of Message objects received. These each contain the body
             of the message and metadata and custom attributes.
    """
    try:
        messages = queue.receive_messages(
            MessageAttributeNames=["All"],
            MaxNumberOfMessages=max_number,
            WaitTimeSeconds=wait_time,
        )
```

```
        for msg in messages:
            logger.info("Received message: %s: %s", msg.message_id, msg.body)
    except ClientError as error:
        logger.exception("Couldn't receive messages from queue: %s", queue)
        raise error
    else:
        return messages
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

# Receives messages in a queue in Amazon Simple Queue Service (Amazon SQS).
#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param max_number_of_messages [Integer] The maximum number of messages
#   to receive. This number must be 10 or less. The default is 10.
# @example
#   receive_messages(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     10
#   )
def receive_messages(sqs_client, queue_url, max_number_of_messages = 10)
```

```
  if max_number_of_messages > 10
    puts 'Maximum number of messages to receive must be 10 or less. ' \
      'Stopping program.'
    return
  end

  response = sqs_client.receive_message(
    queue_url: queue_url,
    max_number_of_messages: max_number_of_messages
  )

  if response.messages.count.zero?
    puts 'No messages to receive, or all messages have already ' \
      'been previously received.'
    return
  end

  response.messages.each do |message|
    puts '-' * 20
    puts "Message body: #{message.body}"
    puts "Message ID:   #{message.message_id}"
  end
rescue StandardError => e
  puts "Error receiving messages: #{e.message}"
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  max_number_of_messages = 10

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Receiving messages from queue '#{queue_name}'..."
```

```
  receive_messages(sqs_client, queue_url, max_number_of_messages)
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
async fn receive(client: &Client, queue_url: &String) -> Result<(), Error> {
    let rcv_message_output =
 client.receive_message().queue_url(queue_url).send().await?;

    println!("Messages from queue with url: {}", queue_url);

    for message in rcv_message_output.messages.unwrap_or_default() {
        println!("Got the message: {:#?}", message);
    }

    Ok(())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューからメッセージを受信します。

```
   TRY.
       oo_result = lo_sqs->receivemessage( iv_queueurl = iv_queue_url ).    "
oo_result is returned for testing purposes. "
       DATA(lt_messages) = oo_result->get_messages( ).
       MESSAGE 'Message received from SQS queue.' TYPE 'I'.
     CATCH /aws1/cx_sqsoverlimit.
       MESSAGE 'Maximum number of in-flight messages reached.' TYPE 'E'.
   ENDTRY.
```

ロングポーリングサポートを使用して Amazon SQS キューからメッセージを受信します。

```
   TRY.
       oo_result = lo_sqs->receivemessage(           " oo_result is returned for
testing purposes. "
               iv_queueurl = iv_queue_url
               iv_waittimeseconds = iv_wait_time ).    " Time in seconds for
long polling, such as how long the call waits for a message to arrive in the
queue before returning. " ).
       DATA(lt_messages) = oo_result->get_messages( ).
       MESSAGE 'Message received from SQS queue.' TYPE 'I'.
     CATCH /aws1/cx_sqsoverlimit.
       MESSAGE 'Maximum number of in-flight messages reached.' 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「ReceiveMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        let output = try await sqsClient.receiveMessage(
            input: ReceiveMessageInput(
                maxNumberOfMessages: maxMessages,
                queueUrl: url
            )
        )

        guard let messages = output.messages else {
            print("No messages received.")
            return
        }

        for message in messages {
            print("Message ID:     \(message.messageId ?? "<unknown>")")
            print("Receipt handle: \(message.receiptHandle ?? "<unknown>")")
            print(message.body ?? "<body missing>")
            print("---")
        }
```

- API の詳細については、 AWS SDK for Swift API リファレンスのReceiveMessage」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 SQS 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。

CLI で **RemovePermission** を使用する

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

CLI

AWS CLI

アクセス許可を削除するには

この例では、指定されたキューから指定されたラベルを持つアクセス許可を削除します。

コマンド:

```
aws sqs remove-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue
```

出力:

```
None.
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定したラベルを持つアクセス許可設定を、指定したキューから削除します。

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet リファレンス (V4)」の「RemovePermission」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定したラベルを持つアクセス許可設定を、指定したキューから削除します。

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet リファレンス (V5)」の「RemovePermission」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 SQS 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 **SendMessage**で を使用する

次のサンプルコードは、SendMessage 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。

- S3 を使用して大きなメッセージを管理する
- バッチメッセージを送受信する

.NET

SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューを作成して、そこにメッセージを送信します。

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
```

```
    using Amazon;
    using Amazon.SQS;
    using Amazon.SQS.Model;

    public class CreateSendExample
    {
        // Specify your AWS Region (an example Region is shown).
        private static readonly string QueueName = "Example_Queue";
        private static readonly RegionEndpoint ServiceRegion =
 RegionEndpoint.USWest2;
        private static IAmazonSQS client;

        public static async Task Main()
        {
            client = new AmazonSQSClient(ServiceRegion);
            var createQueueResponse = await CreateQueue(client, QueueName);

            string queueUrl = createQueueResponse.QueueUrl;

            Dictionary<string, MessageAttributeValue> messageAttributes = new
 Dictionary<string, MessageAttributeValue>
            {
                { "Title",   new MessageAttributeValue { DataType = "String",
 StringValue = "The Whistler" } },
                { "Author",  new MessageAttributeValue { DataType = "String",
 StringValue = "John Grisham" } },
                { "WeeksOn", new MessageAttributeValue { DataType = "Number",
 StringValue = "6" } },
            };

            string messageBody = "Information about current NY Times fiction
 bestseller for week of 12/11/2016.";

            var sendMsgResponse = await SendMessage(client, queueUrl,
 messageBody, messageAttributes);
        }

        /// <summary>
        /// Creates a new Amazon SQS queue using the queue name passed to it
        /// in queueName.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueName">A string representing the name of the queue
```

```
        /// to create.</param>
        /// <returns>A CreateQueueResponse that contains information about the
        /// newly created queue.</returns>
        public static async Task<CreateQueueResponse> CreateQueue(IAmazonSQS
 client, string queueName)
        {
            var request = new CreateQueueRequest
            {
                QueueName = queueName,
                Attributes = new Dictionary<string, string>
                {
                    { "DelaySeconds", "60" },
                    { "MessageRetentionPeriod", "86400" },
                },
            };

            var response = await client.CreateQueueAsync(request);
            Console.WriteLine($"Created a queue with URL : {response.QueueUrl}");

            return response;
        }

        /// <summary>
        /// Sends a message to an SQS queue.
        /// </summary>
        /// <param name="client">An SQS client object used to send the message.</
param>
        /// <param name="queueUrl">The URL of the queue to which to send the
        /// message.</param>
        /// <param name="messageBody">A string representing the body of the
        /// message to be sent to the queue.</param>
        /// <param name="messageAttributes">Attributes for the message to be
        /// sent to the queue.</param>
        /// <returns>A SendMessageResponse object that contains information
        /// about the message that was sent.</returns>
        public static async Task<SendMessageResponse> SendMessage(
            IAmazonSQS client,
            string queueUrl,
            string messageBody,
            Dictionary<string, MessageAttributeValue> messageAttributes)
        {
            var sendMessageRequest = new SendMessageRequest
            {
                DelaySeconds = 10,
```

```
                MessageAttributes = messageAttributes,
                MessageBody = messageBody,
                QueueUrl = queueUrl,
            };

            var response = await client.SendMessageAsync(sendMessageRequest);
            Console.WriteLine($"Sent a message with id : {response.MessageId}");

            return response;
        }
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「SendMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! Send a message to an Amazon Simple Queue Service (Amazon SQS) queue.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param messageBody: A message body.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::sendMessage(const Aws::String &queueUrl,
                              const Aws::String &messageBody,
```

```
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SendMessageRequest request;
    request.SetQueueUrl(queueUrl);
    request.SetMessageBody(messageBody);

    const Aws::SQS::Model::SendMessageOutcome outcome =
 sqsClient.SendMessage(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully sent message to " << queueUrl <<
                  std::endl;
    }
    else {
        std::cerr << "Error sending message to " << queueUrl << ": " <<
                  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「SendMessage」を参照してください。

CLI

AWS CLI

単一のメッセージを送信するには

この例は、指定された単一のメッセージ本文、遅延期間、メッセージ属性を含むメッセージを指定されたキューに送信します。

コマンド:

```
aws sqs send-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --message-body "Information about the
 largest city in Any Region." --delay-seconds 10 --message-attributes file://
send-message.json
```

入力ファイル (send-message.json):

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

出力:

```
{
  "MD5OfMessageBody": "51b0a325...39163aa0",
  "MD5OfMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「SendMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

SendMessage オペレーションの 2 つの例を次に示します。

- 本文と遅延を含むメッセージを送信する

- 本文とメッセージ属性を含むメッセージを送信する

本文と遅延を含むメッセージを送信します。

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SqsException;

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

                Usage:    <queueName> <message>

                Where:
                   queueName - The name of the queue.
                   message - The message to send.
                """;

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

        String queueName = args[0];
        String message = args[1];
        SqsClient sqsClient = SqsClient.builder()
                .region(Region.US_WEST_2)
                .build();
        sendMessage(sqsClient, queueName, message);
        sqsClient.close();
```

```
    }

    public static void sendMessage(SqsClient sqsClient, String queueName, String
 message) {
        try {
            CreateQueueRequest request = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();
            sqsClient.createQueue(request);

            GetQueueUrlRequest getQueueRequest = GetQueueUrlRequest.builder()
                    .queueName(queueName)
                    .build();

            String queueUrl = sqsClient.getQueueUrl(getQueueRequest).queueUrl();
            SendMessageRequest sendMsgRequest = SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody(message)
                    .delaySeconds(5)
                    .build();

            sqsClient.sendMessage(sendMsgRequest);

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

本文とメッセージ属性を含むメッセージを送信します。

```
    /**
     * <p>This method demonstrates how to add message attributes to a message.
     * Each attribute must specify a name, value, and data type. You use a Java
 Map to supply the attributes. The map's
     * key is the attribute name, and you specify the map's entry value using a
 builder that includes the attribute
     * value and data type.</p>
     *
     * <p>The data type must start with one of "String", "Number" or "Binary".
 You can optionally
```

```
     * define a custom extension by using a "." and your extension.</p>
     *
     * <p>The SQS Developer Guide provides more information on @see <a
     * href="https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-message-metadata.html#sqs-message-attributes">message
     * attributes</a>.</p>
     *
     * @param thumbailPath Filesystem path of the image.
     * @param queueUrl     URL of the SQS queue.
     */
    static void sendMessageWithAttributes(Path thumbailPath, String queueUrl) {
        Map<String, MessageAttributeValue> messageAttributeMap;
        try {
            messageAttributeMap = Map.of(
                    "Name", MessageAttributeValue.builder()
                            .stringValue("Jane Doe")
                            .dataType("String").build(),
                    "Age", MessageAttributeValue.builder()
                            .stringValue("42")
                            .dataType("Number.int").build(),
                    "Image", MessageAttributeValue.builder()

 .binaryValue(SdkBytes.fromByteArray(Files.readAllBytes(thumbailPath)))
                            .dataType("Binary.jpg").build()
            );
        } catch (IOException e) {
            LOGGER.error("An I/O exception occurred reading thumbnail image: {}",
 e.getMessage(), e);
            throw new RuntimeException(e);
        }

        SendMessageRequest request = SendMessageRequest.builder()
                .queueUrl(queueUrl)
                .messageBody("Hello SQS")
                .messageAttributes(messageAttributeMap)
                .build();
        try {
            SendMessageResponse sendMessageResponse =
 SQS_CLIENT.sendMessage(request);
            LOGGER.info("Message ID: {}", sendMessageResponse.messageId());
        } catch (SqsException e) {
            LOGGER.error("Exception occurred sending message: {}",
 e.getMessage(), e);
            throw new RuntimeException(e);
```

```
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「SendMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューにメッセージを送信します。

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
      Title: {
        DataType: "String",
        StringValue: "The Whistler",
      },
      Author: {
        DataType: "String",
        StringValue: "John Grisham",
      },
      WeeksOn: {
        DataType: "Number",
        StringValue: "6",
      },
    },
```

```
    MessageBody:
      "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  });

  const response = await client.send(command);
  console.log(response);
  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「SendMessage」を参照してください。

SDK for JavaScript (v2)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS キューにメッセージを送信します。

```
// Load the AWS SDK for Node.js
var AWS = require("aws-sdk");
// Set the region
AWS.config.update({ region: "REGION" });

// Create an SQS service object
var sqs = new AWS.SQS({ apiVersion: "2012-11-05" });

var params = {
  // Remove DelaySeconds parameter and value for FIFO queues
  DelaySeconds: 10,
  MessageAttributes: {
    Title: {
      DataType: "String",
      StringValue: "The Whistler",
    },
    Author: {
      DataType: "String",
```

```
      StringValue: "John Grisham",
    },
    WeeksOn: {
      DataType: "Number",
      StringValue: "6",
    },
  },
  MessageBody:
    "Information about current NY Times fiction bestseller for week of
 12/11/2016.",
  // MessageDeduplicationId: "TheWhistler",  // Required for FIFO queues
  // MessageGroupId: "Group1",  // Required for FIFO queues
  QueueUrl: "SQS_QUEUE_URL",
};

sqs.sendMessage(params, function (err, data) {
  if (err) {
    console.log("Error", err);
  } else {
    console.log("Success", data.MessageId);
  }
});
```

- 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「SendMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
suspend fun sendMessages(
    queueUrlVal: String,
    message: String,
```

```
) {
    println("Sending multiple messages")
    println("\nSend message")
    val sendRequest =
        SendMessageRequest {
            queueUrl = queueUrlVal
            messageBody = message
            delaySeconds = 10
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.sendMessage(sendRequest)
        println("A single message was successfully sent.")
    }
}

suspend fun sendBatchMessages(queueUrlVal: String?) {
    println("Sending multiple messages")

    val msg1 =
        SendMessageBatchRequestEntry {
            id = "id1"
            messageBody = "Hello from msg 1"
        }

    val msg2 =
        SendMessageBatchRequestEntry {
            id = "id2"
            messageBody = "Hello from msg 2"
        }

    val sendMessageBatchRequest =
        SendMessageBatchRequest {
            queueUrl = queueUrlVal
            entries = listOf(msg1, msg2)
        }

    SqsClient.fromEnvironment { region = "us-east-1" }.use { sqsClient ->
        sqsClient.sendMessageBatch(sendMessageBatchRequest)
        println("Batch message were successfully sent.")
    }
}
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定した属性とメッセージ本文を持つメッセージを指定したキューに送信し、メッセージの配信を 10 秒遅延させます。

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

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

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

Send-SQSMessage -DelayInSeconds 10 -MessageAttributes $messageAttributes -
MessageBody "Information about the largest city in Any Region." -QueueUrl
 https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

出力:

```
MD5OfMessageAttributes              MD5OfMessageBody                    MessageId

----------------------              ----------------                    ---------

1d3e51347bc042efbdf6dda31EXAMPLE    51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- 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「SendMessage」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定した属性とメッセージ本文を持つメッセージを指定したキューに送信し、メッセージの配信を 10 秒遅延させます。

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

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

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

Send-SQSMessage -DelayInSeconds 10 -MessageAttributes $messageAttributes -
MessageBody "Information about the largest city in Any Region." -QueueUrl
 https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

出力:

```
MD5OfMessageAttributes             MD5OfMessageBody                   MessageId

----------------------             ----------------                   ---------

1d3e51347bc042efbdf6dda31EXAMPLE   51b0a3256d59467f973009b73EXAMPLE   c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- 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「SendMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def send_message(queue, message_body, message_attributes=None):
    """
```

```
    Send a message to an Amazon SQS queue.

    :param queue: The queue that receives the message.
    :param message_body: The body text of the message.
    :param message_attributes: Custom attributes of the message. These are key-
value
                               pairs that can be whatever you want.
    :return: The response from SQS that contains the assigned message ID.
    """
    if not message_attributes:
        message_attributes = {}

    try:
        response = queue.send_message(
            MessageBody=message_body, MessageAttributes=message_attributes
        )
    except ClientError as error:
        logger.exception("Send message failed: %s", message_body)
        raise error
    else:
        return response
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「SendMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'
```

```
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param message_body [String] The contents of the message to be sent.
# @return [Boolean] true if the message was sent; otherwise, false.
# @example
#   exit 1 unless message_sent?(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     'This is my message.'
#   )
def message_sent?(sqs_client, queue_url, message_body)
  sqs_client.send_message(
    queue_url: queue_url,
    message_body: message_body
  )
  true
rescue StandardError => e
  puts "Error sending message: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  message_body = 'This is my message.'

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Sending a message to the queue named '#{queue_name}'..."

  if message_sent?(sqs_client, queue_url, message_body)
    puts 'Message sent.'
  else
    puts 'Message not sent.'
```

```
  end
end

# Example usage:
run_me if $PROGRAM_NAME == __FILE__
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「SendMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
async fn send(client: &Client, queue_url: &String, message: &SQSMessage) ->
 Result<(), Error> {
    println!("Sending message to queue with URL: {}", queue_url);

    let rsp = client
        .send_message()
        .queue_url(queue_url)
        .message_body(&message.body)
        // If the queue is FIFO, you need to set .message_deduplication_id
        // and message_group_id or configure the queue for
 ContentBasedDeduplication.
        .send()
        .await?;

    println!("Send message to the queue: {:#?}", rsp);

    Ok(())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「SendMessage」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
   TRY.
       oo_result = lo_sqs->sendmessage(              " oo_result is returned for
testing purposes. "
          iv_queueurl = iv_queue_url
          iv_messagebody = iv_message ).
       MESSAGE 'Message sent to SQS queue.' TYPE 'I'.
     CATCH /aws1/cx_sqsinvalidmsgconts.
       MESSAGE 'Message contains non-valid characters.' TYPE 'E'.
     CATCH /aws1/cx_sqsunsupportedop.
       MESSAGE 'Operation not supported.' 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「SendMessage」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 SQS 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 **SendMessageBatch**で を使用する

次のサンプルコードは、SendMessageBatch 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。

- バッチメッセージを送受信する

CLI

AWS CLI

複数のメッセージを一括送信するには

この例は、メッセージ本文、遅延期間、およびメッセージ属性を指定した 2 つのメッセージを指定されたキューに送信します。

コマンド:

```
aws sqs send-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-batch.json
```

入力ファイル (send-message-batch.json):

```
[
  {
    "Id": "FuelReport-0001-2015-09-16T140731Z",
        "MessageBody": "Fuel report for account 0001 on 2015-09-16 at 02:07:31
 PM.",
        "DelaySeconds": 10,
        "MessageAttributes": {
          "SellerName": {
            "DataType": "String",
                "StringValue": "Example Store"
      },
          "City": {
        "DataType": "String",
        "StringValue": "Any City"
      },
          "Region": {
            "DataType": "String",
                "StringValue": "WA"
      },
          "PostalCode": {
            "DataType": "String",
                "StringValue": "99065"
          },
```

```
          "PricePerGallon": {
            "DataType": "Number",
                "StringValue": "1.99"
      }
        }
  },
  {
    "Id": "FuelReport-0002-2015-09-16T140930Z",
        "MessageBody": "Fuel report for account 0002 on 2015-09-16 at 02:09:30
 PM.",
        "DelaySeconds": 10,
        "MessageAttributes": {
          "SellerName": {
            "DataType": "String",
                "StringValue": "Example Fuels"
      },
          "City": {
        "DataType": "String",
        "StringValue": "North Town"
      },
          "Region": {
            "DataType": "String",
                "StringValue": "WA"
      },
          "PostalCode": {
            "DataType": "String",
                "StringValue": "99123"
          },
          "PricePerGallon": {
            "DataType": "Number",
                "StringValue": "1.87"
      }
        }
  }
]
```

出力:

```
{
  "Successful": [
    {
      "MD5OfMessageBody": "203c4a38...7943237e",
      "MD5OfMessageAttributes": "10809b55...baf283ef",
```

```
      "Id": "FuelReport-0001-2015-09-16T140731Z",
      "MessageId": "d175070c-d6b8-4101-861d-adeb3EXAMPLE"
    },
    {
      "MD5OfMessageBody": "2cf0159a...c1980595",
      "MD5OfMessageAttributes": "55623928...ae354a25",
      "Id": "FuelReport-0002-2015-09-16T140930Z",
      "MessageId": "f9b7d55d-0570-413e-b9c5-a9264EXAMPLE"
    }
  ]
}
```

- API の詳細については、AWS CLI コマンドリファレンスの「SendMessageBatch」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
            SendMessageBatchRequest sendMessageBatchRequest =
SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)

.entries(SendMessageBatchRequestEntry.builder().id("id1").messageBody("Hello
from msg 1").build(),

SendMessageBatchRequestEntry.builder().id("id2").messageBody("msg
2").delaySeconds(10)
                                        .build())
                    .build();
            sqsClient.sendMessageBatch(sendMessageBatchRequest);
```

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

PowerShell

Tools for PowerShell V4

例 1: この例では、指定した属性とメッセージ本文を持つ 2 つのメッセージを指定したキューに送信します。最初のメッセージでは配信を 15 秒遅延させ、2 番目のメッセージでは配信を 10 秒遅延させます。

```
$student1NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1NameAttributeValue.DataType = "String"
$student1NameAttributeValue.StringValue = "John Doe"

$student1GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1GradeAttributeValue.DataType = "Number"
$student1GradeAttributeValue.StringValue = "89"

$student2NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2NameAttributeValue.DataType = "String"
$student2NameAttributeValue.StringValue = "Jane Doe"

$student2GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2GradeAttributeValue.DataType = "Number"
$student2GradeAttributeValue.StringValue = "93"

$message1 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message1.DelaySeconds = 15
$message1.Id = "FirstMessage"
$message1.MessageAttributes.Add("StudentName", $student1NameAttributeValue)
$message1.MessageAttributes.Add("StudentGrade", $student1GradeAttributeValue)
$message1.MessageBody = "Information about John Doe's grade."

$message2 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message2.DelaySeconds = 10
$message2.Id = "SecondMessage"
$message2.MessageAttributes.Add("StudentName", $student2NameAttributeValue)
$message2.MessageAttributes.Add("StudentGrade", $student2GradeAttributeValue)
$message2.MessageBody = "Information about Jane Doe's grade."

Send-SQSMessageBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -Entry $message1, $message2
```

出力:

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- 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SendMessageBatch」を参照してください。

Tools for PowerShell V5

例 1: この例では、指定した属性とメッセージ本文を持つ 2 つのメッセージを指定したキューに送信します。最初のメッセージでは配信を 15 秒遅延させ、2 番目のメッセージでは配信を 10 秒遅延させます。

```
$student1NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1NameAttributeValue.DataType = "String"
$student1NameAttributeValue.StringValue = "John Doe"

$student1GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student1GradeAttributeValue.DataType = "Number"
$student1GradeAttributeValue.StringValue = "89"

$student2NameAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2NameAttributeValue.DataType = "String"
$student2NameAttributeValue.StringValue = "Jane Doe"

$student2GradeAttributeValue = New-Object Amazon.SQS.Model.MessageAttributeValue
$student2GradeAttributeValue.DataType = "Number"
$student2GradeAttributeValue.StringValue = "93"

$message1 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message1.DelaySeconds = 15
$message1.Id = "FirstMessage"
$message1.MessageAttributes.Add("StudentName", $student1NameAttributeValue)
$message1.MessageAttributes.Add("StudentGrade", $student1GradeAttributeValue)
$message1.MessageBody = "Information about John Doe's grade."

$message2 = New-Object Amazon.SQS.Model.SendMessageBatchRequestEntry
$message2.DelaySeconds = 10
$message2.Id = "SecondMessage"
$message2.MessageAttributes.Add("StudentName", $student2NameAttributeValue)
$message2.MessageAttributes.Add("StudentGrade", $student2GradeAttributeValue)
```

```
$message2.MessageBody = "Information about Jane Doe's grade."

Send-SQSMessageBatch -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -Entry $message1, $message2
```

出力:

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- 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SendMessageBatch」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
def send_messages(queue, messages):
    """
    Send a batch of messages in a single request to an SQS queue.
    This request may return overall success even when some messages were not
 sent.
    The caller must inspect the Successful and Failed lists in the response and
    resend any failed messages.

    :param queue: The queue to receive the messages.
    :param messages: The messages to send to the queue. These are simplified to
                     contain only the message body and attributes.
    :return: The response from SQS that contains the list of successful and
 failed
             messages.
```

```
    """
    try:
        entries = [
            {
                "Id": str(ind),
                "MessageBody": msg["body"],
                "MessageAttributes": msg["attributes"],
            }
            for ind, msg in enumerate(messages)
        ]
        response = queue.send_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info(
                    "Message sent: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Failed to send: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
    except ClientError as error:
        logger.exception("Send messages failed to queue: %s", queue)
        raise error
    else:
        return response
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「SendMessageBatch」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
require 'aws-sdk-sqs'
require 'aws-sdk-sts'

#
# @param sqs_client [Aws::SQS::Client] An initialized Amazon SQS client.
# @param queue_url [String] The URL of the queue.
# @param entries [Hash] The contents of the messages to be sent,
#   in the correct format.
# @return [Boolean] true if the messages were sent; otherwise, false.
# @example
#   exit 1 unless messages_sent?(
#     Aws::SQS::Client.new(region: 'us-west-2'),
#     'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue',
#     [
#       {
#         id: 'Message1',
#         message_body: 'This is the first message.'
#       },
#       {
#         id: 'Message2',
#         message_body: 'This is the second message.'
#       }
#     ]
#   )
def messages_sent?(sqs_client, queue_url, entries)
  sqs_client.send_message_batch(
    queue_url: queue_url,
    entries: entries
  )
  true
rescue StandardError => e
```

```
  puts "Error sending messages: #{e.message}"
  false
end

# Full example call:
# Replace us-west-2 with the AWS Region you're using for Amazon SQS.
def run_me
  region = 'us-west-2'
  queue_name = 'my-queue'
  entries = [
    {
      id: 'Message1',
      message_body: 'This is the first message.'
    },
    {
      id: 'Message2',
      message_body: 'This is the second message.'
    }
  ]

  sts_client = Aws::STS::Client.new(region: region)

  # For example:
  # 'https://sqs.us-west-2.amazonaws.com/111111111111/my-queue'
  queue_url = "https://sqs.#{region}.amazonaws.com/
#{sts_client.get_caller_identity.account}/#{queue_name}"

  sqs_client = Aws::SQS::Client.new(region: region)

  puts "Sending messages to the queue named '#{queue_name}'..."

  if messages_sent?(sqs_client, queue_url, entries)
    puts 'Messages sent.'
  else
    puts 'Messages not sent.'
  end
end
```

- API の詳細については、「AWS SDK for Ruby API リファレンス」の「SendMessageBatch」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 SQS 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 **SetQueueAttributes**で を使用する

次のサンプルコードは、SetQueueAttributes 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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

トピックのキューのポリシー属性を設定します。

```
    /// <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;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「SetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、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";

//! Set the value for an attribute in an Amazon Simple Queue Service (Amazon SQS)
 queue.
```

```
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param attributeName: An attribute name enum.
  \param attribute: The attribute value as a string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setQueueAttributes(const Aws::String &queueURL,
                                     Aws::SQS::Model::QueueAttributeName
 attributeName,
                                     const Aws::String &attribute,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(queueURL);
    request.AddAttributes(
            attributeName,
            attribute);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
 sqsClient.SetQueueAttributes(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully set the attribute  " <<

 Aws::SQS::Model::QueueAttributeNameMapper::GetNameForQueueAttributeName(
                         attributeName)
                  << " with value " << attribute << " in queue " <<
                  queueURL << "." << std::endl;
    }
    else {
        std::cout << "Error setting attribute for  queue " <<
                  queueURL << ": " << outcome.GetError().GetMessage() <<
                  std::endl;
    }

    return outcome.IsSuccess();
}
```

デッドレターキューを設定します。

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

//! Connect an Amazon Simple Queue Service (Amazon SQS) queue to an associated
//! dead-letter queue.
/*!
  \param srcQueueUrl: An Amazon SQS queue URL.
  \param deadLetterQueueARN: The Amazon Resource Name (ARN) of an Amazon SQS
 dead-letter queue.
  \param maxReceiveCount: The max receive count of a message before it is sent to
 the dead-letter queue.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setDeadLetterQueue(const Aws::String &srcQueueUrl,
                                     const Aws::String &deadLetterQueueARN,
                                     int maxReceiveCount,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String redrivePolicy = MakeRedrivePolicy(deadLetterQueueARN,
 maxReceiveCount);

    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(srcQueueUrl);
    request.AddAttributes(
            Aws::SQS::Model::QueueAttributeName::RedrivePolicy,
            redrivePolicy);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
            sqsClient.SetQueueAttributes(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully set dead letter queue for queue  " <<
                  srcQueueUrl << " to " << deadLetterQueueARN << std::endl;
    }
    else {
        std::cerr << "Error setting dead letter queue for queue " <<
                  srcQueueUrl << ": " << outcome.GetError().GetMessage() <<
                  std::endl;
    }
```

```
    return outcome.IsSuccess();
}

//! Make a redrive policy for a dead-letter queue.
/*!
  \param queueArn: An Amazon SQS ARN for the dead-letter queue.
  \param maxReceiveCount: The max receive count of a message before it is sent to
 the dead-letter queue.
  \return Aws::String: Policy as JSON string.
 */
Aws::String MakeRedrivePolicy(const Aws::String &queueArn, int maxReceiveCount) {
    Aws::Utils::Json::JsonValue redrive_arn_entry;
    redrive_arn_entry.AsString(queueArn);

    Aws::Utils::Json::JsonValue max_msg_entry;
    max_msg_entry.AsInteger(maxReceiveCount);

    Aws::Utils::Json::JsonValue policy_map;
    policy_map.WithObject("deadLetterTargetArn", redrive_arn_entry);
    policy_map.WithObject("maxReceiveCount", max_msg_entry);

    return policy_map.View().WriteReadable();
}
```

ロングポーリングを使用するように Amazon SQS キューを設定します。

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

//! Set the wait time for an Amazon Simple Queue Service (Amazon SQS) queue poll.
/*!
  \param queueUrl: An Amazon SQS queue URL.
  \param pollTimeSeconds: The receive message wait time in seconds.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::SQS::setQueueLongPollingAttribute(const Aws::String &queueURL,
                                               const Aws::String
 &pollTimeSeconds,
                                               const
 Aws::Client::ClientConfiguration &clientConfiguration) {
```

```
    Aws::SQS::SQSClient sqsClient(clientConfiguration);

    Aws::SQS::Model::SetQueueAttributesRequest request;
    request.SetQueueUrl(queueURL);
    request.AddAttributes(
            Aws::SQS::Model::QueueAttributeName::ReceiveMessageWaitTimeSeconds,
            pollTimeSeconds);

    const Aws::SQS::Model::SetQueueAttributesOutcome outcome =
 sqsClient.SetQueueAttributes(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated long polling time for queue " <<
                  queueURL << " to " << pollTimeSeconds << std::endl;
    }
    else {
        std::cout << "Error updating long polling time for queue " <<
                  queueURL << ": " << 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「SetQueueAttributes」を参照してください。

CLI

AWS CLI

キューの属性を設定するには

この例は、指定されたキューの配信遅延時間を 10 秒、最大メッセージサイズを 128 KB (128 KB × 1,024 バイト)、メッセージ保持期間を 3 日間 (3 日 × 24 時間 × 60 分 × 60 秒)、受信メッセージ待機時間を 20 秒、デフォルトの可視性タイムアウトを 60 秒に設定します。また、この例では、指定されたデッドレターキューの最大受信数を 1,000 メッセージと関連付けます。

コマンド:

```
aws sqs set-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-
attributes.json
```

入力ファイル (set-queue-attributes.json):

```
{
  "DelaySeconds": "10",
  "MaximumMessageSize": "131072",
  "MessageRetentionPeriod": "259200",
  "ReceiveMessageWaitTimeSeconds": "20",
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "VisibilityTimeout": "60"
}
```

出力:

```
None.
```

- API の詳細については、AWS CLI コマンドリファレンスの「SetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
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
}



// 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
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「SetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

カスタム KMS キーを使用してサーバー側の暗号化 (SSE) を使用するように Amazon SQS を設定します。

```
   public static void addEncryption(String queueName, String kmsMasterKeyAlias)
{
       SqsClient sqsClient = SqsClient.create();

       GetQueueUrlRequest urlRequest = GetQueueUrlRequest.builder()
               .queueName(queueName)
               .build();

       GetQueueUrlResponse getQueueUrlResponse;
       try {
           getQueueUrlResponse = sqsClient.getQueueUrl(urlRequest);
       } catch (QueueDoesNotExistException e) {
           LOGGER.error(e.getMessage(), e);
           throw new RuntimeException(e);
       }
       String queueUrl = getQueueUrlResponse.queueUrl();


       Map<QueueAttributeName, String> attributes = Map.of(
               QueueAttributeName.KMS_MASTER_KEY_ID, kmsMasterKeyAlias,
               QueueAttributeName.KMS_DATA_KEY_REUSE_PERIOD_SECONDS, "140" //
Set the data key reuse period to 140 seconds.
       );                                                                  //
This is how long SQS can reuse the data key before requesting a new one from
KMS.

       SetQueueAttributesRequest attRequest =
SetQueueAttributesRequest.builder()
               .queueUrl(queueUrl)
               .attributes(attributes)
               .build();
       try {
           sqsClient.setQueueAttributes(attRequest);
           LOGGER.info("The attributes have been applied to {}", queueName);
       } catch (InvalidAttributeNameException | InvalidAttributeValueException
e) {
           LOGGER.error(e.getMessage(), e);
           throw new RuntimeException(e);
       } finally {
           sqsClient.close();
```

```
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「SetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

ロングポーリングを使用するように Amazon SQS キューを設定します。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
```

```
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

デッドレターキューを設定します。

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });
```

```
  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- API の詳細については、AWS SDK for JavaScript API リファレンスの「SetQueueAttributes」を参照してください。

PowerShell

Tools for PowerShell V4

例 1: この例では、キューを SNS トピックにサブスクライブするポリシーを設定する方法を示します。メッセージをトピックに発行すると、メッセージはサブスクライブしたキューに送信されます。

```
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName
 "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"
{
  "Version": "2008-10-17",
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
      {
      "Sid": "1",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
```

```
            }
        }
      }
    ]
}
"@

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

例 2: この例では、指定したキューに指定した属性を設定します。

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 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 SetQueueAttributes」を参照してください。

Tools for PowerShell V5

例 1: この例では、キューを SNS トピックにサブスクライブするポリシーを設定する方法を示します。メッセージをトピックに発行すると、メッセージはサブスクライブしたキューに送信されます。

```
# create the queue and topic to be associated
$qurl = New-SQSQueue -QueueName "myQueue"
$topicarn = New-SNSTopic -Name "myTopic"

# get the queue ARN to inject into the policy; it will be returned
# in the output's QueueARN member but we need to put it into a variable
# so text expansion in the policy string takes effect
$qarn = (Get-SQSQueueAttribute -QueueUrl $qurl -AttributeName
 "QueueArn").QueueARN

# construct the policy and inject arns
$policy = @"
{
  "Version": "2008-10-17",
  "Id": "$qarn/SQSPOLICY",
  "Statement": [
      {
      "Sid": "1",
      "Effect": "Allow",
```

```
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "$qarn",
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "$topicarn"
          }
      }
    }
  ]
}
"@

# set the policy
Set-SQSQueueAttribute -QueueUrl $qurl -Attribute @{ Policy=$policy }
```

例 2: この例では、指定したキューに指定した属性を設定します。

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- 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 SetQueueAttributes」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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
import AWSSQS

        let config = try await SQSClient.SQSClientConfiguration(region: region)
        let sqsClient = SQSClient(config: config)

        do {
            _ = try await sqsClient.setQueueAttributes(
```

```
                input: SetQueueAttributesInput(
                    attributes: [
                        "MaximumMessageSize": "\(maxSize)"
                    ],
                    queueUrl: url
                )
            )
        } catch _ as AWSSQS.InvalidAttributeValue {
            print("Invalid maximum message size: \(maxSize) kB.")
        }
```

- API の詳細については、 AWS SDK for Swift API リファレンスの「SetQueueAttributes」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 SQS 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 SQS のシナリオ AWS SDKs

次のコード例は、 AWS SDKs を使用して Amazon SQS で一般的なシナリオを実装する方法を示しています。これらのシナリオは、Amazon SQS 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、特定のタスクを達成する方法を示しています。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

- Amazon SQS を使用してメッセージを送受信するウェブアプリケーションを作成する
- ステップ関数でメッセンジャーアプリケーションを作成する
- 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 で Amazon S3 を使用して大規模な Amazon SQS Amazon S3 メッセージを管理する
- AWS SDK を使用して Amazon S3 イベント通知を受信して処理する

- 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
- AWS SDK を使用して Amazon SQS でメッセージのバッチを送受信する
- Message AWS Processing Framework for .NET を使用して Amazon SQS メッセージを公開および受信する
- Amazon SQS Java Messaging Library を使用して Amazon SQS の Java Message Service (JMS) インターフェイスを操作する
- AWS SDK を使用してキュータグと Amazon SQS を操作する

Amazon SQS を使用してメッセージを送受信するウェブアプリケーションを作成する

次のコード例は、Amazon SQS でメッセージングアプリケーションを作成する方法を示しています。

Java

SDK for Java 2.x

Amazon SQS API を使用して、メッセージを送受信する Spring REST API を開発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

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK for Kotlin

Amazon SQS API を使用して、メッセージを送受信する Spring REST API を開発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

- Amazon Comprehend
- 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 SQS 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。

ステップ関数でメッセンジャーアプリケーションを作成する

次のコード例は、データベーステーブルからメッセージレコードを取得する AWS Step Functions メッセンジャーアプリケーションを作成する方法を示しています。

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3) で を使用して AWS Step Functions 、Amazon DynamoDB テーブルからメッセージレコードを取得し、Amazon Simple Queue Service (Amazon SQS) で送信するメッセンジャーアプリケーションを作成する方法を示します。ステートマシンは AWS Lambda 関数と統合して、未送信メッセージがないかデータベースをスキャンします。

- Amazon DynamoDB テーブルからメッセージレコードを取得および更新するステートマシンを作成します。
- ステートマシンの定義を更新して、Amazon Simple Queue Service (Amazon SQS) にもメッセージを送信します。
- ステートマシンの実行を開始および停止します。
- サービス統合を使用して、ステートマシンから Lambda、DynamoDB、および Amazon SQS に接続します。

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
- Lambda
- Amazon SQS
- Step Functions

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 SQS 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 SQS 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。AWS コード例リポジトリ で全く同じ例を見つけて、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。AWS コード例リポジトリ で全く同じ例を見つけて、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。AWS コード例リポジトリ で全く同じ例を見つけて、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_snstopicattrsmap_w=>tt_topicattributesmap.
    DATA ls_tpc_attributes TYPE /aws1/
cl_snstopicattrsmap_w=>ts_topicattributesmap_maprow.
    ls_tpc_attributes-key = 'FifoTopic'.
    ls_tpc_attributes-value = NEW /aws1/cl_snstopicattrsmap_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_snssubscriptionlmte00.
        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 SQS 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 SQS 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 で Amazon S3 を使用して大規模な Amazon SQS Amazon S3 メッセージを管理する

次のコード例は、Amazon SQS 拡張クライアントライブラリを使用して大規模な Amazon SQS メッセージを操作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。完全な例を見つけて、AWS コード例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 com.amazon.sqs.javamessaging.AmazonSQSExtendedClient;
import com.amazon.sqs.javamessaging.ExtendedClientConfiguration;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.BucketLifecycleConfiguration;
import software.amazon.awssdk.services.s3.model.CreateBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteBucketRequest;
import software.amazon.awssdk.services.s3.model.DeleteObjectRequest;
import software.amazon.awssdk.services.s3.model.ExpirationStatus;
import software.amazon.awssdk.services.s3.model.LifecycleExpiration;
import software.amazon.awssdk.services.s3.model.LifecycleRule;
import software.amazon.awssdk.services.s3.model.LifecycleRuleFilter;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsRequest;
import software.amazon.awssdk.services.s3.model.ListObjectVersionsResponse;
import software.amazon.awssdk.services.s3.model.ListObjectsV2Request;
```

```
import software.amazon.awssdk.services.s3.model.ListObjectsV2Response;
import
 software.amazon.awssdk.services.s3.model.PutBucketLifecycleConfigurationRequest;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.CreateQueueResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;

import java.util.Arrays;
import java.util.List;
import java.util.UUID;

/**
 * Example of using Amazon SQS Extended Client Library for Java 2.x.
 */
public class SqsExtendedClientExample {
    private static final Logger logger =
 LoggerFactory.getLogger(SqsExtendedClientExample.class);

    private String s3BucketName;
    private String queueUrl;
    private final String queueName;
    private final S3Client s3Client;
    private final SqsClient sqsExtendedClient;
    private final int messageSize;

    /**
     * Constructor with default clients and message size.
     */
    public SqsExtendedClientExample() {
        this(S3Client.create(), 300000);
    }

    /**
     * Constructor with custom S3 client and message size.
     *
     * @param s3Client The S3 client to use
     * @param messageSize The size of the test message to create
     */
```

```
    public SqsExtendedClientExample(S3Client s3Client, int messageSize) {
        this.s3Client = s3Client;
        this.messageSize = messageSize;

        // Generate a unique bucket name.
        this.s3BucketName = UUID.randomUUID() + "-" +
                DateTimeFormat.forPattern("yyMMdd-hhmmss").print(new DateTime());

        // Generate a unique queue name.
        this.queueName = "MyQueue-" + UUID.randomUUID();

        // Configure the SQS extended client.
        final ExtendedClientConfiguration extendedClientConfig = new
ExtendedClientConfiguration()
                .withPayloadSupportEnabled(s3Client, s3BucketName);

        this.sqsExtendedClient = new
AmazonSQSExtendedClient(SqsClient.builder().build(), extendedClientConfig);
    }

    public static void main(String[] args) {
        SqsExtendedClientExample example = new SqsExtendedClientExample();
        try {
            example.setup();
            example.sendAndReceiveMessage();
        } finally {
            example.cleanup();
        }
    }

    /**
     * Send a large message and receive it back.
     *
     * @return The received message
     */
    public Message sendAndReceiveMessage() {
        try {
            // Create a large message.
            char[] chars = new char[messageSize];
            Arrays.fill(chars, 'x');
            String largeMessage = new String(chars);

            // Send the message.
```

```
            final SendMessageRequest sendMessageRequest =
SendMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageBody(largeMessage)
                    .build();

            sqsExtendedClient.sendMessage(sendMessageRequest);
            logger.info("Sent message of size: {}", largeMessage.length());

            // Receive and return the message.
            final ReceiveMessageResponse receiveMessageResponse =
sqsExtendedClient.receiveMessage(
                    ReceiveMessageRequest.builder().queueUrl(queueUrl).build());

            List<Message> messages = receiveMessageResponse.messages();
            if (messages.isEmpty()) {
                throw new RuntimeException("No messages received");
            }

            Message message = messages.getFirst();
            logger.info("\nMessage received.");
            logger.info("  ID: {}", message.messageId());
            logger.info("  Receipt handle: {}", message.receiptHandle());
            logger.info("  Message body size: {}", message.body().length());
            logger.info("  Message body (first 5 characters): {}",
message.body().substring(0, 5));

            return message;
        } catch (RuntimeException e) {
            logger.error("Error during message processing: {}", e.getMessage(),
e);
            throw e;
        }
    }
```

- 詳細については、「AWS SDK for Java 2.x 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 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。
  - CreateBucket
  - PutBucketLifecycleConfiguration
  - ReceiveMessage

- SendMessage

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 SQS 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 イベント通知を受信して処理する

次のコード例は、オブジェクト指向の方法で S3 イベント通知を操作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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

この例は、Amazon SQS を使用して S3 通知イベントを処理する方法を示しています。

```
    /**
     * This method receives S3 event notifications by using an SqsAsyncClient.
     * After the client receives the messages it deserializes the JSON payload
 and logs them. It uses
     * the S3EventNotification class (part of the S3 event notification API for
 Java) to deserialize
     * the JSON payload and access the messages in an object-oriented way.
     *
     * @param queueUrl The URL of the AWS SQS queue that receives the S3 event
 notifications.
     * @see <a href="https://sdk.amazonaws.com/java/api/latest/software/amazon/
awssdk/eventnotifications/s3/model/package-summary.html">S3EventNotification
 API</a>.
     * <p>
     * To use S3 event notification serialization/deserialization to objects, add
 the following
     * dependency to your Maven pom.xml file.
     * <dependency>
     * <groupId>software.amazon.awssdk</groupId>
```

```
    * <artifactId>s3-event-notifications</artifactId>
    * <version><LATEST></version>
    * </dependency>
    * <p>
    * The S3 event notification API became available with version 2.25.11 of the
Java SDK.
    * <p>
    * This example shows the use of the API with AWS SQS, but it can be used to
process S3 event notifications
    * in AWS SNS or AWS Lambda as well.
    * <p>
    * Note: The S3EventNotification class does not work with messages routed
through AWS EventBridge.
    */
   static void processS3Events(String bucketName, String queueUrl, String
queueArn) {
       try {
           // Configure the bucket to send Object Created and Object Tagging
notifications to an existing SQS queue.
           s3Client.putBucketNotificationConfiguration(b -> b
                   .notificationConfiguration(ncb -> ncb
                           .queueConfigurations(qcb -> qcb
                                   .events(Event.S3_OBJECT_CREATED,
Event.S3_OBJECT_TAGGING)
                                   .queueArn(queueArn)))
                           .bucket(bucketName)
           ).join();

           triggerS3EventNotifications(bucketName);
           // Wait for event notifications to propagate.
           Thread.sleep(Duration.ofSeconds(5).toMillis());

           boolean didReceiveMessages = true;
           while (didReceiveMessages) {
               // Display the number of messages that are available in the
queue.
               sqsClient.getQueueAttributes(b -> b
                               .queueUrl(queueUrl)

.attributeNames(QueueAttributeName.APPROXIMATE_NUMBER_OF_MESSAGES)
                       ).thenAccept(attributeResponse ->
                               logger.info("Approximate number of messages in
the queue: {}",
```

```
attributeResponse.attributes().get(QueueAttributeName.APPROXIMATE_NUMBER_OF_MESSAGES)))
                        .join();

                // Receive the messages.
                ReceiveMessageResponse response = sqsClient.receiveMessage(b -> b
                        .queueUrl(queueUrl)
                ).get();
                logger.info("Count of received messages: {}",
response.messages().size());
                didReceiveMessages = !response.messages().isEmpty();

                // Create a collection to hold the received message for deletion
                // after we log the messages.
                HashSet<DeleteMessageBatchRequestEntry> messagesToDelete = new
HashSet<>();
                // Process each message.
                response.messages().forEach(message -> {
                    logger.info("Message id: {}", message.messageId());
                    // Deserialize JSON message body to a S3EventNotification
object
                    // to access messages in an object-oriented way.
                    S3EventNotification event =
S3EventNotification.fromJson(message.body());

                    // Log the S3 event notification record details.
                    if (event.getRecords() != null) {
                        event.getRecords().forEach(record -> {
                            String eventName = record.getEventName();
                            String key = record.getS3().getObject().getKey();
                            logger.info(record.toString());
                            logger.info("Event name is {} and key is {}",
eventName, key);
                        });
                    }
                    // Add logged messages to collection for batch deletion.
                    messagesToDelete.add(DeleteMessageBatchRequestEntry.builder()
                            .id(message.messageId())
                            .receiptHandle(message.receiptHandle())
                            .build());
                });
                // Delete messages.
                if (!messagesToDelete.isEmpty()) {
```

```
sqsClient.deleteMessageBatch(DeleteMessageBatchRequest.builder()
                            .queueUrl(queueUrl)
                            .entries(messagesToDelete)
                            .build()
                    ).join();
                }
            } // End of while block.
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
    }
```

- 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。
  - DeleteMessageBatch
  - GetQueueAttributes
  - PutBucketNotificationConfiguration
  - ReceiveMessage

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 SQS 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。AWS コード例リポジトリ で全く同じ例を見つけて、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の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\nYou can select from several options for
configuring the topic and the subscriptions for the 2 queues." +
                          $"\r\nYou 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 messageGroupId = 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($"\t{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:" +
                              $"\n\t{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="amazonSQS">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
  - 公開
  - 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。AWS コード例リポジトリ で全く同じ例を見つけて、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 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;
            }
```

```
        }

        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
- 公開
- 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。AWS コード例リポジトリ で全く同じ例を見つけて、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の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
 messsage? (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 susbscribe 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
  - 公開
  - 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。AWS コード例リポジトリ で全く同じ例を見つけて、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)
                    .maxNumberOfMessages(5)
                    .build();
```

```
                return
sqsClient.receiveMessage(receiveMessageRequest).messages();
            } else {
                // We know there are filters on the message.
                ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
                    .queueUrl(queueUrl)
                    .messageAttributeNames(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
- 公開
- 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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

これは、このシナリオのエントリポイントです。

```
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();
};
```

前述のコードは、必要な依存関係を提供し、シナリオを開始します。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
  - 公開
  - 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。AWS コード例リポジトリ で全く同じ例を見つけて、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
  - 公開
  - 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。AWS コード例リポジトリ で全く同じ例を見つけて、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.\n
              """
        )

        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
  - 公開
  - 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 SQS 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 SQS でメッセージのバッチを送受信する

次のコード例は、以下を実行する方法を示しています。

- Amazon SQS キューを作成します。
- キューにバッチメッセージを送信します。
- キューからバッチメッセージを受信します。
- キューからバッチメッセージを受信します。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

次の例に示すように、 で 2 つの異なるアプローチを使用して Amazon SQS でバッチメッセージオペレーションを処理できます AWS SDK for Java 2.x。

SendRecvBatch.java は明示的なバッチオペレーションを使用します。メッセージバッチを手動で作成し、 `sendMessageBatch()` と `deleteMessageBatch()`を直接呼び出します。また、失敗したメッセージを含むバッチレスポンスも処理します。このアプローチにより、バッチサイズ設定とエラー処理を完全に制御できます。ただし、バッチ処理ロジックを管理するには、より多くのコードが必要です。

SimpleProducerConsumer.java は、自動リクエストバッチ処理に高レベル`SqsAsyncBatchManager`ライブラリを使用します。標準クライアントと同じメソッド署名を使用して、個別の `sendMessage()`および `deleteMessage()`呼び出しを行います。SDK はこれらの呼び出しを自動的にバッファし、バッチオペレーションとして送信します。このアプローチでは、バッチ処理のパフォーマンス上の利点を提供しながら、コードの変更を最小限に抑える必要があります。

バッチ構成とエラー処理をきめ細かく制御する必要がある場合は、明示的なバッチ処理を使用します。コードの変更を最小限に抑えてパフォーマンスを最適化する場合は、自動バッチ処理を使用します。

SendRecvBatch.java - メッセージで明示的なバッチオペレーションを使用します。

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.BatchResultErrorEntry;
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.DeleteMessageBatchResponse;
import software.amazon.awssdk.services.sqs.model.DeleteMessageBatchResultEntry;
import software.amazon.awssdk.services.sqs.model.DeleteQueueRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.MessageAttributeValue;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchRequestEntry;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageBatchResultEntry;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
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
```

```
 */

/**
 * This code demonstrates basic message operations in Amazon Simple Queue Service
 (Amazon SQS).
 */

public class SendRecvBatch {
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SendRecvBatch.class);
    private static final SqsClient sqsClient = SqsClient.create();


    public static void main(String[] args) {
        usageDemo();
    }
    /**
     * Send a batch of messages in a single request to an SQS queue.
     * This request may return overall success even when some messages were not
 sent.
     * The caller must inspect the Successful and Failed lists in the response
 and
     * resend any failed messages.
     *
     * @param queueUrl  The URL of the queue to receive the messages.
     * @param messages  The messages to send to the queue. Each message contains
 a body and attributes.
     * @return The response from SQS that contains the list of successful and
 failed messages.
     */
    public static SendMessageBatchResponse sendMessages(
            String queueUrl, List<MessageEntry> messages) {

        try {
            List<SendMessageBatchRequestEntry> entries = new ArrayList<>();

            for (int i = 0; i < messages.size(); i++) {
                MessageEntry message = messages.get(i);
                entries.add(SendMessageBatchRequestEntry.builder()
                        .id(String.valueOf(i))
                        .messageBody(message.getBody())
                        .messageAttributes(message.getAttributes())
                        .build());
            }
```

```
            SendMessageBatchRequest sendBatchRequest =
SendMessageBatchRequest.builder()
                    .queueUrl(queueUrl)
                    .entries(entries)
                    .build();

            SendMessageBatchResponse response =
sqsClient.sendMessageBatch(sendBatchRequest);

            if (!response.successful().isEmpty()) {
                for (SendMessageBatchResultEntry resultEntry :
response.successful()) {
                    LOGGER.info("Message sent: {}: {}", resultEntry.messageId(),

messages.get(Integer.parseInt(resultEntry.id())).getBody());
                }
            }

            if (!response.failed().isEmpty()) {
                for (BatchResultErrorEntry errorEntry : response.failed()) {
                    LOGGER.warn("Failed to send: {}: {}", errorEntry.id(),

messages.get(Integer.parseInt(errorEntry.id())).getBody());
                }
            }

            return response;

        } catch (SqsException e) {
            LOGGER.error("Send messages failed to queue: {}", queueUrl, e);
            throw e;
        }
    }

    /**
     * Receive a batch of messages in a single request from an SQS queue.
     *
     * @param queueUrl   The URL of the queue from which to receive messages.
     * @param maxNumber  The maximum number of messages to receive (capped at 10
by SQS).
     *                   The actual number of messages received might be less.
     * @param waitTime   The maximum time to wait (in seconds) before returning.
When
```

```
    *                   this number is greater than zero, long polling is used.
This
    *                   can result in reduced costs and fewer false empty
responses.
    * @return The list of Message objects received. These each contain the body
    *         of the message and metadata and custom attributes.
    */
   public static List<Message> receiveMessages(String queueUrl, int maxNumber,
int waitTime) {
       try {
           ReceiveMessageRequest receiveRequest =
ReceiveMessageRequest.builder()
                   .queueUrl(queueUrl)
                   .maxNumberOfMessages(maxNumber)
                   .waitTimeSeconds(waitTime)
                   .messageAttributeNames("All")
                   .build();

           List<Message> messages =
sqsClient.receiveMessage(receiveRequest).messages();

           for (Message message : messages) {
               LOGGER.info("Received message: {}: {}", message.messageId(),
message.body());
           }

           return messages;

       } catch (SqsException e) {
           LOGGER.error("Couldn't receive messages from queue: {}", queueUrl,
e);
           throw e;
       }
   }

   /**
    * Delete a batch of messages from a queue in a single request.
    *
    * @param queueUrl  The URL of the queue from which to delete the messages.
    * @param messages  The list of messages to delete.
    * @return The response from SQS that contains the list of successful and
failed
    *         message deletions.
    */
```

```
   public static DeleteMessageBatchResponse deleteMessages(String queueUrl,
List<Message> messages) {
       try {
           List<DeleteMessageBatchRequestEntry> entries = new ArrayList<>();

           for (int i = 0; i < messages.size(); i++) {
               entries.add(DeleteMessageBatchRequestEntry.builder()
                       .id(String.valueOf(i))
                       .receiptHandle(messages.get(i).receiptHandle())
                       .build());
           }

           DeleteMessageBatchRequest deleteRequest =
DeleteMessageBatchRequest.builder()
                   .queueUrl(queueUrl)
                   .entries(entries)
                   .build();

           DeleteMessageBatchResponse response =
sqsClient.deleteMessageBatch(deleteRequest);

           if (!response.successful().isEmpty()) {
               for (DeleteMessageBatchResultEntry resultEntry :
response.successful()) {
                   LOGGER.info("Deleted {}",
messages.get(Integer.parseInt(resultEntry.id())).receiptHandle());
               }
           }

           if (!response.failed().isEmpty()) {
               for (BatchResultErrorEntry errorEntry : response.failed()) {
                   LOGGER.warn("Could not delete {}",
messages.get(Integer.parseInt(errorEntry.id())).receiptHandle());
               }
           }

           return response;

       } catch (SqsException e) {
           LOGGER.error("Couldn't delete messages from queue {}", queueUrl, e);
           throw e;
       }
   }
```

```
    /**
     * Helper class to represent a message with body and attributes.
     */
    public static class MessageEntry {
        private final String body;
        private final Map<String, MessageAttributeValue> attributes;

        public MessageEntry(String body, Map<String, MessageAttributeValue>
attributes) {
            this.body = body;
            this.attributes = attributes != null ? attributes : new HashMap<>();
        }

        public String getBody() {
            return body;
        }

        public Map<String, MessageAttributeValue> getAttributes() {
            return attributes;
        }
    }

    /**
     * Shows how to:
     * * Read the lines from a file and send the lines in
     *   batches of 10 as messages to a queue.
     * * Receive the messages in batches until the queue is empty.
     * * Reassemble the lines of the file and verify they match the original
file.
     */
    public static void usageDemo() {
        LOGGER.info("-".repeat(88));
        LOGGER.info("Welcome to the Amazon Simple Queue Service (Amazon SQS)
demo!");
        LOGGER.info("-".repeat(88));

        String queueUrl = null;
        try {
            // Create a queue for the demo.
            String queueName = "sqs-usage-demo-message-wrapper-" +
System.currentTimeMillis();
            CreateQueueRequest createRequest = CreateQueueRequest.builder()
                    .queueName(queueName)
                    .build();
```

```
            queueUrl = sqsClient.createQueue(createRequest).queueUrl();
            LOGGER.info("Created queue: {}", queueUrl);

            try (InputStream inputStream =
 SendRecvBatch.class.getResourceAsStream("/log4j2.xml");
                 BufferedReader reader = new BufferedReader(new
 InputStreamReader(inputStream))) {

                List<String> lines = reader.lines().toList();

                // Send file lines in batches.
                int batchSize = 10;
                LOGGER.info("Sending file lines in batches of {} as messages.",
 batchSize);

                for (int i = 0; i < lines.size(); i += batchSize) {
                    List<MessageEntry> messageBatch = new ArrayList<>();

                    for (int j = i; j < Math.min(i + batchSize, lines.size()); j+
+) {
                        String line = lines.get(j);
                        if (line == null || line.trim().isEmpty()) {
                            continue; // Skip empty lines.
                        }

                        Map<String, MessageAttributeValue> attributes = new
 HashMap<>();
                        attributes.put("line", MessageAttributeValue.builder()
                                .dataType("String")
                                .stringValue(String.valueOf(j))
                                .build());

                        messageBatch.add(new MessageEntry(lines.get(j),
 attributes));
                    }

                    sendMessages(queueUrl, messageBatch);
                    System.out.print(".");
                    System.out.flush();
                }

                LOGGER.info("\nDone. Sent {} messages.", lines.size());

                // Receive and process messages.
```

```
                LOGGER.info("Receiving, handling, and deleting messages in
batches of {}.", batchSize);
                String[] receivedLines = new String[lines.size()];
                boolean moreMessages = true;

                while (moreMessages) {
                    List<Message> receivedMessages = receiveMessages(queueUrl,
batchSize, 5);

                    for (Message message : receivedMessages) {
                        int lineNumber =
Integer.parseInt(message.messageAttributes().get("line").stringValue());
                        receivedLines[lineNumber] = message.body();
                    }

                    if (!receivedMessages.isEmpty()) {
                        deleteMessages(queueUrl, receivedMessages);
                    } else {
                        moreMessages = false;
                    }
                }

                LOGGER.info("\nDone.");

                // Verify that all lines were received correctly.
                boolean allLinesMatch = true;
                for (int i = 0; i < lines.size(); i++) {
                    String originalLine = lines.get(i);
                    String receivedLine = receivedLines[i] == null ? "" :
receivedLines[i];

                    if (!originalLine.equals(receivedLine)) {
                        allLinesMatch = false;
                        break;
                    }
                }

                if (allLinesMatch) {
                    LOGGER.info("Successfully reassembled all file lines!");
                } else {
                    LOGGER.info("Uh oh, some lines were missed!");
                }
            }
        } catch (SqsException e) {
```

```
            LOGGER.error("SQS operation failed", e);
        } catch (RuntimeException | IOException e) {
            LOGGER.error("Unexpected runtime error during demo", e);
        } finally {
            // Clean up by deleting the queue if it was created.
            if (queueUrl != null) {
                try {
                    DeleteQueueRequest deleteQueueRequest =
 DeleteQueueRequest.builder()
                            .queueUrl(queueUrl)
                            .build();
                    sqsClient.deleteQueue(deleteQueueRequest);
                    LOGGER.info("Deleted queue: {}", queueUrl);
                } catch (SqsException e) {
                    LOGGER.error("Failed to delete queue: {}", queueUrl, e);
                }
            }
        }

        LOGGER.info("Thanks for watching!");
        LOGGER.info("-".repeat(88));
    }
 }
```

SimpleProducerConsumer.java - メッセージの自動バッチ処理を使用します。

```
package com.example.sqs;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsAsyncClient;
import software.amazon.awssdk.services.sqs.batchmanager.SqsAsyncBatchManager;
import software.amazon.awssdk.services.sqs.model.DeleteMessageRequest;
import software.amazon.awssdk.services.sqs.model.DeleteMessageResponse;
import software.amazon.awssdk.services.sqs.model.GetQueueUrlRequest;
import software.amazon.awssdk.services.sqs.model.Message;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageRequest;
import software.amazon.awssdk.services.sqs.model.ReceiveMessageResponse;
import software.amazon.awssdk.services.sqs.model.SendMessageRequest;
import software.amazon.awssdk.services.sqs.model.SendMessageResponse;
import software.amazon.awssdk.core.exception.SdkException;
```

```
import java.math.BigInteger;
import java.util.List;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Demonstrates the AWS SDK for Java 2.x Automatic Request Batching API for
 Amazon SQS.
 *
 * This example showcases the high-level SqsAsyncBatchManager library that
 provides
 * efficient batching and buffering for SQS operations. The batch manager offers
 * methods that directly mirror SqsAsyncClient methods—sendMessage,
 changeMessageVisibility,
 * deleteMessage, and receiveMessage—making it a drop-in replacement with minimal
 code changes.
 *
 * Key features of the SqsAsyncBatchManager:
 * - Automatic batching: The SDK automatically buffers individual requests and
 sends them
 *   as batches when maxBatchSize (default: 10) or sendRequestFrequency (default:
 200ms)
 *   thresholds are reached
 * - Familiar API: Method signatures match SqsAsyncClient exactly, requiring no
 learning curve
 * - Background optimization: The batch manager maintains internal buffers and
 handles
 *   batching logic transparently
 * - Asynchronous operations: All methods return CompletableFuture for non-
blocking execution
 *
 * Performance benefits demonstrated:
 * - Reduced API calls: Multiple individual requests are consolidated into single
 batch operations
 * - Lower costs: Fewer API calls result in reduced SQS charges
 * - Higher throughput: Batch operations process more messages per second
 * - Efficient resource utilization: Fewer network round trips and better
 connection reuse
 *
 * This example compares:
```

```
 * 1. Single-message operations using SqsAsyncClient directly
 * 2. Batch operations using SqsAsyncBatchManager with identical method calls
 *
 * Usage patterns:
 * - Set batch size to 1 to use SqsAsyncClient for baseline performance
 measurement
 * - Set batch size > 1 to use SqsAsyncBatchManager for optimized batch
 processing
 * - Monitor real-time throughput metrics to observe performance improvements
 *
 * Prerequisites:
 * - AWS SDK for Java 2.x version 2.28.0 or later
 * - An existing SQS queue
 * - Valid AWS credentials configured
 *
 * The program displays real-time metrics showing the dramatic performance
 difference
 * between individual operations and automatic batching.
 */
public class SimpleProducerConsumer {

    // The maximum runtime of the program.
    private final static int MAX_RUNTIME_MINUTES = 60;
    private final static Logger log =
 LoggerFactory.getLogger(SimpleProducerConsumer.class);

    /**
     * Runs the SQS batching demonstration with user-configured parameters.
     *
     * Prompts for queue name, thread counts, batch size, message size, and
 runtime.
     * Creates producer and consumer threads to demonstrate batching performance.
     *
     * @param args command line arguments (not used)
     * @throws InterruptedException if thread operations are interrupted
     */
    public static void main(String[] args) throws InterruptedException {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the number of producers: ");
```

```
        final int producerCount = input.nextInt();

        System.out.print("Enter the number of consumers: ");
        final int consumerCount = input.nextInt();

        System.out.print("Enter the number of messages per batch: ");
        final int batchSize = input.nextInt();

        System.out.print("Enter the message size in bytes: ");
        final int messageSizeByte = input.nextInt();

        System.out.print("Enter the run time in minutes: ");
        final int runTimeMinutes = input.nextInt();

        // Create SQS async client and batch manager for all operations.
        // The SqsAsyncBatchManager is created from the SqsAsyncClient using the
        // batchManager() factory method, which provides default batching
configuration.
        // This high-level library automatically handles request buffering and
batching
        // while maintaining the same method signatures as SqsAsyncClient.
        final SqsAsyncClient sqsAsyncClient = SqsAsyncClient.create();
        final SqsAsyncBatchManager batchManager = sqsAsyncClient.batchManager();

        final String queueUrl =
sqsAsyncClient.getQueueUrl(GetQueueUrlRequest.builder()
                .queueName(queueName)
                .build()).join().queueUrl();

        // The flag used to stop producer, consumer, and monitor threads.
        final AtomicBoolean stop = new AtomicBoolean(false);

        // Start the producers.
        final AtomicInteger producedCount = new AtomicInteger();
        final Thread[] producers = new Thread[producerCount];
        for (int i = 0; i < producerCount; i++) {
            if (batchSize == 1) {
                producers[i] = new Producer(sqsAsyncClient, queueUrl,
messageSizeByte,
                        producedCount, stop);
            } else {
                producers[i] = new BatchProducer(batchManager, queueUrl,
batchSize,
                        messageSizeByte, producedCount, stop);
```

```
            }
            producers[i].start();
        }

        // Start the consumers.
        final AtomicInteger consumedCount = new AtomicInteger();
        final Thread[] consumers = new Thread[consumerCount];
        for (int i = 0; i < consumerCount; i++) {
            if (batchSize == 1) {
                consumers[i] = new Consumer(sqsAsyncClient, queueUrl,
consumedCount, stop);
            } else {
                consumers[i] = new BatchConsumer(batchManager, queueUrl,
batchSize,
                        consumedCount, stop);
            }
            consumers[i].start();
        }

        // Start the monitor thread.
        final Thread monitor = new Monitor(producedCount, consumedCount, stop);
        monitor.start();

        // Wait for the specified amount of time then stop.
        Thread.sleep(TimeUnit.MINUTES.toMillis(Math.min(runTimeMinutes,
                MAX_RUNTIME_MINUTES)));
        stop.set(true);

        // Join all threads.
        for (int i = 0; i < producerCount; i++) {
            producers[i].join();
        }

        for (int i = 0; i < consumerCount; i++) {
            consumers[i].join();
        }

        monitor.interrupt();
        monitor.join();

        // Close resources
        batchManager.close();
        sqsAsyncClient.close();
    }
```

```
    /**
     * Creates a random string of approximately the specified size in bytes.
     *
     * @param sizeByte the target size in bytes for the generated string
     * @return a random string encoded in base-32
     */
    private static String makeRandomString(int sizeByte) {
        final byte[] bs = new byte[(int) Math.ceil(sizeByte * 5 / 8)];
        new Random().nextBytes(bs);
        bs[0] = (byte) ((bs[0] | 64) & 127);
        return new BigInteger(bs).toString(32);
    }

    /**
     * Sends messages individually using SqsAsyncClient for baseline performance
measurement.
     *
     * This producer demonstrates traditional single-message operations without
batching.
     * Each sendMessage() call results in a separate API request to SQS,
providing
     * a performance baseline for comparison with the batch operations.
     *
     * The sendMessage() method signature is identical to
SqsAsyncBatchManager.sendMessage(),
     * showing how the high-level batching library maintains API compatibility
while
     * adding automatic optimization behind the scenes.
     */
    private static class Producer extends Thread {
        final SqsAsyncClient sqsAsyncClient;
        final String queueUrl;
        final AtomicInteger producedCount;
        final AtomicBoolean stop;
        final String theMessage;

        /**
         * Creates a producer thread for single-message operations.
         *
         * @param sqsAsyncClient the SQS client for sending messages
         * @param queueUrl the URL of the target queue
         * @param messageSizeByte the size of messages to generate
         * @param producedCount shared counter for tracking sent messages
```

```
         * @param stop shared flag to signal thread termination
         */
        Producer(SqsAsyncClient sqsAsyncClient, String queueUrl, int
messageSizeByte,
                AtomicInteger producedCount, AtomicBoolean stop) {
            this.sqsAsyncClient = sqsAsyncClient;
            this.queueUrl = queueUrl;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /**
         * Continuously sends messages until the stop flag is set.
         *
         * Uses SqsAsyncClient.sendMessage() directly, resulting in one API call
per message.
         * This approach provides baseline performance metrics for comparison
with batching.
         * Each call blocks until the individual message is sent, demonstrating
traditional
         * one-request-per-operation behavior.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    sqsAsyncClient.sendMessage(SendMessageRequest.builder()
                            .queueUrl(queueUrl)
                            .messageBody(theMessage)
                            .build()).join();
                    producedCount.incrementAndGet();
                }
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                // Handle both SdkException and CompletionException from async
operations.
                // If this unlikely condition occurs, stop.
                log.error("Producer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
```

```
    * Sends messages using SqsAsyncBatchManager for automatic request batching
and optimization.
    *
    * This producer demonstrates the AWS SDK for Java 2.x high-level batching
library.
    * The SqsAsyncBatchManager automatically buffers individual sendMessage()
calls and
    * sends them as batches when thresholds are reached:
    * - maxBatchSize: Maximum 10 messages per batch (default)
    * - sendRequestFrequency: 200ms timeout before sending partial batches
(default)
    *
    * Key advantages of the batching approach:
    * - Identical API: batchManager.sendMessage() has the same signature as
sqsAsyncClient.sendMessage()
    * - Automatic optimization: No code changes needed to benefit from batching
    * - Transparent buffering: The SDK handles batching logic internally
    * - Reduced API calls: Multiple messages sent in single batch requests
    * - Lower costs: Fewer API calls result in reduced SQS charges
    * - Higher throughput: Batch operations process significantly more messages
per second
    */
   private static class BatchProducer extends Thread {
       final SqsAsyncBatchManager batchManager;
       final String queueUrl;
       final int batchSize;
       final AtomicInteger producedCount;
       final AtomicBoolean stop;
       final String theMessage;

       /**
        * Creates a producer thread for batch operations.
        *
        * @param batchManager the batch manager for efficient message sending
        * @param queueUrl the URL of the target queue
        * @param batchSize the number of messages to send per batch
        * @param messageSizeByte the size of messages to generate
        * @param producedCount shared counter for tracking sent messages
        * @param stop shared flag to signal thread termination
        */
       BatchProducer(SqsAsyncBatchManager batchManager, String queueUrl, int
batchSize,
                     int messageSizeByte, AtomicInteger producedCount,
                     AtomicBoolean stop) {
```

```
            this.batchManager = batchManager;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.producedCount = producedCount;
            this.stop = stop;
            this.theMessage = makeRandomString(messageSizeByte);
        }

        /**
         * Continuously sends batches of messages using the high-level batching
library.
         *
         * Notice how batchManager.sendMessage() uses the exact same method
signature
         * and request builder pattern as SqsAsyncClient.sendMessage(). This
demonstrates
         * the drop-in replacement capability of the SqsAsyncBatchManager.
         *
         * The SDK automatically:
         * - Buffers individual sendMessage() calls internally
         * - Groups them into batch requests when thresholds are met
         * - Sends SendMessageBatchRequest operations to SQS
         * - Returns individual CompletableFuture responses for each message
         *
         * This transparent batching provides significant performance
improvements
         * without requiring changes to application logic or error handling
patterns.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    // Send multiple messages using the high-level batch manager.
                    // Each batchManager.sendMessage() call uses identical syntax
to
                    // sqsAsyncClient.sendMessage(), demonstrating API
compatibility.
                    // The SDK automatically buffers these calls and sends them
as
                    // batch operations when maxBatchSize (10) or
sendRequestFrequency (200ms)
                    // thresholds are reached, significantly improving
throughput.
                    for (int i = 0; i < batchSize; i++) {
```

```
                        CompletableFuture<SendMessageResponse> future =
batchManager.sendMessage(
                                SendMessageRequest.builder()
                                        .queueUrl(queueUrl)
                                        .messageBody(theMessage)
                                        .build());

                        // Handle the response asynchronously
                        future.whenComplete((response, throwable) -> {
                            if (throwable == null) {
                                producedCount.incrementAndGet();
                            } else if (!(throwable instanceof
java.util.concurrent.CancellationException) &&
                                      !(throwable.getMessage() != null &&
throwable.getMessage().contains("executor not accepting a task"))) {
                                log.error("BatchProducer: Failed to send
message", throwable);
                            }
                            // Ignore CancellationException and executor shutdown
errors - expected during shutdown
                        });
                    }

                    // Small delay to allow batching to occur
                    Thread.sleep(10);
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                log.error("BatchProducer interrupted: " + e.getMessage());
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                log.error("BatchProducer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * Receives and deletes messages individually using SqsAsyncClient for
baseline measurement.
     *
     * This consumer demonstrates traditional single-message operations without
batching.
     * Each receiveMessage() and deleteMessage() call results in separate API
requests,
```

```
     * providing a performance baseline for comparison with batch operations.
     *
     * The method signatures are identical to SqsAsyncBatchManager methods:
     * - receiveMessage() matches batchManager.receiveMessage()
     * - deleteMessage() matches batchManager.deleteMessage()
     *
     * This API consistency allows easy migration to the high-level batching
 library.
     */
    private static class Consumer extends Thread {
        final SqsAsyncClient sqsAsyncClient;
        final String queueUrl;
        final AtomicInteger consumedCount;
        final AtomicBoolean stop;

        /**
         * Creates a consumer thread for single-message operations.
         *
         * @param sqsAsyncClient the SQS client for receiving messages
         * @param queueUrl the URL of the source queue
         * @param consumedCount shared counter for tracking processed messages
         * @param stop shared flag to signal thread termination
         */
        Consumer(SqsAsyncClient sqsAsyncClient, String queueUrl, AtomicInteger
 consumedCount,
                 AtomicBoolean stop) {
            this.sqsAsyncClient = sqsAsyncClient;
            this.queueUrl = queueUrl;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Continuously receives and deletes messages using traditional single-
request operations.
         *
         * Uses SqsAsyncClient methods directly:
         * - receiveMessage(): One API call per receive operation
         * - deleteMessage(): One API call per delete operation
         *
         * This approach demonstrates the baseline performance without batching
 optimization.
         * Compare these method calls with the identical signatures used in
 BatchConsumer
```

```
        * to see how the high-level batching library maintains API
compatibility.
        */
       public void run() {
           try {
               while (!stop.get()) {
                   try {
                       final ReceiveMessageResponse result =
sqsAsyncClient.receiveMessage(
                               ReceiveMessageRequest.builder()
                                       .queueUrl(queueUrl)
                                       .build()).join();

                       if (!result.messages().isEmpty()) {
                           final Message m = result.messages().get(0);
                           // Note: deleteMessage() signature identical to
batchManager.deleteMessage()

sqsAsyncClient.deleteMessage(DeleteMessageRequest.builder()
                                   .queueUrl(queueUrl)
                                   .receiptHandle(m.receiptHandle())
                                   .build()).join();
                           consumedCount.incrementAndGet();
                       }
                   } catch (SdkException |
java.util.concurrent.CompletionException e) {
                       log.error(e.getMessage());
                   }
               }
           } catch (SdkException | java.util.concurrent.CompletionException e) {
               // Handle both SdkException and CompletionException from async
operations.
               // If this unlikely condition occurs, stop.
               log.error("Consumer: " + e.getMessage());
               System.exit(1);
           }
       }
   }

   /**
    * Receives and deletes messages using SqsAsyncBatchManager for automatic
optimization.
    *
```

```
    * This consumer demonstrates the AWS SDK for Java 2.x high-level batching
library
    * for message consumption. The SqsAsyncBatchManager provides two key
optimizations:
    *
    * 1. Receive optimization: Maintains an internal buffer of messages fetched
in the
    *    background, so receiveMessage() calls return immediately from the
buffer
    * 2. Delete batching: Automatically buffers deleteMessage() calls and sends
them
    *    as DeleteMessageBatchRequest operations when thresholds are reached
    *
    * Key features:
    * - Identical API: receiveMessage() and deleteMessage() have the same
signatures
    *   as SqsAsyncClient methods, making this a true drop-in replacement
    * - Background fetching: The batch manager continuously fetches messages to
keep
    *   the internal buffer populated, reducing receive latency
    * - Automatic delete batching: Individual deleteMessage() calls are buffered
and
    *   sent as batch operations (up to 10 per batch, 200ms frequency)
    * - Transparent optimization: No application logic changes needed to benefit
    *
    * Performance benefits:
    * - Reduced API calls through automatic batching of delete operations
    * - Lower latency for receives due to background message buffering
    * - Higher overall throughput with fewer network round trips
    */
   private static class BatchConsumer extends Thread {
       final SqsAsyncBatchManager batchManager;
       final String queueUrl;
       final int batchSize;
       final AtomicInteger consumedCount;
       final AtomicBoolean stop;

       /**
        * Creates a consumer thread for batch operations.
        *
        * @param batchManager the batch manager for efficient message processing
        * @param queueUrl the URL of the source queue
        * @param batchSize the maximum number of messages to receive per batch
        * @param consumedCount shared counter for tracking processed messages
```

```
         * @param stop shared flag to signal thread termination
         */
        BatchConsumer(SqsAsyncBatchManager batchManager, String queueUrl, int
batchSize,
                      AtomicInteger consumedCount, AtomicBoolean stop) {
            this.batchManager = batchManager;
            this.queueUrl = queueUrl;
            this.batchSize = batchSize;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Continuously receives and deletes messages using the high-level
batching library.
         *
         * Demonstrates the key advantage of SqsAsyncBatchManager: identical
method signatures
         * with automatic optimization. Notice how:
         *
         * - batchManager.receiveMessage() uses the same syntax as
sqsAsyncClient.receiveMessage()
         * - batchManager.deleteMessage() uses the same syntax as
sqsAsyncClient.deleteMessage()
         *
         * Behind the scenes, the batch manager:
         * 1. Maintains an internal message buffer populated by background
fetching
         * 2. Returns messages immediately from the buffer (reduced latency)
         * 3. Automatically batches deleteMessage() calls into
DeleteMessageBatchRequest operations
         * 4. Sends batch deletes when maxBatchSize (10) or sendRequestFrequency
(200ms) is reached
         *
         * This provides significant performance improvements with zero code
changes
         * compared to traditional SqsAsyncClient usage patterns.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    // Receive messages using the high-level batch manager.
                    // This call uses identical syntax to
sqsAsyncClient.receiveMessage()
```

```
                    // but benefits from internal message buffering for improved
performance.
                    final ReceiveMessageResponse result =
batchManager.receiveMessage(
                            ReceiveMessageRequest.builder()
                                    .queueUrl(queueUrl)
                                    .maxNumberOfMessages(Math.min(batchSize, 10))
                                    .build()).join();

                    if (!result.messages().isEmpty()) {
                        final List<Message> messages = result.messages();

                        // Delete messages using the batch manager.
                        // Each deleteMessage() call uses identical syntax to
SqsAsyncClient
                        // but the SDK automatically buffers these calls and
sends them
                        // as DeleteMessageBatchRequest operations for optimal
performance.
                        for (Message message : messages) {
                            CompletableFuture<DeleteMessageResponse> future =
batchManager.deleteMessage(
                                    DeleteMessageRequest.builder()
                                            .queueUrl(queueUrl)

.receiptHandle(message.receiptHandle())
                                            .build());

                            future.whenComplete((response, throwable) -> {
                                if (throwable == null) {
                                    consumedCount.incrementAndGet();
                                } else if (!(throwable instanceof
java.util.concurrent.CancellationException) &&
                                            !(throwable.getMessage() != null &&
throwable.getMessage().contains("executor not accepting a task"))) {
                                    log.error("BatchConsumer: Failed to delete
message", throwable);
                                }
                                // Ignore CancellationException and executor
shutdown errors - expected during shutdown
                            });
                        }
                    }
```

```
                    // Small delay to prevent tight polling
                    Thread.sleep(10);
                }
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                log.error("BatchConsumer interrupted: " + e.getMessage());
            } catch (SdkException | java.util.concurrent.CompletionException e) {
                // Handle both SdkException and CompletionException from async
operations.
                // If this unlikely condition occurs, stop.
                log.error("BatchConsumer: " + e.getMessage());
                System.exit(1);
            }
        }
    }

    /**
     * Displays real-time throughput statistics every second.
     *
     * This thread logs the current count of produced and consumed messages
     * to help you monitor the performance comparison.
     */
    private static class Monitor extends Thread {
        private final AtomicInteger producedCount;
        private final AtomicInteger consumedCount;
        private final AtomicBoolean stop;

        /**
         * Creates a monitoring thread that displays throughput statistics.
         *
         * @param producedCount shared counter for messages sent
         * @param consumedCount shared counter for messages processed
         * @param stop shared flag to signal thread termination
         */
        Monitor(AtomicInteger producedCount, AtomicInteger consumedCount,
                AtomicBoolean stop) {
            this.producedCount = producedCount;
            this.consumedCount = consumedCount;
            this.stop = stop;
        }

        /**
         * Logs throughput statistics every second until stopped.
         *
```

```
         * Displays the current count of produced and consumed messages
         * to help monitor the performance comparison between batching
 strategies.
         */
        public void run() {
            try {
                while (!stop.get()) {
                    Thread.sleep(1000);
                    log.info("produced messages = " + producedCount.get()
                            + ", consumed messages = " + consumedCount.get());
                }
            } catch (InterruptedException e) {
                // Allow the thread to exit.
            }
        }
    }
}
```

- 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
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage
  - SendMessageBatch

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

Amazon SQS メッセージ関数をラップする関数を作成します。

```
import logging
import sys

import boto3
from botocore.exceptions import ClientError

import queue_wrapper

logger = logging.getLogger(__name__)
sqs = boto3.resource("sqs")

def send_messages(queue, messages):
    """
    Send a batch of messages in a single request to an SQS queue.
    This request may return overall success even when some messages were not
 sent.
    The caller must inspect the Successful and Failed lists in the response and
    resend any failed messages.

    :param queue: The queue to receive the messages.
    :param messages: The messages to send to the queue. These are simplified to
                     contain only the message body and attributes.
    :return: The response from SQS that contains the list of successful and
 failed
             messages.
    """
    try:
        entries = [
            {
                "Id": str(ind),
                "MessageBody": msg["body"],
                "MessageAttributes": msg["attributes"],
            }
            for ind, msg in enumerate(messages)
        ]
        response = queue.send_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info(
                    "Message sent: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
```

```
                )
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Failed to send: %s: %s",
                    msg_meta["MessageId"],
                    messages[int(msg_meta["Id"])]["body"],
                )
    except ClientError as error:
        logger.exception("Send messages failed to queue: %s", queue)
        raise error
    else:
        return response



def receive_messages(queue, max_number, wait_time):
    """
    Receive a batch of messages in a single request from an SQS queue.

    :param queue: The queue from which to receive messages.
    :param max_number: The maximum number of messages to receive. The actual
 number
                        of messages received might be less.
    :param wait_time: The maximum time to wait (in seconds) before returning.
 When
                       this number is greater than zero, long polling is used.
 This
                       can result in reduced costs and fewer false empty
 responses.
    :return: The list of Message objects received. These each contain the body
             of the message and metadata and custom attributes.
    """
    try:
        messages = queue.receive_messages(
            MessageAttributeNames=["All"],
            MaxNumberOfMessages=max_number,
            WaitTimeSeconds=wait_time,
        )
        for msg in messages:
            logger.info("Received message: %s: %s", msg.message_id, msg.body)
    except ClientError as error:
        logger.exception("Couldn't receive messages from queue: %s", queue)
        raise error
```

```
    else:
        return messages



def delete_messages(queue, messages):
    """
    Delete a batch of messages from a queue in a single request.

    :param queue: The queue from which to delete the messages.
    :param messages: The list of messages to delete.
    :return: The response from SQS that contains the list of successful and
 failed
             message deletions.
    """
    try:
        entries = [
            {"Id": str(ind), "ReceiptHandle": msg.receipt_handle}
            for ind, msg in enumerate(messages)
        ]
        response = queue.delete_messages(Entries=entries)
        if "Successful" in response:
            for msg_meta in response["Successful"]:
                logger.info("Deleted %s",
 messages[int(msg_meta["Id"])].receipt_handle)
        if "Failed" in response:
            for msg_meta in response["Failed"]:
                logger.warning(
                    "Could not delete %s",
 messages[int(msg_meta["Id"])].receipt_handle
                )
    except ClientError:
        logger.exception("Couldn't delete messages from queue %s", queue)
    else:
        return response
```

ラッパー関数を使用してメッセージをバッチで送受信します。

```
def usage_demo():
    """
```

```
    Shows how to:
    * Read the lines from this Python file and send the lines in
      batches of 10 as messages to a queue.
    * Receive the messages in batches until the queue is empty.
    * Reassemble the lines of the file and verify they match the original file.
    """

    def pack_message(msg_path, msg_body, msg_line):
        return {
            "body": msg_body,
            "attributes": {
                "path": {"StringValue": msg_path, "DataType": "String"},
                "line": {"StringValue": str(msg_line), "DataType": "String"},
            },
        }

    def unpack_message(msg):
        return (
            msg.message_attributes["path"]["StringValue"],
            msg.body,
            int(msg.message_attributes["line"]["StringValue"]),
        )

    print("-" * 88)
    print("Welcome to the Amazon Simple Queue Service (Amazon SQS) demo!")
    print("-" * 88)

    queue = queue_wrapper.create_queue("sqs-usage-demo-message-wrapper")

    with open(__file__) as file:
        lines = file.readlines()

    line = 0
    batch_size = 10
    received_lines = [None] * len(lines)
    print(f"Sending file lines in batches of {batch_size} as messages.")
    while line < len(lines):
        messages = [
            pack_message(__file__, lines[index], index)
            for index in range(line, min(line + batch_size, len(lines)))
        ]
        line = line + batch_size
        send_messages(queue, messages)
        print(".", end="")
```

```
        sys.stdout.flush()
    print(f"Done. Sent {len(lines) - 1} messages.")

    print(f"Receiving, handling, and deleting messages in batches of
{batch_size}.")
    more_messages = True
    while more_messages:
        received_messages = receive_messages(queue, batch_size, 2)
        print(".", end="")
        sys.stdout.flush()
        for message in received_messages:
            path, body, line = unpack_message(message)
            received_lines[line] = body
        if received_messages:
            delete_messages(queue, received_messages)
        else:
            more_messages = False
    print("Done.")

    if all([lines[index] == received_lines[index] for index in
range(len(lines))]):
        print(f"Successfully reassembled all file lines!")
    else:
        print(f"Uh oh, some lines were missed!")

    queue.delete()

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

- 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
  - DeleteMessage
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessage

- SendMessageBatch

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 SQS 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。

Message AWS Processing Framework for .NET を使用して Amazon SQS メッセージを公開および受信する

次のコード例は、Message Processing Framework for .NET を使用して Amazon SQS AWS メッセージを発行および受信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

Message AWS Processing Framework for .NET のチュートリアルを提供します。このチュートリアルでは、ユーザーが Amazon SQS メッセージを発行できるウェブアプリケーションと、メッセージを受信するコマンドラインアプリケーションを作成します。

完全なソースコードとセットアップと実行の手順については、 AWS SDK for .NET デベロッパーガイドのチュートリアル全体と GitHub の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 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 SQS 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 SQS Java Messaging Library を使用して Amazon SQS の Java Message Service (JMS) インターフェイスを操作する

次のコード例は、Amazon SQS Java Messaging Library を使用して JMS インターフェイスを操作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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

次の例は、標準の Amazon SQS キューで動作し、以下を含みます。

- テキストメッセージの送信。
- メッセージを同期的に受信する。
- メッセージを非同期的に受信する。
- CLIENT_ACKNOWLEDGE モードを使用したメッセージの受信。
- UNORDERED_ACKNOWLEDGE モードを使用したメッセージの受信。
- Spring を使用して依存関係を挿入します。
- 他の例で使用される一般的なメソッドを提供するユーティリティクラス。

Amazon SQS で JMS を使用する方法の詳細については、Amazon SQS デベロッパーガイド」を参照してください。

テキストメッセージの送信。

```
   /**
    * This method establishes a connection to a standard Amazon SQS queue using
the Amazon SQS
    * Java Messaging Library and sends text messages to it. It uses JMS (Java
Message Service) API
    * with automatic acknowledgment mode to ensure reliable message delivery,
and automatically
    * manages all messaging resources.
    *
    * @throws JMSException If there is a problem connecting to or sending
messages to the queue
    */
   public static void doSendTextMessage() throws JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
```

```
                SqsClient.create()
        );

        // Create the connection in a try-with-resources statement so that it's
closed automatically.
        try (SQSConnection connection = connectionFactory.createConnection()) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session that uses the JMS auto-acknowledge mode.
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
            MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));

            createAndSendMessages(session, producer);
        } // The connection closes automatically. This also closes the session.
        LOGGER.info("Connection closed");
    }

    /**
     * This method reads text input from the keyboard and sends each line as a
separate message
     * to a standard Amazon SQS queue using the Amazon SQS Java Messaging
Library. It continues
     * to accept input until the user enters an empty line, using JMS (Java
Message Service) API to
     * handle the message delivery.
     *
     * @param session The JMS session used to create messages
     * @param producer The JMS message producer used to send messages to the
queue
     */
    private static void createAndSendMessages(Session session, MessageProducer
producer) {
        BufferedReader inputReader = new BufferedReader(
                new InputStreamReader(System.in, Charset.defaultCharset()));

        try {
            String input;
            while (true) {
                LOGGER.info("Enter message to send (leave empty to exit): ");
```

```
                input = inputReader.readLine();
                if (input == null || input.isEmpty()) break;

                TextMessage message = session.createTextMessage(input);
                producer.send(message);
                LOGGER.info("Send message {}", message.getJMSMessageID());
            }
        } catch (EOFException e) {
            // Just return on EOF
        } catch (IOException e) {
            LOGGER.error("Failed reading input: {}", e.getMessage(), e);
        } catch (JMSException e) {
            LOGGER.error("Failed sending message: {}", e.getMessage(), e);
        }
    }
```

メッセージを同期的に受信する。

```
    /**
     * This method receives messages from a standard Amazon SQS queue using the
Amazon SQS Java
     * Messaging Library. It creates a connection to the queue using JMS (Java
Message Service),
     * waits for messages to arrive, and processes them one at a time. The method
handles all
     * necessary setup and cleanup of messaging resources.
     *
     * @throws JMSException If there is a problem connecting to or receiving
messages from the queue
     */
    public static void doReceiveMessageSync() throws JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create a connection.
        try (SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
```

```
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session.
            Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);
            MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

            connection.start();

            receiveMessages(consumer);
        }  // The connection closes automatically. This also closes the session.
        LOGGER.info("Connection closed");
    }

    /**
     * This method continuously checks for new messages from a standard Amazon
SQS queue using
     * the Amazon SQS Java Messaging Library. It waits up to 20 seconds for each
message, processes
     * it using JMS (Java Message Service), and confirms receipt. The method
stops checking for
     * messages after 20 seconds of no activity.
     *
     * @param consumer The JMS message consumer that receives messages from the
queue
     */
    private static void receiveMessages(MessageConsumer consumer) {
        try {
            while (true) {
                LOGGER.info("Waiting for messages...");
                // Wait 1 minute for a message
                Message message =
consumer.receive(Duration.ofSeconds(20).toMillis());
                if (message == null) {
                    LOGGER.info("Shutting down after 20 seconds of silence.");
                    break;
                }
                SqsJmsExampleUtils.handleMessage(message);
                message.acknowledge();
                LOGGER.info("Acknowledged message {}",
message.getJMSMessageID());
            }
```

```
        } catch (JMSException e) {
            LOGGER.error("Error receiving from SQS: {}", e.getMessage(), e);
        }
    }
```

メッセージを非同期的に受信する。

```
    /**
     * This method sets up automatic message handling for a standard Amazon SQS
queue using the
     * Amazon SQS Java Messaging Library. It creates a listener that processes
messages as soon
     * as they arrive using JMS (Java Message Service), runs for 5 seconds, then
cleans up all
     * messaging resources.
     *
     * @throws JMSException If there is a problem connecting to or receiving
messages from the queue
     */
    public static void doReceiveMessageAsync() throws JMSException {
        // Create a connection factory.
        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
                SqsClient.create()
        );

        // Create a connection.
        try (SQSConnection connection = connectionFactory.createConnection() ) {

            // Create the queue if needed.
            SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);

            // Create a session.
            Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

            try {
                // Create a consumer for the queue.
                MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));
```

```
                // Provide an implementation of the MessageListener interface,
which has a single 'onMessage' method.
                // We use a lambda expression for the implementation.
                consumer.setMessageListener(message -> {
                    try {
                        SqsJmsExampleUtils.handleMessage(message);
                        message.acknowledge();
                    } catch (JMSException e) {
                        LOGGER.error("Error processing message: {}",
e.getMessage());
                    }
                });
                // Start receiving incoming messages.
                connection.start();
                LOGGER.info("Waiting for messages...");
            } catch (JMSException e) {
                throw new RuntimeException(e);
            }
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }  // The connection closes automatically. This also closes the session.
        LOGGER.info( "Connection closed" );
    }
```

CLIENT_ACKNOWLEDGE モードを使用したメッセージの受信。

```
    /**
     * This method demonstrates how message acknowledgment affects message
processing in a standard
     * Amazon SQS queue using the Amazon SQS Java Messaging Library. It sends
messages to the queue,
     * then shows how JMS (Java Message Service) client acknowledgment mode
handles both explicit
     * and implicit message confirmations, including how acknowledging one
message can automatically
     * acknowledge previous messages.
     *
     * @throws JMSException If there is a problem with the messaging operations
     */
```

```
   public static void doReceiveMessagesSyncClientAcknowledge() throws
JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               SqsClient.create()
       );

       // Create the connection in a try-with-resources statement so that it's
closed automatically.
       try (SQSConnection connection = connectionFactory.createConnection() ) {

           // Create the queue if needed.
           SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
TIME_OUT_SECONDS);

           // Create a session with client acknowledge mode.
           Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

           // Create a producer and consumer.
           MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));
           MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

           // Open the connection.
           connection.start();

           // Send two text messages.
           sendMessage(producer, session, "Message 1");
           sendMessage(producer, session, "Message 2");

           // Receive a message and don't acknowledge it.
           receiveMessage(consumer, false);

           // Receive another message and acknowledge it.
           receiveMessage(consumer, true);

           // Wait for the visibility time out, so that unacknowledged messages
reappear in the queue,
           LOGGER.info("Waiting for visibility timeout...");
           try {
               Thread.sleep(TIME_OUT_MILLIS);
```

```
            } catch (InterruptedException e) {
                LOGGER.error("Interrupted while waiting for visibility timeout",
e);
                Thread.currentThread().interrupt();
                throw new RuntimeException("Processing interrupted", e);
            }

            /*  We will attempt to receive another message, but none will be
available. This is because in
                CLIENT_ACKNOWLEDGE mode, when we acknowledged the second message,
all previous messages were
                automatically acknowledged as well. Therefore, although we never
directly acknowledged the first
                message, it was implicitly acknowledged when we confirmed the
second one. */
            receiveMessage(consumer, true);
        } // The connection closes automatically. This also closes the session.
        LOGGER.info("Connection closed.");

    }


    /**
     * Sends a text message using the specified JMS MessageProducer and Session.
     *
     * @param producer    The JMS MessageProducer used to send the message
     * @param session     The JMS Session used to create the text message
     * @param messageText The text content to be sent in the message
     * @throws JMSException If there is an error creating or sending the message
     */
    private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSException {
        // Create a text message and send it.
        producer.send(session.createTextMessage(messageText));
    }

    /**
     * Receives and processes a message from a JMS queue using the specified
consumer.
     * The method waits for a message until the configured timeout period is
reached.
     * If a message is received, it is logged and optionally acknowledged based
on the
     * acknowledge parameter.
```

```
    *
    * @param consumer    The JMS MessageConsumer used to receive messages from
the queue
    * @param acknowledge Boolean flag indicating whether to acknowledge the
message.
    *                    If true, the message will be acknowledged after
processing
    * @throws JMSException If there is an error receiving, processing, or
acknowledging the message
    */
   private static void receiveMessage(MessageConsumer consumer, boolean
acknowledge) throws JMSException {
       // Receive a message.
       Message message = consumer.receive(TIME_OUT_MILLIS);

       if (message == null) {
           LOGGER.info("Queue is empty!");
       } else {
           // Since this queue has only text messages, cast the message object
and print the text.
           LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
message).getText(), acknowledge);

           // Acknowledge the message if asked.
           if (acknowledge) message.acknowledge();
       }
   }
```

UNORDERED_ACKNOWLEDGE モードを使用したメッセージの受信。

```
   /**
    * Demonstrates message acknowledgment behavior in UNORDERED_ACKNOWLEDGE mode
with Amazon SQS JMS.
    * In this mode, each message must be explicitly acknowledged regardless of
receive order.
    * Unacknowledged messages return to the queue after the visibility timeout
expires,
    * unlike CLIENT_ACKNOWLEDGE mode where acknowledging one message
acknowledges all previous messages.
    *
    * @throws JMSException         If a JMS-related error occurs during message
operations
```

```
    */
   public static void doReceiveMessagesUnorderedAcknowledge() throws
JMSException {
       // Create a connection factory.
       SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
               new ProviderConfiguration(),
               SqsClient.create()
       );

       // Create the connection in a try-with-resources statement so that it's
closed automatically.
       try( SQSConnection connection = connectionFactory.createConnection() ) {

           // Create the queue if needed.
           SqsJmsExampleUtils.ensureQueueExists(connection, QUEUE_NAME,
TIME_OUT_SECONDS);

           // Create a session with unordered acknowledge mode.
           Session session = connection.createSession(false,
SQSSession.UNORDERED_ACKNOWLEDGE);

           // Create the producer and consumer.
           MessageProducer producer =
session.createProducer(session.createQueue(QUEUE_NAME));
           MessageConsumer consumer =
session.createConsumer(session.createQueue(QUEUE_NAME));

           // Open a connection.
           connection.start();

           // Send two text messages.
           sendMessage(producer, session, "Message 1");
           sendMessage(producer, session, "Message 2");

           // Receive a message and don't acknowledge it.
           receiveMessage(consumer, false);

           // Receive another message and acknowledge it.
           receiveMessage(consumer, true);

           // Wait for the visibility time out, so that unacknowledged messages
reappear in the queue.
           LOGGER.info("Waiting for visibility timeout...");
           try {
```

```
                Thread.sleep(TIME_OUT_MILLIS);
            } catch (InterruptedException e) {
                LOGGER.error("Interrupted while waiting for visibility timeout",
e);
                Thread.currentThread().interrupt();
                throw new RuntimeException("Processing interrupted", e);
            }

            /*  We will attempt to receive another message, and we'll get the
first message again. This occurs
                because in UNORDERED_ACKNOWLEDGE mode, each message requires its
own separate acknowledgment.
                Since we only acknowledged the second message, the first message
remains in the queue for
                redelivery. */
            receiveMessage(consumer, true);

            LOGGER.info("Connection closed.");
        } // The connection closes automatically. This also closes the session.
    }

    /**
     * Sends a text message to an Amazon SQS queue using JMS.
     *
     * @param producer    The JMS MessageProducer for the queue
     * @param session     The JMS Session for message creation
     * @param messageText The message content
     * @throws JMSException If message creation or sending fails
     */
    private static void sendMessage(MessageProducer producer, Session session,
String messageText) throws JMSException {
        // Create a text message and send it.
        producer.send(session.createTextMessage(messageText));
    }
    /**
     * Synchronously receives a message from an Amazon SQS queue using the JMS
API
     * with an acknowledgment parameter.
     *
     * @param consumer    The JMS MessageConsumer for the queue
     * @param acknowledge If true, acknowledges the message after receipt
     * @throws JMSException If message reception or acknowledgment fails
     */
```

```
    private static void receiveMessage(MessageConsumer consumer, boolean
 acknowledge) throws JMSException {
        // Receive a message.
        Message message = consumer.receive(TIME_OUT_MILLIS);

        if (message == null) {
            LOGGER.info("Queue is empty!");
        } else {
            // Since this queue has only text messages, cast the message object
 and print the text.
            LOGGER.info("Received: {}    Acknowledged: {}", ((TextMessage)
 message).getText(), acknowledge);

            // Acknowledge the message if asked.
            if (acknowledge) message.acknowledge();
        }
    }
```

Spring を使用して依存関係を挿入します。

```
package com.example.sqs.jms.spring;

import com.amazon.sqs.javamessaging.SQSConnection;
import com.example.sqs.jms.SqsJmsExampleUtils;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.NoSuchBeanDefinitionException;
import org.springframework.context.support.FileSystemXmlApplicationContext;

import java.io.File;
import java.net.URL;
import java.util.concurrent.TimeUnit;

/**
 * Demonstrates how to send and receive messages using the Amazon SQS Java
 Messaging Library
 * with Spring Framework integration. This example connects to a standard Amazon
 SQS message
 * queue using Spring's dependency injection to configure the connection and
 messaging components.
```

```
 * The application uses the JMS (Java Message Service) API to handle message
 operations.
 */
public class SpringExample {
    private static final Integer POLLING_SECONDS = 15;
    private static final String SPRING_XML_CONFIG_FILE =
 "SpringExampleConfiguration.xml.txt";
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SpringExample.class);

    /**
     * Demonstrates sending and receiving messages through a standard Amazon SQS
 message queue
     * using Spring Framework configuration. This method loads connection
 settings from an XML file,
     * establishes a messaging session using the Amazon SQS Java Messaging
 Library, and processes
     * messages using JMS (Java Message Service) operations. If the queue doesn't
 exist, it will
     * be created automatically.
     *
     * @param args Command line arguments (not used)
     */
    public static void main(String[] args) {

        URL resource =
 SpringExample.class.getClassLoader().getResource(SPRING_XML_CONFIG_FILE);
        File springFile = new File(resource.getFile());
        if (!springFile.exists() || !springFile.canRead()) {
            LOGGER.error("File " + SPRING_XML_CONFIG_FILE + " doesn't exist or
 isn't readable.");
            System.exit(1);
        }

        try (FileSystemXmlApplicationContext context =
                     new FileSystemXmlApplicationContext("file://" +
 springFile.getAbsolutePath())) {

            Connection connection;
            try {
                connection = context.getBean(Connection.class);
            } catch (NoSuchBeanDefinitionException e) {
                LOGGER.error("Can't find the JMS connection to use: " +
 e.getMessage(), e);
```

```
                System.exit(2);
                return;
            }

            String queueName;
            try {
                queueName = context.getBean("queueName", String.class);
            } catch (NoSuchBeanDefinitionException e) {
                LOGGER.error("Can't find the name of the queue to use: " +
e.getMessage(), e);
                System.exit(3);
                return;
            }
            try {
                if (connection instanceof SQSConnection) {
                    SqsJmsExampleUtils.ensureQueueExists((SQSConnection)
connection, queueName, SqsJmsExampleUtils.QUEUE_VISIBILITY_TIMEOUT);
                }
                // Create the JMS session.
                Session session = connection.createSession(false,
Session.CLIENT_ACKNOWLEDGE);

                SqsJmsExampleUtils.sendTextMessage(session, queueName);
                MessageConsumer consumer =
session.createConsumer(session.createQueue(queueName));

                receiveMessages(consumer);
            } catch (JMSException e) {
                LOGGER.error(e.getMessage(), e);
                throw new RuntimeException(e);
            }
        }   // Spring context autocloses. Managed Spring beans that implement
AutoClosable, such as the
        // 'connection' bean, are also closed.
        LOGGER.info("Context closed");
    }

    /**
     * Continuously checks for and processes messages from a standard Amazon SQS
message queue
     * using the Amazon SQS Java Messaging Library underlying the JMS API. This
method waits for incoming messages,
     * processes them when they arrive, and acknowledges their receipt using JMS
(Java Message
```

```
     * Service) operations. The method will stop checking for messages after 15
 seconds of
     * inactivity.
     *
     * @param consumer The JMS message consumer used to receive messages from the
 queue
     */
    private static void receiveMessages(MessageConsumer consumer) {
        try {
            while (true) {
                LOGGER.info("Waiting for messages...");
                // Wait 15 seconds for a message.
                Message message =
 consumer.receive(TimeUnit.SECONDS.toMillis(POLLING_SECONDS));
                if (message == null) {
                    LOGGER.info("Shutting down after {} seconds of silence.",
 POLLING_SECONDS);
                    break;
                }
                SqsJmsExampleUtils.handleMessage(message);
                message.acknowledge();
                LOGGER.info("Message acknowledged.");
            }
        } catch (JMSException e) {
            LOGGER.error("Error receiving from SQS.", e);
        }
    }
}
```

Spring Bean 定義。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
        xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-3.0.xsd
        ">
    <!-- Define the AWS Region -->
    <bean id="region" class="software.amazon.awssdk.regions.Region" factory-
method="of">
```

```
        <constructor-arg value="us-east-1"/>
    </bean>

    <bean id="credentialsProviderBean"
 class="software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider"
          factory-method="create"/>

    <bean id="clientBuilder"
 class="software.amazon.awssdk.services.sqs.SqsClient" factory-method="builder"/>

    <bean id="regionSetClientBuilder" factory-bean="clientBuilder" factory-
method="region">
        <constructor-arg ref="region"/>
    </bean>

    <!-- Configure the Builder with Credentials Provider -->
    <bean id="sqsClient" factory-bean="regionSetClientBuilder" factory-
method="credentialsProvider">
        <constructor-arg ref="credentialsProviderBean"/>
    </bean>

    <bean id="providerConfiguration"
 class="com.amazon.sqs.javamessaging.ProviderConfiguration">
        <property name="numberOfMessagesToPrefetch" value="5"/>
    </bean>

    <bean id="connectionFactory"
 class="com.amazon.sqs.javamessaging.SQSConnectionFactory">
        <constructor-arg ref="providerConfiguration"/>
        <constructor-arg ref="clientBuilder"/>
    </bean>

    <bean id="connection"
          factory-bean="connectionFactory"
          factory-method="createConnection"
          init-method="start"
          destroy-method="close"/>

    <bean id="queueName" class="java.lang.String">
        <constructor-arg value="SQSJMSClientExampleQueue"/>
    </bean>
</beans>
```

他の例で使用される一般的なメソッドを提供するユーティリティクラス。

```
package com.example.sqs.jms;

import com.amazon.sqs.javamessaging.AmazonSQSMessagingClientWrapper;
import com.amazon.sqs.javamessaging.ProviderConfiguration;
import com.amazon.sqs.javamessaging.SQSConnection;
import com.amazon.sqs.javamessaging.SQSConnectionFactory;
import jakarta.jms.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.core.exception.SdkException;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.CreateQueueRequest;
import software.amazon.awssdk.services.sqs.model.QueueAttributeName;

import java.time.Duration;
import java.util.Base64;
import java.util.Map;

/**
 * This utility class provides helper methods for working with Amazon Simple
 Queue Service (Amazon SQS)
 * through the Java Message Service (JMS) interface. It contains common
 operations for managing message
 * queues and handling message delivery.
 */
public class SqsJmsExampleUtils {
    private static final Logger LOGGER =
 LoggerFactory.getLogger(SqsJmsExampleUtils.class);
    public static final Long QUEUE_VISIBILITY_TIMEOUT = 5L;

    /**
     * This method verifies that a message queue exists and creates it if
 necessary. The method checks for
     * an existing queue first to optimize performance.
     *
     * @param connection The active connection to the messaging service
     * @param queueName The name of the queue to verify or create
     * @param visibilityTimeout The duration in seconds that messages will be
 hidden after being received
     * @throws JMSException If there is an error accessing or creating the queue
     */
```

```
   public static void ensureQueueExists(SQSConnection connection, String
queueName, Long visibilityTimeout) throws JMSException {
       AmazonSQSMessagingClientWrapper client =
connection.getWrappedAmazonSQSClient();

      /* In most cases, you can do this with just a 'createQueue' call, but
'getQueueUrl'
      (called by 'queueExists') is a faster operation for the common case where
the queue
      already exists. Also, many users and roles have permission to call
'getQueueUrl'
      but don't have permission to call 'createQueue'.
      */
       if( !client.queueExists(queueName) ) {
           CreateQueueRequest createQueueRequest = CreateQueueRequest.builder()
                   .queueName(queueName)
                   .attributes(Map.of(QueueAttributeName.VISIBILITY_TIMEOUT,
String.valueOf(visibilityTimeout)))
                   .build();
           client.createQueue( createQueueRequest );
       }
   }

   /**
    * This method sends a simple text message to a specified message queue. It
handles all necessary
    * setup for the message delivery process.
    *
    * @param session The active messaging session used to create and send the
message
    * @param queueName The name of the queue where the message will be sent
    */
   public static void sendTextMessage(Session session, String queueName) {
       // Rest of implementation...

       try {
           MessageProducer producer =
session.createProducer( session.createQueue( queueName) );
           Message message = session.createTextMessage("Hello world!");
           producer.send(message);
       } catch (JMSException e) {
           LOGGER.error( "Error receiving from SQS", e );
       }
   }
```

```
   /**
    * This method processes incoming messages and logs their content based on
the message type.
    * It supports text messages, binary data, and Java objects.
    *
    * @param message The message to be processed and logged
    * @throws JMSException If there is an error reading the message content
    */
   public static void handleMessage(Message message) throws JMSException {
       // Rest of implementation...
       LOGGER.info( "Got message {}", message.getJMSMessageID() );
       LOGGER.info( "Content: ");
       if(message instanceof TextMessage txtMessage) {
           LOGGER.info( "\t{}", txtMessage.getText() );
       } else if(message instanceof BytesMessage byteMessage){
           // Assume the length fits in an int - SQS only supports sizes up to
256k so that
           // should be true
           byte[] bytes = new byte[(int)byteMessage.getBodyLength()];
           byteMessage.readBytes(bytes);
           LOGGER.info( "\t{}", Base64.getEncoder().encodeToString( bytes ) );
       } else if( message instanceof ObjectMessage) {
           ObjectMessage objMessage = (ObjectMessage) message;
           LOGGER.info( "\t{}", objMessage.getObject() );
       }
   }

   /**
    * This method sets up automatic message processing for a specified queue. It
creates a listener
    * that will receive and handle incoming messages without blocking the main
program.
    *
    * @param session The active messaging session
    * @param queueName The name of the queue to monitor
    * @param connection The active connection to the messaging service
    */
   public static void receiveMessagesAsync(Session session, String queueName,
Connection connection) {
       // Rest of implementation...
       try {
           // Create a consumer for the queue.
```

```
            MessageConsumer consumer =
 session.createConsumer(session.createQueue(queueName));
            // Provide an implementation of the MessageListener interface, which
 has a single 'onMessage' method.
            // We use a lambda expression for the implementation.
            consumer.setMessageListener(message -> {
                try {
                    SqsJmsExampleUtils.handleMessage(message);
                    message.acknowledge();
                } catch (JMSException e) {
                    LOGGER.error("Error processing message: {}", e.getMessage());
                }
            });
            // Start receiving incoming messages.
            connection.start();
        } catch (JMSException e) {
            throw new RuntimeException(e);
        }
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }


    /**
     * This method performs cleanup operations after message processing is
 complete. It receives
     * any messages in the specified queue, removes the message queue and closes
 all
     * active connections to prevent resource leaks.
     *
     * @param queueName The name of the queue to be removed
     * @param visibilityTimeout The duration in seconds that messages are hidden
 after being received
     * @throws JMSException If there is an error during the cleanup process
     */
    public static void cleanUpExample(String queueName, Long visibilityTimeout)
 throws JMSException {
        LOGGER.info("Performing cleanup.");

        SQSConnectionFactory connectionFactory = new SQSConnectionFactory(
                new ProviderConfiguration(),
```

```
                SqsClient.create()
        );

        try (SQSConnection connection = connectionFactory.createConnection() ) {
            ensureQueueExists(connection, queueName, visibilityTimeout);
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            receiveMessagesAsync(session, queueName, connection);

            SqsClient sqsClient =
connection.getWrappedAmazonSQSClient().getAmazonSQSClient();
            try {
                String queueUrl = sqsClient.getQueueUrl(b ->
b.queueName(queueName)).queueUrl();
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
                LOGGER.info("Queue deleted: {}", queueUrl);
            } catch (SdkException e) {
                LOGGER.error("Error during SQS operations: ", e);
            }
        }
        LOGGER.info("Clean up: Connection closed");
    }

    /**
     * This method creates a background task that sends multiple messages to a
specified queue
     * after waiting for a set time period. The task operates independently to
ensure efficient
     * message processing without interrupting other operations.
     *
     * @param queueName The name of the queue where messages will be sent
     * @param secondsToWait The number of seconds to wait before sending messages
     * @param numMessages The number of messages to send
     * @param visibilityTimeout The duration in seconds that messages remain
hidden after being received
     * @return A task that can be executed to send the messages
     */
    public static Runnable sendAMessageAsync(String queueName, Long
secondsToWait, Integer numMessages, Long visibilityTimeout) {
        return () -> {
            try {
                Thread.sleep(Duration.ofSeconds(secondsToWait).toMillis());
            } catch (InterruptedException e) {
```

```
                Thread.currentThread().interrupt();
                throw new RuntimeException(e);
            }
            try {
                SQSConnectionFactory connectionFactory = new
 SQSConnectionFactory(
                        new ProviderConfiguration(),
                        SqsClient.create()
                );
                try (SQSConnection connection =
 connectionFactory.createConnection()) {
                    ensureQueueExists(connection, queueName, visibilityTimeout);
                    Session session = connection.createSession(false,
 Session.CLIENT_ACKNOWLEDGE);
                    for (int i = 1; i <= numMessages; i++) {
                        MessageProducer producer =
 session.createProducer(session.createQueue(queueName));
                        producer.send(session.createTextMessage("Hello World " +
 i + "!"));
                    }
                }
            } catch (JMSException e) {
                LOGGER.error(e.getMessage(), e);
                throw new RuntimeException(e);
            }
        };
    }
}
```

- 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
  - DeleteQueue

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 SQS 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 を使用してキュータグと Amazon SQS を操作する

次のコード例は、Amazon SQS でタグ付けオペレーションを実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。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

次の の例では、キューのタグを作成し、タグを一覧表示して、タグを削除します。

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.sqs.SqsClient;
import software.amazon.awssdk.services.sqs.model.ListQueueTagsResponse;
import software.amazon.awssdk.services.sqs.model.QueueDoesNotExistException;
import software.amazon.awssdk.services.sqs.model.SqsException;

import java.util.Map;
import java.util.UUID;

/**
 * Before running this Java V2 code example, set up your development environment,
 including your credentials. For more
 * information, see the <a href="https://docs.aws.amazon.com/sdk-for-java/latest/
developer-guide/get-started.html">AWS
 * SDK for Java Developer Guide</a>.
 */
public class TagExamples {
    static final SqsClient sqsClient = SqsClient.create();
    static final String queueName = "TagExamples-queue-" +
 UUID.randomUUID().toString().replace("-", "").substring(0, 20);
    private static final Logger LOGGER =
 LoggerFactory.getLogger(TagExamples.class);
```

```
    public static void main(String[] args) {
        final String queueUrl;
        try {
            queueUrl = sqsClient.createQueue(b ->
b.queueName(queueName)).queueUrl();
            LOGGER.info("Queue created. The URL is: {}", queueUrl);
        } catch (RuntimeException e) {
            LOGGER.error("Program ending because queue was not created.");
            throw new RuntimeException(e);
        }
        try {
            addTags(queueUrl);
            listTags(queueUrl);
            removeTags(queueUrl);
        } catch (RuntimeException e) {
            LOGGER.error("Program ending because of an error in a method.");
        } finally {
            try {
                sqsClient.deleteQueue(b -> b.queueUrl(queueUrl));
                LOGGER.info("Queue successfully deleted. Program ending.");
                sqsClient.close();
            } catch (RuntimeException e) {
                LOGGER.error("Program ending.");
            } finally {
                sqsClient.close();
            }
        }
    }

    /** This method demonstrates how to use a Java Map to a tag a aueue.
     * @param queueUrl The URL of the queue to tag.
     */
    public static void addTags(String queueUrl) {
        // Build a map of the tags.
        final Map<String, String> tagsToAdd = Map.of(
                "Team", "Development",
                "Priority", "Beta",
                "Accounting ID", "456def");

        try {
            // Add tags to the queue using a Consumer<TagQueueRequest.Builder>
parameter.
            sqsClient.tagQueue(b -> b
                    .queueUrl(queueUrl)
```

```
                    .tags(tagsToAdd)
            );
        } catch (QueueDoesNotExistException e) {
            LOGGER.error("Queue does not exist: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }
    }

    /** This method demonstrates how to view the tags for a queue.
     * @param queueUrl The URL of the queue whose tags you want to list.
     */
    public static void listTags(String queueUrl) {
        ListQueueTagsResponse response;
        try {
            // Call the listQueueTags method with a
Consumer<ListQueueTagsRequest.Builder> parameter that creates a
ListQueueTagsRequest.
            response = sqsClient.listQueueTags(b -> b
                    .queueUrl(queueUrl));
        } catch (SqsException e) {
            LOGGER.error("Exception thrown: {}", e.getMessage(), e);
            throw new RuntimeException(e);
        }

        // Log the tags.
        response.tags()
                .forEach((k, v) ->
                        LOGGER.info("Key: {} -> Value: {}", k, v));
    }

    /**
     * This method demonstrates how to remove tags from a queue.
     * @param queueUrl The URL of the queue whose tags you want to remove.
     */
    public static void removeTags(String queueUrl) {
        try {
            // Call the untagQueue method with a
Consumer<UntagQueueRequest.Builder> parameter.
            sqsClient.untagQueue(b -> b
                    .queueUrl(queueUrl)
                    .tagKeys("Accounting ID") // Remove a single tag.
            );
        } catch (SqsException e) {
            LOGGER.error("Exception thrown: {}", e.getMessage(), e);
```

```
            throw new RuntimeException(e);
        }
    }
}
```

- 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。
  - ListQueueTags
  - TagQueue
  - UntagQueue

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 SQS 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 SQS のサーバーレスの例

次のコード例は、 AWS SDKs で Amazon SQS を使用する方法を示しています。

例

- Amazon SQS トリガーから Lambda 関数を呼び出す
- Amazon SQS トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

Amazon SQS トリガーから Lambda 関数を呼び出す

次のコード例では、SQS キュー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 関数の実装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、設定および実行の方法を確認してください。

.NET を使用した Lambda での SQS 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.SQSEvents;


// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace SqsIntegrationSampleCode
{
    public async Task FunctionHandler(SQSEvent evnt, ILambdaContext context)
    {
        foreach (var message in evnt.Records)
        {
            await ProcessMessageAsync(message, context);
        }

        context.Logger.LogInformation("done");
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        try
        {
            context.Logger.LogInformation($"Processed message {message.Body}");

            // 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検索し、設定および実行の方法を確認してください。

Go を使用した Lambda での SQS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
```

```
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

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検索し、設定および実行の方法を確認してください。

Java を使用した Lambda での SQS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSEvent.SQSMessage;

public class Function implements RequestHandler<SQSEvent, Void> {
    @Override
    public Void handleRequest(SQSEvent sqsEvent, Context context) {
        for (SQSMessage msg : sqsEvent.getRecords()) {
            processMessage(msg, context);
        }
        context.getLogger().log("done");
```

```
        return null;
    }

    private void processMessage(SQSMessage msg, Context context) {
        try {
            context.getLogger().log("Processed message " + msg.getBody());

            // TODO: Do interesting work based on the new message

        } catch (Exception e) {
            context.getLogger().log("An error occurred");
            throw 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、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での SQS 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 message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new 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 SQS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new 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検索し、設定および実行の方法を確認してください。

PHP を使用した Lambda での SQS イベントの消費。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\InvalidLambdaEvent;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

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

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        foreach ($event->getRecords() as $record) {
            $body = $record->getBody();
            // TODO: Do interesting work based on the new message
        }
```

```
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

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検索し、設定および実行の方法を確認してください。

Python を使用した Lambda での SQS 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):
    for message in event['Records']:
        process_message(message)
    print("done")

def process_message(message):
    try:
        print(f"Processed message {message['body']}")
        # TODO: Do interesting work based on the new message
    except Exception as err:
        print("An error occurred")
        raise err
```

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検索し、設定および実行の方法を確認してください。

Ruby を使用した Lambda での SQS 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'].each do |message|
    process_message(message)
  end
  puts "done"
end

def process_message(message)
  begin
    puts "Processed message #{message['body']}"
    # TODO: Do interesting work based on the new message
  rescue StandardError => err
    puts "An error occurred"
    raise err
  end
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検索し、設定および実行の方法を確認してください。

Rust を使用して Lambda で SQS 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::sqs::SqsEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<SqsEvent>) -> Result<(), Error> {
    event.payload.records.iter().for_each(|record| {
        // process the record
        tracing::info!("Message body: {}",
 record.body.as_deref().unwrap_or_default())
    });

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .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 SQS 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 SQS トリガーを使用した Lambda 関数でのバッチアイテムの失敗のレポート

以下のコード例では、SQS キューからイベントを受け取る Lambda 関数のための、部分的なバッチレスポンスの実装方法を示しています。この関数は、レスポンスとしてバッチアイテムの失敗を報告し、対象のメッセージを後で再試行するよう Lambda に伝えます。

.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、設定および実行の方法を確認してください。

.NET を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using Amazon.Lambda.Core;
using Amazon.Lambda.SQSEvents;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri
namespace sqsSample;

public class Function
{
    public async Task<SQSBatchResponse> FunctionHandler(SQSEvent evnt,
 ILambdaContext context)
    {
        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 List<SQSBatchResponse.BatchItemFailure>();
        foreach(var message in evnt.Records)
        {
            try
            {
                //process your message
                await ProcessMessageAsync(message, context);
            }
            catch (System.Exception)
            {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.Add(new
 SQSBatchResponse.BatchItemFailure{ItemIdentifier=message.MessageId});
            }
```

```
        }
        return new SQSBatchResponse(batchItemFailures);
    }

    private async Task ProcessMessageAsync(SQSEvent.SQSMessage message,
 ILambdaContext context)
    {
        if (String.IsNullOrEmpty(message.Body))
        {
            throw new Exception("No Body in SQS Message.");
        }
        context.Logger.LogInformation($"Processed message {message.Body}");
        // TODO: Do interesting work based on the new message
        await Task.CompletedTask;
    }
}
```

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検索し、設定および実行の方法を確認してください。

Go を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "encoding/json"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)
```

```
func handler(ctx context.Context, sqsEvent events.SQSEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {

  if /* Your message processing condition here */ {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

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検索し、設定および実行の方法を確認してください。

Java を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;
import com.amazonaws.services.lambda.runtime.events.SQSBatchResponse;
```

```
import java.util.ArrayList;
import java.util.List;

public class ProcessSQSMessageBatch implements RequestHandler<SQSEvent,
 SQSBatchResponse> {
    @Override
    public SQSBatchResponse handleRequest(SQSEvent sqsEvent, Context context) {

        List<SQSBatchResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<SQSBatchResponse.BatchItemFailure>();
        String messageId = "";
        for (SQSEvent.SQSMessage message : sqsEvent.getRecords()) {
            try {
                //process your message
            } catch (Exception e) {
                //Add failed message identifier to the batchItemFailures list
                batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(message.getMessageId()));
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

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、設定および実行の方法を確認してください。

JavaScript を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
    const batchItemFailures = [];
    for (const record of event.Records) {
        try {
            await processMessageAsync(record, context);
```

```
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }
    return { batchItemFailures };
};

async function processMessageAsync(record, context) {
    if (record.body && record.body.includes("error")) {
        throw new Error("There is an error in the SQS Message.");
    }
    console.log(`Processed message: ${record.body}`);
}
```

TypeScript を使用して Lambda で SQS バッチ項目の失敗を報告します。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord }
 from 'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
 Promise<SQSBatchResponse> => {
    const batchItemFailures: SQSBatchItemFailure[] = [];

    for (const record of event.Records) {
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
    }
    console.log(`Processed message ${record.body}`);
}
```

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検索し、設定および実行の方法を確認してください。

PHP を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

use Bref\Context\Context;
use Bref\Event\Sqs\SqsEvent;
use Bref\Event\Sqs\SqsHandler;
use Bref\Logger\StderrLogger;

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

class Handler extends SqsHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handleSqs(SqsEvent $event, Context $context): void
    {
        $this->logger->info("Processing SQS records");
        $records = $event->getRecords();

        foreach ($records as $record) {
            try {
                // Assuming the SQS message is in JSON format
```

```
                $message = json_decode($record->getBody(), true);
                $this->logger->info(json_encode($message));
                // TODO: Implement your custom processing logic here
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $this->markAsFailed($record);
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords SQS records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

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検索し、設定および実行の方法を確認してください。

Python を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

def lambda_handler(event, context):
    if event:
        batch_item_failures = []
        sqs_batch_response = {}

        for record in event["Records"]:
            try:
                # process message
            except Exception as e:
```

```
                batch_item_failures.append({"itemIdentifier":
 record['messageId']})

        sqs_batch_response["batchItemFailures"] = batch_item_failures
        return sqs_batch_response
```

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検索し、設定および実行の方法を確認してください。

Ruby を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'json'

def lambda_handler(event:, context:)
  if event
    batch_item_failures = []
    sqs_batch_response = {}

    event["Records"].each do |record|
      begin
        # process message
      rescue StandardError => e
        batch_item_failures << {"itemIdentifier" => record['messageId']}
      end
    end

    sqs_batch_response["batchItemFailures"] = batch_item_failures
    return sqs_batch_response
  end
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検索し、設定および実行の方法を確認してください。

Rust を使用した Lambda での SQS バッチアイテム失敗のレポート。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::sqs::{SqsBatchResponse, SqsEvent},
    sqs::{BatchItemFailure, SqsMessage},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn process_record(_: &SqsMessage) -> Result<(), Error> {
    Err(Error::from("Error processing message"))
}

async fn function_handler(event: LambdaEvent<SqsEvent>) ->
 Result<SqsBatchResponse, Error> {
    let mut batch_item_failures = Vec::new();
    for record in event.payload.records {
        match process_record(&record).await {
            Ok(_) => (),
            Err(_) => batch_item_failures.push(BatchItemFailure {
                item_identifier: record.message_id.unwrap(),
            }),
        }
    }

    Ok(SqsBatchResponse {
        batch_item_failures,
    })
}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
    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 SQS 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 SQS の問題のトラブルシューティング

このトピックでは、Amazon SQS コンソール、Amazon SQS API、または Amazon SQS で他のツールを使用する際に発生する可能性がある一般的なエラーや問題のトラブルシューティングに関するアドバイスを提供します Amazon SQS 。ここに記載されていない問題が見つかった場合は、このページの [Feedback] ボタンを使用して報告することができます。

トラブルシューティングに関するアドバイス、およびサポートへの一般的な質問に対する回答については、AWS ナレッジセンターにアクセスしてください。

トピック

- Amazon S3 でのアクセス拒否エラーのトラブルシューティング
- Amazon SQS API エラーのトラブルシューティング
- Amazon SQS デッドレターキューと DLQ 再処理の問題のトラブルシューティング
- Amazon SQS での FIFO スロットリング問題のトラブルシューティング
- Amazon SQS の ReceiveMessage API コールで返されないメッセージのトラブルシューティング
- Amazon SQS のネットワークエラーのトラブルシューティング
- Amazon Simple Queue Serviceのキュー AWS X-Rayを使用したトラブルシューティング

Amazon S3 でのアクセス拒否エラーのトラブルシューティング

以下のトピックでは、Amazon SQS API コールの `AccessDenied` エラーや `AccessDeniedException` エラーの最も一般的な原因について説明します。これらのエラーのトラブルシューティング方法の詳細については、「AWS 情報センターガイド」の「Amazon SQS API コールでの AccessDenied エラーや AccessDeniedException エラーのトラブルシューティングを行うにはどうすればよいですか?」を参照してください。

エラーメッセージの例:

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to
        the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

- または -

```
An error occurred (KMS.AccessDeniedException) when calling the SendMessage
```

```
        operation: User: arn:aws:iam::xxxxx:user/xxxx is not authorized to perform:
        kms:GenerateDataKey on resource: arn:aws:kms:us-east-1:xxxx:key/xxxx with an
explicit
        deny.
```

Amazon SQS キューポリシーと IAM ポリシー

リクエスタが Amazon S3 オペレーションを実行するための適切なアクセス許可を持っているかどうかを確認するには、次の手順を実行します。

- Amazon SQS API コールを実行している IAM プリンシパルを特定します。IAM プリンシパルが同じアカウントに属している場合、Amazon SQS キューポリシーまたは AWS Identity and Access Management (IAM) ポリシーのいずれかに、アクションへのアクセスを明示的に許可するアクセス許可が含まれている必要があります。
- プリンシパルが IAM エンティティの場合:
  - IAM ユーザーまたはロールを特定するには、 AWS Management Consoleの右上隅を確認するか、`aws sts get-caller-identity` コマンドを使用できます。
  - IAM ユーザーまたはロールに関連する IAM ポリシーを確認します。次のいずれかの方法を使用します。
    - IAM ポリシーシミュレーターを使用して IAM ポリシーをテストします。
    - さまざまな IAM ポリシータイプを確認します。
  - 必要に応じて、IAM ユーザーポリシーを編集します。
  - キューポリシーを確認し、必要に応じて編集します。
- プリンシパルが AWS サービスの場合、Amazon SQS キューポリシーは明示的にアクセスを許可する必要があります。
- プリンシパルがクロスアカウントプリンシパルである場合、Amazon SQS キューポリシーと IAM ポリシーの両方がアクセスを明示的に許可する必要があります。
- ポリシーで条件要素を使用している場合は、条件がアクセスを制限していることを確認します。

> **Important**
>
> いずれかのポリシー内の明示的な拒否は、明示的な許可に優先します。Amazon SQS ポリシーの基本的な例をいくつか示します。

AWS Key Management Service アクセス許可

Amazon SQS キューで、カスタマー管理のサーバー側の暗号化 (SSE) が有効になっている場合は AWS KMS key、プロデューサーとコンシューマーの両方にアクセス許可を付与する必要があります。キューが暗号化されているかどうかを確認するには、`GetQueueAttributes` API の `KmsMasterKeyId` 属性を使用するか、キューコンソールの [暗号化] を使用します。

- プロデューサーに必要なアクセス許可:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

- コンシューマーに必要なアクセス許可:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt"
],
"Resource": "<Key ARN>"
}
```

- クロスアカウントアクセスに必要なアクセス許可:

```
{
"Effect": "Allow",
"Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
    "kms:ReEncrypt",
    "kms:GenerateDataKey"
],
"Resource": "<Key ARN>"
}
```

Amazon SQS キューの暗号化を有効にするには、次のいずれかのオプションを選択します。

- SSE-Amazon SQS (Amazon SQS サービスで作成および管理する暗号化キー)
- AWS マネージドデフォルトキー (alias/aws/sqs)
- カスタマーマネージドキー

ただし、 AWSマネージド KMS キーを使用している場合、デフォルトのキーポリシーを変更することはできません。したがって、他のサービスやクロスアカウントへのアクセスを提供するには、カスタマーマネージドキーを使用します。これにより、キーポリシーを編集できます。

VPC エンドポイントポリシー

Amazon Virtual Private Cloud (Amazon VPC) エンドポイントを介して Amazon SQS にアクセスする場合、Amazon SQS の VPC エンドポイントポリシーでアクセスを許可する必要があります。Amazon SQS 用の Amazon VPC エンドポイントのポリシーを作成して、以下を指定できます。

1. アクションを実行できるプリンシパル。
2. 実行可能なアクション。
3. アクションを実行できるリソース。

次の例で、VPC エンドポイントポリシーは、Amazon SQS キュー *MyQueue* にメッセージを送信することを IAM ユーザー *MyUser* に許可するよう指定しています。その他のアクション、IAM ユーザー、Amazon SQS リソースは、VPC エンドポイント経由のアクセスが拒否されます。

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

組織のサービスコントロールポリシー

が組織に AWS アカウント 属している場合、 AWS Organizations ポリシーによって Amazon SQS キューへのアクセスがブロックされる可能性があります。デフォルトでは、 AWS Organizations ポリシーは Amazon SQS へのリクエストをブロックしません。ただし、Amazon SQS キューへのアクセスをブロックするように AWS Organizations ポリシーが設定されていないことを確認してください。 AWS Organizations ポリシーを確認する方法については、「 AWS Organizations ユーザーガイド」の「すべてのポリシーの一覧表示」を参照してください。

Amazon SQS API エラーのトラブルシューティング

以下のトピックでは、Amazon SQS API コールの実行時に返される最も一般的なエラーおよびトラブルシューティング方法について説明します。

QueueDoesNotExist エラー

このエラーは、Amazon SQS サービスが Amazon SQS アクションで指定されたキューを見つけられない場合に返されます。

考えられる原因と緩和策:

- リージョンが正しくない: Amazon SQS クライアント設定を参照して、クライアントで正しいリージョンを設定していることを確認します。クライアントでリージョンを設定しない場合、SDK または は設定ファイルまたは環境変数からリージョン AWS CLI を選択します。SDK は、設定ファイルにリージョンを見つけられない場合、デフォルトでリージョンを us-east-1 に設定します。
- キューが最近削除された可能性がある: API コールの実行前にキューが削除されていた場合、API コールはこのエラーを返します。エラーの発生前に DeleteQueue オペレーションがあったかどうかを CloudTrail で確認してください。
- アクセス許可の問題: リクエスト元の AWS Identity and Access Management (IAM) ユーザーまたはロールに必要なアクセス許可がない場合、次のエラーが返されることがあります。

  ```
  The specified queue does not exist or you do not have access to it.
  ```

  アクセス許可を確認し、正しいアクセス許可を使用して API コールを行ってください。

QueueDoesNotExist エラーのトラブルシューティングの詳細については、「AWS 情報センターガイド」の「Amazon SQS キューに API 呼び出しを行う際の QueueDoesNotExist エラーのトラブルシューティング方法を教えてください。」を参照してください。

InvalidAttributeValue エラー

このエラーは、誤ったポリシーまたはプリンシパルを使用して Amazon SQS キューリソースポリシーまたはプロパティを更新すると返されます。

考えられる原因と緩和策:

- 無効なリソースポリシー: リソースポリシーにすべての必要なフィールドがあることを確認します。詳細については、「IAM JSON ポリシー要素のリファレンス」と「IAM ポリシーの検証」を参照してください。IAM ポリシージェネレータを使用して、Amazon SQS リソースポリシーを作成およびテストすることもできます。ポリシーが JSON 形式であることを確認します。
- 無効なプリンシパル: `Principal` 要素がリソースポリシーに存在すること、および値が有効であることを確認します。Amazon SQS リソースポリシーの `Principal` 要素に IAM エンティティが含まれている場合は、ポリシーを使用する前にエンティティが存在することを確認してください。Amazon SQS はリソースポリシーを検証し、IAM エンティティをチェックします。IAM エンティティが存在しない場合、エラーが返されます。IAM エンティティを確認するには、`GetRole` API と `GetUser` API を使用します。

`InvalidAttributeValue` エラーのトラブルシューティング方法の詳細については、「AWS 情報センターガイド」の「Amazon SQS キューに API 呼び出しを行う際の QueueDoesNotExist エラーのトラブルシューティング方法を教えてください。」を参照してください。

ReceiptHandle エラー

`DeleteMessage` API コールを実行すると、受信ハンドルが正しくないか、期限切れである場合、エラーとして `ReceiptHandleIsInvalid` または `InvalidParameterValue` が返されることがあります。

- ReceiptHandleIsInvalid エラー: 受信ハンドルが正しくない場合、次の例のようなエラーが返されます。

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:
 The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- InvalidParameterValue エラー: 受信ハンドルが期限切れである場合、次の例のようなエラーが返されます。

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:
 Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The
 receipt handle has expired.
```

考えられる原因と緩和策:

受信ハンドルは受信メッセージごとに作成され、可視性タイムアウト期間のみ有効です。可視性タイムアウト期間が過ぎると、メッセージはコンシューマーのキューに表示されます。コンシューマーからメッセージを再度受信すると、新しい受信ハンドルを受け取ります。受信ハンドルの誤りや期限切れに伴うエラーを防ぐには、正しい受信ハンドルを使用して Amazon SQS キューの可視性タイムアウト期間内にメッセージを削除します。

`ReceiptHandle` エラーのトラブルシューティング方法の詳細については、「AWS 情報センターガイド」の「Amazon SQS DeleteMessage API を呼び出すときに、ReceipthHandleisInvalid エラーおよび InvalidParameterValue エラーのトラブルシューティングを行うにはどうすればよいですか？」を参照してください。

Amazon SQS デッドレターキューと DLQ 再処理の問題のトラブルシューティング

以下のトピックでは、Amazon SQS の DLQ および DLQ 再処理の問題に関する最も一般的な原因とトラブルシューティング方法を示します。詳細については、「AWS 情報センターガイド」の「Amazon SQS DLQ 再処理の問題をトラブルシューティングする方法を教えてください」を参照してください。

DLQ の問題

DLQ の一般的な問題および解決方法について説明します。

コンソールを使用してメッセージを表示すると、メッセージがデッドレターキューに移動される場合がある

Amazon SQS は、コンソールでのメッセージの表示回数を、対応するキューの再処理ポリシーに従ってカウントします。このため、コンソールでのメッセージの表示回数が、対応するキューの再処

理ポリシーに指定された回数に達すると、メッセージは対応するキューのデッドレターキューに移動されます。

この動作を調整するには、次のいずれかを実行します。

- 対応するキューの再処理ポリシーで、[最大受信数] 設定の値を大きくします。
- 対応するキューのメッセージをコンソールに表示しないようにします。

デッドレターキューの NumberOfMessagesSent と NumberOfMessagesReceived が一致しない

デッドレターキューに手動で送信したメッセージは、NumberOfMessagesSent メトリクスによってキャプチャされます。ただし、処理の試行が失敗した結果としてデッドレターキューにメッセージが送信された場合は、このメトリクスによってキャプチャされません。したがって、NumberOfMessagesSent と NumberOfMessagesReceived の値が異なることがあります。

デッドレターキューの再処理の作成と設定

デッドレターキューの再処理では、Amazon SQS でデッドレターキューからメッセージを受信して送信先キューに送信するために、適切なアクセス許可を設定する必要があります。正しいアクセス許可がない場合、デッドレターキューの再処理タスクは失敗する可能性があります。メッセージの再処理タスクのステータスを確認することで、問題を修正して再試行できます。

標準キューおよび FIFO キューのメッセージエラー処理

標準キューは、保持期間が期限切れになるまでメッセージを処理し続けます。この継続的な処理により、未消費のメッセージによってキューがブロックされる可能性が最小限に抑えられます。コンシューマーが多数のメッセージで繰り返し削除に失敗すると、コストが増加し、ハードウェアに余分な負荷がかかる場合があります。コストを抑えるには、失敗したメッセージをデッドレターキューに移動します。

標準キューは、多数のインフライトメッセージも許容します。メッセージの大部分が消費できず、デッドレターキューに送信されない場合、メッセージの処理速度が低下する可能性があります。キューの効率を維持するには、アプリケーションでメッセージ処理が正しく行われていることを確認してください。

FIFO キューは、メッセージグループからのメッセージを順番に消費することで、1 回のみ処理を行います。このため、キューをブロックするメッセージがあると、正常に処理されるか、デッドレター

キューに移動されるまで、メッセージグループは使用できなくなります。ただし、コンシューマーは別のメッセージグループからの順序付けされたメッセージを引き続き取得できます。

さらに、FIFO キューが許容するインフライトメッセージの数が減ります。FIFO キューがメッセージによってブロックされないようにするには、アプリケーションでメッセージ処理が正しく行われていることを確認してください。

詳細については、「Amazon SQS のメッセージキュー」および「Amazon SQS のベストプラクティス」を参照してください。

DLQ 再処理の問題

DLQ の一般的な再処理の問題および解決方法について説明します。

AccessDenied アクセス許可の問題

AWS Identity and Access Management (IAM) エンティティに必要なアクセス許可がないため、DLQ 再処理が失敗すると、`AccessDenied` エラーが発生します。

エラーメッセージの例:

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

DLQ 再処理リクエストを行うには、以下の API アクセス許可が必要です。

メッセージの再処理を開始するには:

- デッドレターキューのアクセス許可:
  - `sqs:StartMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt` – デッドレターキューまたは元のソースキューが暗号化されている場合。
- 送信先キューのアクセス許可:
  - `sqs:SendMessage`
  - `kms:GenerateDataKey` – 送信先キューが暗号化されている場合。
  - `kms:Decrypt` – 送信先キューが暗号化されている場合。

進行中のメッセージの再処理をキャンセルするには:

- デッドレターキューのアクセス許可:
  - `sqs:CancelMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt` – デッドレターキューまたは元のソースキューが暗号化されている場合。

メッセージの移動状況を表示するには:

- デッドレターキューのアクセス許可:
  - `sqs:ListMessageMoveTasks`
  - `sqs:GetQueueAttributes`

`NonExistentQueue` エラー

`NonExistentQueue` エラーは、Amazon SQS のソースキューが存在しないか、削除された場合に発生します。Amazon SQS キューが存在することを確認して再処理のために移動します。

エラーメッセージの例:

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSource エラー

以下のシナリオで DLQ 再処理を開始しようとすると `CouldNotDetermineMessageSource` エラーが発生します。

- `SendMessage` API を使用して Amazon SQS メッセージが DLQ に直接送信されている場合。
- DLQ が設定された Amazon Simple Notification Service (Amazon SNS) トピックまたは AWS Lambda 関数からのメッセージ。

このエラーを解決するには、再処理の開始時に [再処理のためにカスタム送信先に移動] を選択します。次に、Amazon SQS キューの ARN を入力して、DLQ のすべてのメッセージを送信先キューに移動します。

エラーメッセージの例:

```
Failed: CouldNotDetermineMessageSource
```

Amazon SQS での FIFO スロットリング問題のトラブルシューティング

デフォルトでは、FIFO キューは、SendMessage、ReceiveMessage、DeleteMessage の API アクションごとに 1 秒あたり 300 トランザクションをサポートします。キュー内のメッセージが利用可能であっても、300 TPS を超えるリクエストは ThrottlingException エラーになります。これを緩和するには、以下の方法を使用できます。

- Amazon SQS における FIFO キューの高スループットの有効化.
- Amazon SQS の API バッチアクション (SendMessageBatch、DeleteMessageBatch、ChangeMessageVisibilityBatch) を使用して、API アクションごとに 1 秒あたり最大 3,000 メッセージまで TPS 制限を引き上げ、コストを削減します。ReceiveMessage API の場合、トランザクションごとに最大 10 個のメッセージを受信するように MaxNumberofMessages パラメータを設定します。詳細については、「Amazon SQSのバッチアクション」を参照してください。
- 高スループットの FIFO キューの場合は、パーティション使用率を最適化するための推奨事項に従います。同じメッセージグループ ID を共有するメッセージは、バッチで送信します。同じ ReceiveMessage API リクエストからの受信ハンドルを使用して、メッセージをバッチで削除するか、メッセージの可視性タイムアウト値をバッチで変更します。
- 一意の MessageGroupId 値の数を増やします。これにより、FIFO キューパーティション間での均等な分散が可能になります。詳細については、「Amazon SQS メッセージグループ ID の使用」を参照してください。

詳細については、「AWS 情報センターガイド」の「Amazon SQS FIFO キューがすべてのメッセージまたは他のメッセージグループのメッセージを返すわけではないのはなぜですか?」を参照してください。

Amazon SQS の ReceiveMessage API コールで返されないメッセージのトラブルシューティング

以下のトピックでは、Amazon SQS メッセージがコンシューマーに返されない場合の最も一般的な原因およびトラブルシューティング方法について説明します。詳細については、「AWS 情報センターガイド」の「Amazon SQS キューからメッセージを受信できない理由を知りたいです。」を参照してください。

空のキュー

キューが空かどうかを判断するには、ロングポーリングを使用して `ReceiveMessage` API を呼び出します。CloudWatch メトリクス `ApproximateNumberOfMessagesVisible`、`ApproximateNumberOfMessagesNotVisible`、`Approx` を使用することもできます。すべてのメトリクス値が数分間 0 に設定されている場合、キューは空と見なされます。

インフライト制限に到達

ロングポーリングを使用すると、キューのインフライト制限 (デフォルトで FIFO は 20,000、標準は 120,000) を超えても、Amazon SQS はクォータ制限を超えたというエラーメッセージを返しません。

メッセージの遅延

Amazon SQS キューを遅延キューとして設定するか、メッセージをメッセージタイマー付きで送信した場合、遅延時間が終わるまでメッセージは表示されません。キューが遅延キューとして設定されているかどうかは、`GetQueueAttributes` API の `DelaySeconds` 属性またはキューコンソールの [配信遅延] で確認します。メッセージが遅延しているかどうかは、CloudWatch メトリクス `ApproximateNumberOfMessagesDelayed` で確認します。

メッセージはインフライト状態

別のコンシューマーがメッセージをポーリングすると、可視性タイムアウト期間中、メッセージはインフライト状態つまり非表示になります。以降のポーリングでは、空の受信が返される可能性があります。受信できるメッセージの数を確認するには、CloudWatch メトリクス ApproximateNumberOfMessagesVisible をチェックします。FIFO キューの場合、メッセージグループ ID を持つメッセージがインフライトになると、このメッセージが削除されるか表示されるまで、

追加のメッセージは返されません。これは、メッセージの順序付けが FIFO キューのメッセージグループレベルで維持されるためです。

ポーリング方法

ショートポーリングを使用している場合 (WaitTimeSeconds が 0)、Amazon SQS はサーバーのサブセットをサンプリングし、これらのサーバーからのメッセージのみを返します。したがって、メッセージが受信可能な場合でも、メッセージを取得できないことがあります。後続のポーリングリクエストではメッセージが返されます。

ロングポーリングを使用している場合、Amazon SQS はすべてのサーバーをポーリングし、利用可能なメッセージを 1 つ以上 (指定された最大数まで) 収集した後で、レスポンスを送信します。ReceiveMessage の WaitTimeSeconds の値が低すぎると、利用可能なメッセージのすべては受信できない可能性があります。

Amazon SQS のネットワークエラーのトラブルシューティング

以下のトピックでは、Amazon SQS のネットワーク問題の最も一般的な原因およびトラブルシューティング方法について説明します。

ETIMEOUT error

ETIMEOUT エラーは、クライアントが Amazon SQS エンドポイントへの TCP 接続を確立できない場合に発生します。

トラブルシューティング:

- ネットワーク接続を確認する

  `telnet` などのコマンドを実行して、Amazon SQS へのネットワーク接続をテストします。

  Example: telnet sqs.us-east-1.amazonaws.com 443
- ネットワーク設定を確認する
  - ローカルファイアウォールのルール、ルート、アクセスコントロールリスト (ACL) が、使用するポートのトラフィックを許可していることを確認します。
  - セキュリティグループのアウトバウンド (退出) ルールは、ポート 80 または 443 へのトラフィックを許可する必要があります。
  - ネットワーク ACL のアウトバウンド (退出) ルールは、TCP ポート 80 または 443 へのトラフィックを許可する必要があります。

- ネットワーク ACL のインバウンド (進入) ルールは、TCP ポート 1024-65535 でのトラフィックを許可する必要があります。
- パブリックインターネットに接続する Amazon Elastic Compute Cloud (Amazon EC2) インスタンスには、インターネット接続が必要です。

- Amazon Virtual Private Cloud (Amazon VPC) エンドポイント

  Amazon VPC エンドポイントを介して Amazon SQS にアクセスする場合、エンドポイントセキュリティグループは、ポート 443 でクライアントセキュリティグループへのインバウンドトラフィックを許可する必要があります。VPC エンドポイントのサブネットに関連付けられたネットワーク ACL には、以下の設定が必要です。

  - ネットワーク ACL のアウトバウンド (退出) ルールは、TCP ポート 1024-65535 (エフェメラルポート) でのトラフィックを許可する必要があります。
  - ネットワーク ACL のインバウンド (進入) ルールは、ポート 443 でのトラフィックを許可する必要があります。

また、Amazon SQS VPC エンドポイント AWS Identity and Access Management (IAM) ポリシーはアクセスを許可する必要があります。次の VPC エンドポイントポリシーの例では、Amazon SQS キュー *MyQueue* にメッセージを送信することを IAM ユーザー *MyUser* に許可するよう指定しています。その他のアクション、IAM ユーザー、Amazon SQS リソースは、VPC エンドポイント経由のアクセスが拒否されます。

```
{
    "Statement": [{
        "Action": ["sqs:SendMessage"],
        "Effect": "Allow",
        "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
        "Principal": {
            "AWS": "arn:aws:iam:123456789012:user/MyUser"
        }
    }]
}
```

UnknownHostException error

UnknownHostException エラーは、ホスト IP アドレスを確認できなかった場合に発生します。

トラブルシューティング:

nslookup ユーティリティを使用して、ホスト名に関連付けられた IP アドレスを返します。

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI または SDK for Python のレガシーエンドポイント：

```
nslookup <region>.queue.amazonaws.com
```

失敗した出力を受け取った場合は、「AWS 情報センターガイド」の「DNS の仕組み、および部分的または断続的な DNS 障害をトラブルシューティングする方法を教えてください。」の指示に従ってください。

有効な出力を受け取った場合は、アプリケーションレベルの問題である可能性があります。アプリケーションレベルの問題を解決するには、以下の方法を試してください。

- アプリケーションを再起動します。
- Java アプリケーションに不正な DNS キャッシュがないことを確認します。可能であれば、DNS TTL に準拠するようにアプリケーションを設定します。詳細については、「DNS 名検索の JVM TTL を設定する」を参照してください。

ネットワークエラーのトラブルシューティング方法の詳細については、「AWS 情報センターガイド」の「Amazon SQS の ETIMEOUT および UnknownHostException 接続エラーのトラブルシューティング方法を教えてください。」を参照してください。

Amazon Simple Queue ServiceのキューAWS X-Rayを使用した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、データを表示およびフィルタリングして、潜在的な問題と最適化の機会を特定できます。アプリケーションへのトレースされたリクエストについては、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 APIs に対して行う呼び出しに関する詳細情報を確認できます。

Amazon SQS を介して AWS X-Ray トレースヘッダーを送信するには、次のいずれかを実行します。

- `X-Amzn-Trace-Id` トレースヘッダーを使用する。
- `AWSTraceHeader` メッセージシステム属性を使用する。

エラーとレイテンシーに関するデータを収集するには、`AmazonSQS`AWS X-Ray SDKを使用してクライアントをインスツルメンテーションする必要があります。

AWS X-Ray コンソールを使用して、Amazon SQS と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「 をAmazon SQS AWS X-Rayに使用する」を参照してください。

Amazon SQSのセキュリティ

このセクションでは、Amazon SQSのセキュリティ、認証、アクセスコントロールおよびAmazon SQSアクセスポリシー言語について説明します。

トピック

- Amazon SQS でのデータ保護
- Amazon SQSでの Identity and Access Management
- Amazon SQSのロギングとモニタリング
- Amazon SQSのコンプライアンス検証
- Amazon SQSの耐障害性
- Amazon SQSのインフラストラクチャセキュリティ
- Amazon SQSのセキュリティベストプラクティス

Amazon SQS でのデータ保護

責任 AWS 共有モデル、Amazon Simple Queue Service 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。また、使用する「 AWS のサービス 」のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、データプライバシーに関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリティブログに投稿された AWS 責任共有モデルおよび GDPR のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーを設定することをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須で、TLS 1.3 をお勧めします。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。CloudTrail 証跡を使用して AWS アクティビティをキャプチャする方法については、「 AWS CloudTrail ユーザーガイド」のCloudTrail 証跡の使用」を参照してください。

- AWS 暗号化ソリューションと、その中のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度な管理されたセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API AWS を介して にアクセスするときに FIPS 140-3 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報を、タグ、または [名前] フィールドなどの自由形式のテキストフィールドに含めないことを強くお勧めします。これは、コンソール、API、または SDK を使用して Amazon SQS AWS CLIまたは他の AWS のサービス を使用する場合も同様です。 AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータは、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めします。

次のセクションでは、Amazon SQSのデータ保護について説明します。

Amazon SQS でのデータ暗号化

データ保護には、転送時 (Amazon SQS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 SQSデータセンターのディスクに保管されているとき) 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 SQS 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、メッセージを暗号化してから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するよう Amazon SQS にリクエストできます。Amazon SQS 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。

トピック

- Amazon SQS での保管中の暗号化
- Amazon SQS のキー管理

Amazon SQS での保管中の暗号化

サーバー側の暗号化 (SSE)では、機密データを暗号化されたキューで送信できます。SSE は、SQS マネージド暗号化キー (SSE-SQS) または (SSE- AWS Key Management Service KMS) で管理される

キーを使用して、キュー内のメッセージの内容を保護します。を使用して SSE を管理する方法については AWS Management Console、以下を参照してください。

- キューのSSE-SQSの設定(コンソール)
- キューのSSE-KMSの設定(コンソール)

(AWS SDK for Java および `CreateQueue`、、 `GetQueueAttributes`アクション) を使用して SSE を管理する方法については`SetQueueAttributes`、次の例を参照してください。

- Amazon SQS キューにおけるサーバー側の暗号化の使用
- の KMS アクセス許可の設定 AWS のサービス

Amazon SQS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、承認済みのコンシューマーに送信された場合のみ、Amazon SQSによって解読されます。

> **Important**
>
> SSE が有効なキューへのすべて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。
> デフォルトキー (Amazon SQS のAWS マネージド KMS キー) を使用する暗号化されたキューは、別の で Lambda 関数を呼び出すことはできません AWS アカウント。
> AWS Security Token Service `AssumeRole` アクションを使用して Amazon SQS に通知を送信できる AWS サービスの一部の機能は、SSE と互換性がありますが、標準キューでのみ動作します。
>
> - Auto Scalingライフサイクルフック
> - AWS Lambda デッドレターキュー
>
> 暗号化されたキューとの他のサービスとの互換性については、AWS サービスの KMS アクセス許可を設定するとサービスドキュメント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て、クラウド向けにスケーリングされたキー管理システムを提供します。で Amazon SQS 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 keys を作成および管理できます。
- Amazon SQS の AWS マネージド KMS キーを使用することもできます。これは、アカウントとリージョンごとに一意です。
- 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「AWS Key Management Service とは」を参照してください。

暗号化スコープ

SSEでは、Amazon SQS キューのメッセージ本文が暗号化されます。

SSEでは、以下は暗号化されません。

- キューのメタデータ (キュー名と属性)
- メッセージのメタデータ (メッセージ ID、タイムスタンプ、属性)
- キューごとのメトリクス

メッセージを暗号化すると、未承認ユーザーまたは匿名ユーザーはそのメッセージを利用できなくなります。SSE を有効にすると、暗号化されたキューへの匿名の `SendMessage` リクエストと `ReceiveMessage` リクエストは拒否されます。Amazon SQS セキュリティベストプラクティスでは、匿名リクエストを使用しないことを推奨しています。Amazon SQS キューに匿名リクエストを送信する場合は、必ず SSE を無効にしてください。これは、Amazon SQSの正常な機能には影響しません。

- メッセージが暗号化されるのは、キュー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 SQSは、バックログされたメッセージを暗号化しません。
- キュー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。

メッセージをデッドレターキューに移動しても、暗号化には影響しません。

- 暗号化された出典キューから暗号化されていないデッドレターキューにメッセージがAmazon SQS によって移動された場合も、メッセージは暗号化された状態で維持されます。
- 暗号化されていない出典キューから暗号化されたデッドレターキューにメッセージがAmazon SQSによって移動された場合、メッセージは暗号化されていない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 キューサ - ビス API リファレンス」を参照してください。

データキー

キー (DEK) は、Amazon SQSメッセージの内容を暗号化します。

詳細については、AWS Encryption SDK デベロッパーガイド。の「AWS Key Management Service デベロッパーガイドのデータキー」を参照してください。

データキー再利用期間

Amazon SQS がデータキーを再利用して、 AWS KMS 再度 を呼び出す前にメッセージを暗号化または復号できる秒単位の時間の長さ。60秒(1分)から86,400秒(24時間)の秒数を表す整数。デフォルトは300(5分)です。詳細については、「データキー再利用期間について」を参照してください。

> **Note**
>
> 万一 に到達できない場合 AWS KMS、Amazon SQS は接続が再確立されるまでキャッシュされたデータキーを使用し続けます。

KMS キー ID

アカウントまたは別のアカウントの AWS マネージド KMS キーまたはカスタム KMS キーのエイリアス、エイリアス ARN、キー ID、またはキー ARN。Amazon SQS の AWS マネージド KMS キーのエイリアスは常に ですが`alias/aws/sqs`、カスタム KMS キーのエイリアスは などです`alias/MyAlias`。これらの KMS キーで Amazon SQS キュー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。
>
> - カスタム KMS キーを指定しない場合、Amazon SQS は Amazon SQS の AWS マネージド KMS キーを使用します。 Amazon SQS
> - AWS Management Console を使用してキューの Amazon SQS の AWS マネージド KMS キーを初めて指定すると、 AWS KMS は Amazon SQS の AWS マネージド KMS キーを作成します。

- または、SSE が有効になっているキューで `SendMessage`または `SendMessageBatch`アクションを初めて使用すると、 は Amazon SQS の AWS マネージド KMS キー AWS KMS を作成します。

コンソールのカスタマーマネージドキーセクションまたは `CreateKey` AWS KMS アクションを使用して、KMS キーの作成、KMS キーの使用方法を制御するポリシーの定義、KMS キーの使用状況の AWS KMS 監査を行うことができます。詳しい情報については、AWS Key Management Service デベロッパーガイドの KMS キーおよびキー作成をご参照ください。KMS キー識別子の詳しい例については、AWS Key Management Service API リファレンスの KeyId をご参照ください。KMS キー識別子の検索については、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。

エンベロープ暗号化

暗号化されたデータのセキュリティは、復号できるデータキーを保護することによって部分的に異なります。Amazon SQS は KMS キーを使用してデータキーを暗号化し、暗号化されたデータキーは暗号化されたメッセージと一緒に保存されます。データキーを暗号化するために KMS キーを使用するこの方法は、エンベロープ暗号化と呼ばれます。

詳細については、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。

Amazon SQS のキー管理

Amazon SQS は AWS Key Management Service (KMS) と統合して、サーバー側の暗号化 (SSE) 用の KMS キーを管理します。SSE 情報およびキー管理の定義については、「Amazon SQS での保管中の暗号化」をご参照ください。Amazon SQS は KMS キーでメッセージを暗号化および復号するデータキーを検証および保護します。以下のセクションは Amazon SQS サービスの KMS キーとデータキーの操作について説明します。

AWS KMS 許可を設定する

すべての KMS キーにはキーポリシーが必要です。Amazon SQS の AWS マネージド KMS キーのキーポリシーを変更することはできません。この KMS キーのポリシーには、暗号化されたキューを使用するためのアカウント内 (Amazon SQS の使用が承認されているもの) にあるすべてのプリンシパルのアクセス許可が含まれています。

Amazon SQS は、異なる AWS アカウント、IAM ユーザー、または IAM ロールを使用しているかどうかにかかわらず、 AWS 認証情報に基づいて発信者を区別します。さらに、異なるスコープポリシーを持つ同じ IAM ロールは、個別のリクエスタとして扱われます。ただし、IAM ロールセッションを使用する場合、同じ IAM ロールとスコープポリシー`RoleSessionName`を維持しながら のみを変更しても、個別のリクエスタは作成されません。したがって、 AWS KMS キーポリシープリンシパルを指定する場合、これらのセッションは同じリクエスタとして扱われるため、`RoleSessionName`違いのみに依存することは避けてください。

または、必要な許可をIAM ポリシーで指定して、暗号化されたメッセージを作成および使用するプリンシパルにこのポリシーを割り当てます。詳細については、デベロッパーガイド AWS KMS の「AWS Key Management Service IAM ポリシーの使用」を参照してください。

> **Note**
>
> Amazon SQS との間で送受信するグローバルアクセス許可を設定できますが、 では、IAM ポリシーの `Resource`セクションの特定のリージョンで KMS キーの完全な ARN に明示的に名前を付ける AWS KMS 必要があります。

AWS サービスの KMS アクセス許可を設定する

いくつかの AWS サービスは、Amazon SQS キューにイベントを送信できるイベントソースとして機能します。これらのイベントソースが暗号化されたキューと連携できるようにするには、カスタマーマネージド KMS キーを作成し、必要な AWS KMS API メソッドを使用するためのアクセス許可をサービスのキーポリシーに追加する必要があります。アクセス許可を設定するには、次の手順を実行します。

> **Warning**
>
> Amazon SQS メッセージを暗号化するための KMS キーを変更する場合、古い KMS キーで暗号化した既存のメッセージはそのキーで暗号化されたままになることに注意してください。これらのメッセージを復号するには、古い KMS キーを保持していて、そのキーポリ

シーで kms:Decrypt および kms:GenerateDataKey へのアクセス許可を Amazon SQS に付与していることを確認する必要があります。新しいメッセージを暗号化するために新しい KMS キーに更新したら、古い KMS キーで暗号化したすべての既存のメッセージを処理してキューから削除したことを確認してから、古い KMS キーを削除または無効にします。

1. カスタマーマネージド KMS キーを作成。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 サービスイベントソースが kms:Decryptおよび kms:GenerateDataKey API メソッドを使用できるようにするには、KMS キーポリシーに次のステートメントを追加します。

   JSON

```
{
   "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Principal": {
            "Service": "service.amazonaws.com"
         },
         "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
         ],
         "Resource": "*"
       }]
}
```

上記の例の「Service」をイベント出典のサービス名に置き換えます。イベントソースには、次のサービスが含まれます。

イベントソース	サービス名
Amazon CloudWatch Events	events.amazonaws.com
Amazon S3イベント通知について	s3.amazonaws.com

イベントソース	サービス名
Amazon SNS トピックのサブスクリプション	sns.amazonaws.com

3. KMS キーの ARN を使用して既存の SSE キューを設定します。
4. 暗号化されたキューのARNをイベント出典に追加します。

プロデューサーの AWS KMS アクセス許可を設定する

データキー再利用期間が終了した場合、プロデューサーによる次回のSendMessageまたはSendMessageBatchの呼び出しでは、kms:Decryptとkms:GenerateDataKeyの呼び出しもトリガーされます。kms:Decryptの呼び出しでは、新しいデータキーを使用する前に整合性を検証します。したがって、プロデューサーは KMS キーの kms:Decrypt および kms:GenerateDataKey 許可が必要です。

次のステートメントをプロデューサーの IAMポリシーに追加します。キーリソースとキューリソースには正しい ARN値を使用してください。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:SendMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
        }
    ]
```

```
}
```

コンシューマーの AWS KMS アクセス許可を設定する

データキー再利用期間が終了した場合、コンシューマーが次回に `ReceiveMessage` を呼び出すと、それを使用する前に新しいデータキーの整合性を検証するために、`kms:Decrypt` の呼び出しもトリガーされます。したがって、コンシューマーは指定されたキューのメッセージの暗号化に使用される KMS キーへの `kms:Decrypt` 許可が必要です。キューがデッドレターキューとして機能する場合、コンシューマーはソースキューのメッセージの暗号化に使用される KMS キーへの `kms:Decrypt` 許可も必要です。次のステートメントをコンシューマーの IAM ポリシーに追加します。キーリソースとキューリソースには正しい ARN値を使用してください。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sqs:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
        }
    ]
}
```

混乱した代理の保護で AWS KMS アクセス許可を設定する

キーポリシーステートメントのプリンシパルが AWS サービスプリンシパルの場合、`aws:SourceArn` または `aws:SourceAccount` のグローバル条件キーを使用して混乱した使節問題から保護できます。これらの条件キーを使用する場合、暗号化されているリソースの

Amazon リソースネーム (ARN) の値を設定します。リソースの ARN が不明の場合は、代わりに `aws:SourceAccount` を使用してください。

この KMS キーポリシーでは、アカウント 111122223333 が保有されたサービスの特定リソースは、`Decrypt` および `GenerateDataKey` アクションにおいて KMS を呼び出すことが許可されます。これは Amazon SQS の SSE 使用中に発生します。

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "<replaceable>service</replaceable>.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:<replaceable>service<::111122223333:resource"
                    ]
                }
            }
        }
    ]
}
```

SSE を有効にした Amazon SQS キューを使用する場合、以下のサービスが `aws:SourceArn` をサポートします:

- Amazon SNS
- Amazon S3
- CloudWatch Events

- AWS Lambda
- CodeBuild
- Amazon Connect Customer Profiles
- AWS Auto Scaling
- Amazon Chime

データキー再利用期間について

データキー再利用期間は、Amazon SQSが同じデータキーを再利用するための最大期間を定義します。データキー再利用期間が終了すると、Amazon SQSは新しいデータキーを生成します。再利用期間については、次のガイドラインに注意してください。

- 再利用期間が短いほどセキュリティが向上しますが、 への呼び出しが増え AWS KMS、無料利用枠を超えて料金が発生する可能性があります。
- データキーは暗号化用と復号化用に別々にキャッシュされますが、再利用期間はデータキーの両方のコピーに適用されます。
- データキーの再利用期間が終了すると、 `SendMessage`または への次の呼び出しは、`SendMessageBatch`通常、 メソッドへの呼び出しを AWS KMS `GenerateDataKey`トリガーして新しいデータキーを取得します。また、次に `SendMessage`と を呼び出すと`ReceiveMessage`、それぞれが の AWS KMS 呼び出しをトリガー`Decrypt`して、データキーを使用する前にデータキーの整合性を検証します。
- プリンシパル (AWS アカウント またはユーザー) はデータキーを共有しません (一意のプリンシパルによって送信されるメッセージは常に一意のデータキーを取得します) 。したがって、 への呼び出しのボリューム AWS KMS は、データキーの再利用期間中に使用されている一意のプリンシパルの数の倍数です。

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 SQS が KMS キーを使用する頻度を知りたい場合があります。

> **Note**
>
> コストは以下の計算式でかなり正確に予測できますが、Amazon SQSの分散性により、実際のコストの方が高くなることがあります。

APIリクエスト(R) キュー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 + C)
```

B は請求期間(秒)です。

D は、データキー再利用期間(秒)です。

Pはメインの生産数は Amazon SQS キューに送信されます。

C は、Amazon SQSキューから受信する、コンシューマー側のプリンシパル数です。

> **Important**
> 一般的に、プロデューサー側プリンシパルで発生するコストはコンシューマー側プリンシパルの倍程度になります。詳細については、「データキー再利用期間について」を参照してください。
> プロデューサーとコンシューマーの ユーザーが異なる場合、コストは増加します。

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: 2 つのプリンシパルと 1 つのキューの AWS KMS API コールの数を計算する

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)に設定されています。
- キューの数は 1 つです。
- プロデューサー側プリンシパルが1つ、コンシューマー側プリンシパルが1つあります。

```
(2,678,400 / 300) * (2 * 1 + 1) = 26,784
```

例 2: 複数のプロデューサーとコンシューマー、2 つのキューの AWS KMS API コールの数を計算する

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24時間(86,400秒)に設定されています。
- キューの数は2つです。
- 1つ目のキューのプロデューサー側プリンシパルは3つ、コンシューマー側プリンシパルは1つです。
- 2つ目のキューのプロデューサー側プリンシパルは5つ、コンシューマー側プリンシパルは2つです。

```
(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532
```

AWS KMS エラー

Amazon SQS および を使用すると AWS KMS、エラーが発生する可能性があります。次のリファレンスでは、エラーおよび考えられるトラブルシューティング方法について説明します。

- AWS KMS の一般的なエラー
- AWS KMS 復号エラー
- AWS KMS GenerateDataKey エラー

Amazon SQS でのインターネットワークトラフィックのプライバシー

Amazon SQSの Amazon Virtual Private Cloud (Amazon VPC) エンドポイントは、Amazon SQS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 SQS 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 SQSのAmazon Virtual Private Cloud エンドポイント

Amazon VPC を使用して AWS リソースをホストする場合は、VPC と Amazon SQS 間の接続を確立できます。この接続を使用して、公開インターネットと交差せずにAmazon SQSキューにメッセージを送信できます。

Amazon VPC では、カスタム仮想ネットワークで AWS リソースを起動できます。VPC を使用して、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。Amazon VPCの詳細については、Amazon VPC ユーザーガイドを参照してください。

VPCを Amazon SQSに接続するには、最初にインターフェイス VPC エンドポイントを使用すると、VPCを他の AWS のサービス VPC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 SQS 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「チュートリアル:Amazon Virtual Private Cloud から Amazon SQSキューにメッセージを送信するそして例5:VPC エンドポイントからではない場合はアクセスを拒否するこのガイドのインターフェイス VPC エンドポイント (AWS PrivateLink)の」を参照してください。

> **Important**
>
> - Amazon Virtual Private Cloud は HTTPS Amazon SQS エンドポイントでのみ使用できます。
> - Amazon VPC からメッセージを送信するように Amazon SQS を設定する場合は、プライベート DNS を有効にし、デュアルスタックエンドポイント`sqs.`*`us-east-2`*`.api.aws`の`sqs.`*`us-east-2`*`.amazonaws.com`または 形式でエンドポイントを指定する必要があります。
> - Amazon SQS は、 エンドポイントサービスを使用した PrivateLink を介した FIPS `com.amazonaws.region.sqs-fips`エンドポイントもサポートしています。形式の FIPS エンドポイントに接続できます`sqs-fips.region.amazonaws.com`。
> - Amazon Virtual Private Cloud でデュアルスタックエンドポイントを使用する場合、リクエストは IPv4 と IPv6 を使用して送信されます。
> - 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。

Amazon SQS用のAmazon VPCエンドポイントポリシーを作成する

Amazon SQSのAmazon VPC エンドポイントに対するポリシーを作成して、以下を特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。
- 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「VPC エンドポイントによるサービスのアクセスコントロール」を参照してください。

次の VPC エンドポイントポリシーの例では、Amazon SQS キュー `MyQueue` にメッセージを送信することを `MyUser` ユーザーに許可するように指定しています。

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

以下は拒否されます:

- `sqs:CreateQueue`や`sqs:DeleteQueue`など他の Amazon SQS API アクション
- この VPC エンドポイントを使用しようとする の他のユーザーおよびルール。
- `MyUser`別のAmazon SQS キューにメッセージを送信する。

> **Note**
>
> ユーザーはさらに、他の Amazon SQS API アクションを VPC の外側からでも使用できます。詳細については、「例5:VPC エンドポイントからではない場合はアクセスを拒否する」を参照してください。

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 SQS 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。デュアルスタックおよび FIPS エンドポイントの詳細については、 SDK リファレンスガイドを参照してください。

Amazon SQS 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は、命名規則 を使用しますsqs.*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、sqs.*eu-west-1*.amazonaws.com です。

Amazon SQS 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。

Amazon SQSでの Identity and Access Management

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 SQS 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。

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon SQS で行う作業によって異なります。

サービスユーザー – 業務を行うために Amazon SQS サービスを使用する場合、管理者が必要な認証情報および許可を提供します。業務のために使用する Amazon SQS 機能が増えるにつれ、追加の許可が必要な場合があります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。Amazon SQS の機能にアクセスできない場合は、「Amazon Simple Queue 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」を参照してください。

サービス管理者 - 社内の Amazon SQS リソースを担当している場合は、通常、Amazon SQS へのフルアクセスがあります。サービスのユーザーがどの Amazon SQS 機能やリソースにアクセスするかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユーザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。会社で Amazon SQS と IAM を併用する方法の詳細については、「Amazon Simple Queue Service で IAM を使用する方法」を参照してください。

IAM 管理者 - 管理者は、Amazon SQS へのアクセス権を管理するポリシーの作成方法の詳細について確認する場合があります。IAM で使用可能な、Amazon SQS アイデンティティベースのポリシーの例を確認するには、「ポリシーに関するベストプラクティス」を参照してください。

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。として、IAM ユーザーとして AWS アカウントのルートユーザー、または IAM ロールを引き受けることによって認証（にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center（IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook 認証情報は、フェデレーション ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して にアクセスすると、間接的 AWS にロールを引き受けることになります。

ユーザーの種類に応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド」の「 へのサインイン AWS アカウント方法」を参照してください。

AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストを暗号化して署名します。AWS ツールを使用しない場合は、自分でリクエストに署名する必要があります。リクエストに自分で署名する推奨方法の使用については、「IAM ユーザーガイド」の「API リクエストに対するAWS Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。たとえば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを強化することをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「IAM のAWS 多要素認証」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、アカウント内のすべての およびリソースへの AWS のサービス完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。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。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「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。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、ID プロバイダーとのフェデレーションを使用して一時的な認証情報 AWS のサービス を使用して にアクセスすることを要求します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、、AWS Directory Serviceアイデンティティセンターディレクトリ、または ID ソースを介して提供された認証情報 AWS のサービス を使用して にアクセスするすべてのユーザーです。フェデレーティッド ID がアクセスすると AWS アカウント、ロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成するか、独自の ID ソースのユーザーとグループのセットに接続して同期し、すべての AWS アカウント とアプリケーションで使用できます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「What is IAM Identity Center?」(IAM Identity Center 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、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内の ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーション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「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。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 AWS アカウント を持つ 内の ID です。これは IAM ユーザーに似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受けるには AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替えることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「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 ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールについては、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールを作成する」を参照してください。IAM Identity Center を使用する場合は、許可セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。
- 一時的な IAM ユーザー権限 - IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (プロキシとしてロールを使用する代わりに) リソースに直接ポリシーをアタッチできます。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。
- クロスサービスアクセス – 一部の は他の の機能 AWS のサービス を使用します AWS のサービス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用してアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を 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 と組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。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。

- サービスロール - 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、削除することができます。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。
- 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、サービスリンク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。

- Amazon EC2 で実行されているアプリケーション – IAM ロールを使用して、EC2 インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、「IAM ユーザーガイド」の「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して許可を付与する」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 AWS 、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ときにアクセス許可を定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) 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。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。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構造と内容の詳細については、「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 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。

デフォルトでは、ユーザーやロールに権限はありません。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。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

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ポリシーは、IAM ユーザーグループ、ユーザーのグループ、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。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたは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。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制御できます。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は、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。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

アクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的でない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。`Principal` フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPsは、 の組織または組織単位 (OU) の最大アクセス許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の をグループ化して一元管理するためのサービス AWS アカウント です。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー (SCP)」を参照してください。
- リソースコントロールポリシー (RCP) – RCP は、所有する各リソースにアタッチされた IAM ポリシーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定するために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースのアクセス許可を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、 を含む ID の有効なアクセス許可に影響を与える可能性があります。RCP をサポートする のリストを含む Organizations と RCP の詳細については、AWS Organizations RCPs「リソースコントロールポリシー (RCPs」を参照してください。 AWS のサービス

- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。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権限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。

Amazon SQS でのアクセス管理の概要

すべての AWS リソースは によって所有され AWS アカウント、リソースを作成またはアクセスするアクセス許可はアクセス許可ポリシーによって管理されます。アカウント管理者は、IAMアクセス権限ポリシーをアイデンティティ (ユーザー、グループ、ロール) にアタッチできます。一部のサービス (Amazon SQS など) では、アクセス権限ポリシーをリソースにアタッチすることもできます。

> **Note**
>
> アカウント管理者 (または管理者ユーザー) は、管理者権限を持つユーザーです。詳細については、「IAM ユーザーガイド」の「IAM のベストプラクティス」を参照してください。

アクセス権限を付与する場合、アクセス権限を取得するユーザー、取得するアクセス権限の対象となるリソース、およびリソースに対して許可される特定のアクションを指定します。

Amazon Simpleキューサービス リソースと操作

Amazon SQS では、唯一のリソースはキューです。ポリシーで Amazon リソースネーム (ARN)を使用して、ポリシーを適用するリソースを識別します。次のリソースには、関連付けられた一意の ARN があります。

リソースタイプ	ARN 形式
キュー	`arn:aws:sqs:` *`region`*`:`*`account_id`* `:`*`queue_name`*

キューの ARN 形式の例を以下に示します。

- AWS アカウント 123456789012 に属する、米国東部 (オハイオ) リージョンmy_queueの という名前のキューの ARN。

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- Amazon SQSがサポートする異なるリージョンごとのmy_queueというキューのARN。

```
arn:aws:sqs:*:123456789012:my_queue
```

- キュー名に対して*または?をワイルドカードとして使用する ARN。次の例で、ARN はプレフィックス my_prefix_ が付いたすべてのキューに一致します。

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

GetQueueAttributes アクションを呼び出して既存のキューの ARN 値を取得できます。QueueArn 属性の値は、キューの ARN です。ARNの詳細については、IAM ユーザーガイドの「IAM ARN」を参照してください。

Amazon SQSには、キューリソースを操作するための一連の アクションが用意されています。詳細については、「Amazon SQS (Amazon SQS)、 API のアクセス権限: アクションとリソースのリファレンスについて」を参照してください。

リソース所有権についての理解

は、リソースを作成したユーザーに関係なく、アカウントで作成されたリソース AWS アカウントを所有します。具体的には、リソース所有者は、リソースの作成リクエストを認証するプリンシパルエンティティ (ルートアカウント、ユーザー、または IAM ロール) の AWS アカウント です。次の例は、この仕組みを示しています。

- のルートアカウントの認証情報を使用して Amazon SQS キュー AWS アカウント を作成する場合、 AWS アカウント はリソースの所有者です (Amazon SQS では、リソースは Amazon SQS キューです) 。
- でユーザーを作成し AWS アカウント 、そのユーザーにキューを作成するアクセス許可を付与すると、そのユーザーはキューを作成できます。ただし、(ユーザーが属する) AWS アカウント アカウントはキューリソースを所有しています。
- Amazon SQS キューを作成するアクセス許可 AWS アカウント を持つ で IAM ロールを作成する場合、ロールを引き受けることのできるすべてのユーザーがキューを作成できます。 (ロールが属 AWS アカウント する) がキューリソースを所有します。

リソースへのアクセスの管理

アクセス許可ポリシー では、誰が何にアクセスできるかを記述します。以下のセクションで、アクセス権限のポリシーを作成するために使用可能なオプションについて説明します。

> **Note**
>
> このセクションでは、Amazon SQSのコンテキストでの IAM の使用について説明します。これは、IAMサービスに関する詳細情報を取得できません。IAM に関する詳細なドキュメントについては、「IAM ユーザーガイド」の「What is IAM?」(IAM とは？) を参照してください。IAM ポリシーの構文と説明については、「IAM ユーザーガイド」の「AWS IAM ポリシーリファレンス」を参照してください。

IAM ID にアタッチされたポリシーは ID ベースのポリシー (IAM ポリシー) と呼ばれ、リソースにアタッチされたポリシーはリソースベースのポリシーと呼ばれます。

アイデンティティベースのポリシー

ユーザーにAmazon SQSキューのアクセス権限を付与する方法は、 ポリシーシステムを使用する方法とIAMポリシーシステムを使用する方法の 2 つです。いずれかのシステムまたは両方を使用して、ユーザーまたはロールにポリシーをアタッチできます。ほとんどの場合、どちらのシステムを使用しても同じ結果が得られます。例えば、次の操作を実行できます:

- アカウントのユーザーまたはグループに許可ポリシーをアタッチする–Amazon SQS キューを作成する許可を付与するために、ユーザーまたはユーザーが所属するグループに許可ポリシーをアタッチできます。

- 別の のユーザーに許可ポリシーを AWS アカウントアタッチする – 別の のユーザーに許可ポリシーをアタッチ AWS アカウント して、ユーザーが Amazon SQS キューとやり取りできるようにします。ただし、クロスアカウントアクセス許可は、以下のアクションには適用されません。

  クロスアカウント権限は、次のアクションには適用されません。

  - AddPermission
  - CancelMessageMoveTask
  - CreateQueue
  - DeleteQueue
  - ListMessageMoveTask
  - ListQueues
  - ListQueueTags
  - RemovePermission
  - SetQueueAttributes
  - StartMessageMoveTask
  - TagQueue
  - UntagQueue

  これらのアクションへのアクセスを許可するには、ユーザーは Amazon SQS キューを所有 AWS アカウント するのと同じ に属している必要があります。

- アクセス許可ポリシーをロールにアタッチする (クロスアカウントアクセス許可を付与する) – SQS キューにクロスアカウントアクセス許可を付与するには、IAM ポリシーとリソースベースのポリシーの両方を組み合わせる必要があります。

  1. アカウント A (キューを所有) の場合：
     - リソースベースのポリシーを SQS キューにアタッチします。このポリシーは、アカウント B のプリンシパル (IAM ロールなどReceiveMessage) に必要なアクセス許可 (SendMessage、 など) を明示的に付与する必要があります。
  2. アカウント A で、IAM ロールを作成します。
     - アカウント B または がロールを引き受け AWS のサービス ることを許可する信頼ポリシー。

> **Note**
>
> (Lambda や EventBridge AWS のサービス など) がロールを引き受けるようにする場合は、信頼ポリシーでサービスプリンシパル (などlambda.amazonaws.com) を指定します。

- 引き受けたロールにキューを操作するアクセス許可を付与するアイデンティティベースのポリシー。

3. アカウント B で、アカウント A でロールを引き受けるアクセス許可を付与します。

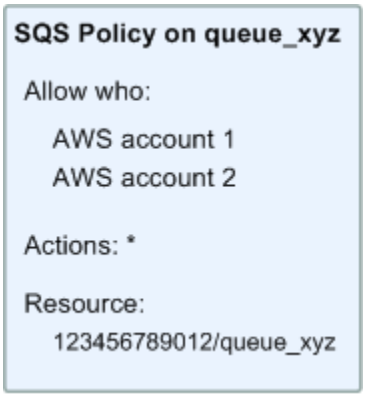
クロスアカウントプリンシパルを許可するようにキューのアクセスポリシーを設定する必要があります。IAM アイデンティティベースのポリシーだけでは、SQS キューへのクロスアカウントアクセスには不十分です。

IAM を使用した許可の委任の詳細については、「IAM ユーザーガイド」の「アクセス管理」を参照してください。

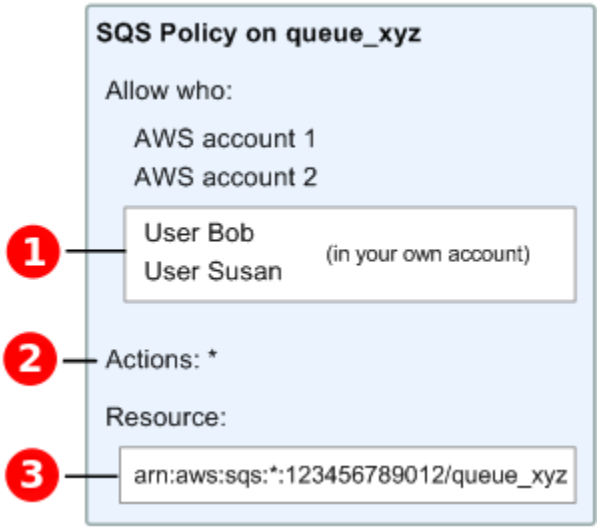
Amazon SQS はIAMポリシーを使用して作業する一方で、独自のポリシーインフラストラクチャがあります。Amazon SQS ポリシーをキューとともに使用して、キューにアクセスできる AWS アカウントを指定できます。アクセスタイプと条件を指定できます (たとえば、リクエストが 2010年12月31日より前の場合は`SendMessage`、`ReceiveMessage`を使用するアクセス権限を付与する条件)。アクセス許可を付与できる特定のアクションは、Amazon SQS アクションリスト全体のサブセットです。Amazon SQS ポリシーを記述して、`*`「すべての アクションを許可」を意味するを指定した場合、ユーザーがこのサブセット内のすべてのアクション実行できることを意味します。

次の図は、これらのベーシックなAmazon SQS ポリシーのうち1つの概念を表しており、アクションのサブセットを取り上げています。ポリシーは 用であり`queue_xyz`、 AWS アカウント 1 と AWS アカウント 2 に、指定されたキューで許可されたアクションのいずれかを使用するアクセス許可を付与します。

> **Note**
>
> ポリシーのリソースは として指定されます。ここで`123456789012/queue_xyz`、`123456789012`はキューを所有するアカウントのアカウント AWS ID です。



IAMと、ユーザーおよび Amazon リソースネーム (ARN)の概念の導入により、SQSポリシーに関するいくつかの点が変わりました。次の図と表は、その変更を示しています。



1

異なるアカウントのユーザーにアクセス許可を付与する方法については、IAM ユーザーガイドの「チュートリアル: IAM ロールを使用して AWS アカウント間でアクセスを委任する」を参照してください。

2

*に含まれるアクションのサブセットが拡大されました。可能なアクションの一覧については、Amazon SQS (Amazon SQS)、 API のアクセス権限: アクションとリソースのリファレンスについてを参照してください。

3

Amazo

リソースネーム (ARN)を使用してリソースを指定できます。これは、IAMポリシーでリソースを指

定するためのスタンダードな方法です。Amazon SQSキューのARN形式については、「Amazon Simpleキューサービス リソースと操作」を参照してください。

例えば、前の図の Amazon SQS ポリシーによると、 AWS アカウント 1 または AWS アカウント 2 のセキュリティ認証情報を所有するユーザーは誰でも にアクセスできますqueue_xyz。さらに、自身の AWS アカウント (ID 123456789012)内のユーザー Bobおよび Susanもキューにアクセスできます。

IAMの導入前は、 Amazon SQSによりキューの作成者に、キューに対する完全なコントロールが付与されていました (そのキューで使用できるすべての Amazon SQ アクションへのアクセス)。これは、作成者が AWS セキュリティ認証情報を使用している場合以外は当てはまらなくなりました。キューを作成するアクセス権限を持つユーザーは、作成されたキューで何らかの操作を実行するには、他のAmazon SQSアクションを使用するアクセス権限も持っている必要があります。

以下に、ユーザーにすべてのAmazon SQSアクションを許可するが、対象を名前にリテラル文字列というプレフィックスがついているキューに限るポリシーの例を示しますbob_queue_。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
   }]
}
```

詳細については、IAMユーザーガイドAmazon SQS でのポリシーの使用「ID (ユーザー、グループ、ロール)」とのIAM ポリシーの概要」を参照してください。

ポリシー要素の指定:アクション、効果、リソース、プリンシパル

Amazon Simple キューサービス リソースの種類ごとに、このサービスは、一連のアクションを定義します。これらの アクションを実行するためのアクセス権限を付与するために、Amazon SQSではポリシーに一連のアクションを定義できます。

> **Note**
>
> 1つの アクションの実行で、複数のアクションのアクセス権限が必要になる場合があります。特定のアクションのアクセス権限を付与した場合は、アクションを許可または拒否するリソースを識別します。

最も基本的なポリシーの要素を次に示します。

- リソース–ポリシーで Amazon リソースネーム (ARN)を使用して、ポリシーを適用するリソースを識別します。
- アクション–アクションのキーワードを使用して、許可または拒否するリソースアクションを識別します。たとえば、`sqs:CreateQueue`権限は、Amazon Simple キューサービス`CreateQueue`アクション の実行をユーザーに許可します。
- 効果–ユーザーが特定のアクションを要求する際の効果を指定します。許可または拒否のいずれかになります。リソースへのアクセスを明示的に許可していない場合、アクセスは暗黙的に拒否されます。また、明示的にリソースへのアクセスを拒否すると、別のポリシーによってアクセスが許可されている場合でも、ユーザーはそのリソースにアクセスできなくなります。
- プリンシパル-アイデンティティベースのポリシー (IAMポリシー) で、ポリシーが添付されているユーザーが黙示的なプリンシパルとなります。リソースベースのポリシーでは、アクセス許可 (リソースベースのポリシーにのみ適用) を受け取りたいユーザー、アカウント、サービス、またはその他のエンティティを指定します。

Amazon SQS ポリシーの構文と記述の詳細については、IAM ユーザーガイドのAWS IAM ポリシーリファレンスを参照してください。

すべてのAmazon Simpleキューサービスアクションおよびそれに適用されるリソースを示す表については、「Amazon SQS (Amazon SQS)、 API のアクセス権限: アクションとリソースのリファレンスについて」を参照してください。

Amazon Simple Queue Service で IAM を使用する方法

IAM を使用して Amazon SQS へのアクセスを管理する前に、Amazon SQS で利用できる IAM 機能について理解しておく必要があります。

Amazon Simple Queue Service で使用できる IAM の機能

IAM の機能	Amazon SQS サポート
アイデンティティベースポリシー	はい
リソースベースのポリシー	あり
ポリシーアクション	はい
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	いいえ
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	あり
転送アクセスセッション (FAS)	あり
サービスロール	はい
サービスリンクロール	なし

Amazon SQS 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。

アクセスコントロール

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許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon S3、および Amazon VPC は AWS WAF、ACLs。ACL の詳細については、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) の概要」を参照してください。

> **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を使用すると、追加のユーザーを管理することなく、 AWS リソースへのアクセスを共有できます。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 SQS カスタムポリシー内のコンテンツ間のアクセス許可と条件キーの詳細については、「Amazon SQS カスタムポリシーの制限」を参照してください。

Amazon SQS 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 ユーザーグループ、ユーザーのグループ、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。ID ベースの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。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用でき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 SQS のアイデンティティベースのポリシー例

Amazon SQS のアイデンティティベースのポリシー例を確認するには、「ポリシーに関するベストプラクティス」を参照してください。

Amazon SQS 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: あり

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制御できます。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は、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、アカウント全体、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。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる場合 AWS アカウント、信頼されたアカウントの IAM 管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。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 SQS 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。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは依存アクションと呼ばれます。

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 SQS 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 Queue Service で定義されるリソース」を参照してください。

Amazon SQS 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。

```
sqs
```

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": [
      "sqs:action1",
      "sqs:action2"
         ]
```

Amazon SQS のアイデンティティベースのポリシー例を確認するには、「ポリシーに関するベストプラクティス」を参照してください。

Amazon SQS 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。ステートメントには`Resource` または `NotResource` 要素を含める必要があります。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。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限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 SQS の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 Queue 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、「Amazon Simple Queue Service で定義されるリソース」を参照してください。

Amazon SQS のアイデンティティベースのポリシー例を確認するには、「ポリシーに関するベストプラクティス」を参照してください。

Amazon SQS 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 (または `Condition` ブロック) を使用すると、ステートメントが有効な条件を指定できます。`Condition` 要素はオプションです。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。

1 つのステートメントに複数の `Condition` 要素を指定する場合、または 1 つの `Condition` 要素に複数のキーを指定する場合、 AWS では `AND` 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理`OR`オペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、「IAM ユーザーガイド」の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「グローバル条件コンテキストキー」を参照してください。

Amazon SQS の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 Queue 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、「Amazon Simple Queue Service で定義されるリソース」を参照してください。

Amazon SQS のアイデンティティベースのポリシー例を確認するには、「ポリシーに関するベストプラクティス」を参照してください。

Amazon SQS 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許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Amazon SQS 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制御 (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグと呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。その後、プリンシパルのタグがアクセスしようとしているリソースのタグと一致した場合にオペレーションを許可するように ABAC ポリシーをします。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

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 SQS 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

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。一時的な認証情報 AWS のサービス を使用する などの詳細については、IAM ユーザーガイドAWS のサービス の「IAM と連携する 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合は、一時的な認証情報を使用します。たとえば、会社のシングルサインオン (SSO) リンク AWS を使用してにアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、「IAM ユーザーガイド」の「ユーザーから IAM ロールに切り替える (コンソール)」を参照してください。

一時的な認証情報は、 AWS CLI または AWS API を使用して手動で作成できます。その後、これらの一時的な認証情報を使用して アクセスできます AWS。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 の一時的セキュリティ認証情報」を参照してください。

Amazon SQS の転送アクセスセッション

転送アクセスセッション (FAS) のサポート: あり

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を 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 と組み合わせて使用します。FAS リクエストは、サービスが他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。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 SQS 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 SQS の機能が破損する可能性があります。Amazon SQS が指示するときにのみ、サービスロールを編集してください。

Amazon SQS のサービスリンクロール

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。

AWS マネージドポリシーに対する Amazon SQS の更新

ユーザー、グループ、ロールにアクセス許可を追加するには自分でポリシーを作成するよりも、AWS マネージドポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには時間と専門知識が必要です。すぐに使用を開始するために、AWS マネージドポリシーを使用できます。これらのポリシーは一般的なユースケースを対象範囲に含めており、 AWS アカウントで利用できます。 AWS 管理ポリシーの詳細については、IAM ユーザーガイドの「 AWS 管理ポリシー」を参照してください。

AWS サービスは、 AWS 管理ポリシーを維持および更新します。 AWS 管理ポリシーのアクセス許可は変更できません。サービスは、新機能をサポートするために、 AWS マネージドポリシーに追加のアクセス許可を追加することがあります。この種類の更新はポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。サービスは、新機能が起動されたとき、または新しいオペレーションが利用可能になったときに、 AWS マネージドポリシーを更新する可能性が最も高いです。サービスは AWS マネージドポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が損なわれることはありません。

さらに、 は、複数のサービスにまたがるジョブ関数の マネージドポリシー AWS をサポートしています。例えば、ReadOnlyAccess AWS マネージドポリシーではすべての AWS のサービスおよびリソースへの読み取り専用アクセスを許可します。サービスが新機能を起動すると、 は新しいオペレーションとリソースの読み取り専用アクセス許可 AWS を追加します。ジョブ機能のポリシーの一覧および詳細については、「IAM ユーザーガイド」の「AWS のジョブ機能のマネージドポリシー」を参照してください。

AWS マネージドポリシー: AmazonSQSFullAccess

`AmazonSQSFullAccess` ポリシーは Amazon SQS ID に添付できます。このポリシーは、Amazon SQS への完全なアクセスを可能にする許可を付与します。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonSESReadOnlyAccess」を参照してください。

AWS マネージドポリシー: AmazonSQSReadOnlyAccess

`AmazonSQSReadOnlyAccess` ポリシーは Amazon SQS ID にアタッチできます。このポリシーは、Amazon SQS への読み取り専用アクセスを可能にする許可を付与します。

このポリシーのアクセス許可を確認するには、「AWS マネージドポリシーリファレンス」の「AmazonSQSReadOnlyAccess」を参照してください。

AWS マネージドポリシー: SQSUnlockQueuePolicy

Amazon SQS キューへのすべてのユーザーのアクセスを拒否するようにメンバーアカウントのキューポリシーを誤って設定した場合は、 `SQSUnlockQueuePolicy` AWS 管理ポリシーを使用してキューのロックを解除できます。

すべてのプリンシパルが Amazon SQS キューにアクセスすることを拒否する誤って設定されたキューポリシーを削除する方法の詳細については、IAM ユーザーガイドの AWS Organizations 「メンバーアカウントで特権タスクを実行する」を参照してください。

AWS マネージドポリシーに対する Amazon SQS の更新

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 SQS 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 SQS 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
SQSUnlockQueuePolicy	Amazon SQS は、 という新しい AWS管理ポリシーを追加してキューを`SQSUnlock QueuePolicy` ロック解除し、すべてのプリンシパルが Amazon SQS キューにアクセスすることを拒否する誤って設定されたキューポリシーを削除しました。	2024 年 11 月 15 日

変更	説明	日付
AmazonSQSReadOnlyAccess	Amazon SQS は、指定した Amazon SQS キューに関連付けられたすべてのタグを取得する `ListQueueTags` アクションを追加しました。これにより、組織またはメタデータの目的でキューに割り当てられたキーと値のペアを表示できます。このアクションは `ListQueueTags` API 演算に関連付けられています。	2024 年 6 月 20 日
AmazonSQSReadOnlyAccess	Amazon SQS には、特定のソースキューにある最新のメッセージ移動タスク (最大 10 個) を一覧表示できる新しいアクションが追加されました。このアクションは `ListMessageMoveTasks` API 演算に関連付けられています。	2023 年 6 月 9 日

Amazon Simple Queue 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 SQS 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 SQS 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 sqs:*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:
 sqs: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 sqs:*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 SQS 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、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

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 SQS でアクションを実行しようとする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
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 SQS リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまた

はアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- Amazon SQS 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 Simple Queue Service で IAM を使用する方法」を参照してください。
- 所有 AWS アカウント している 全体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方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

キューのロックを解除したい

が組織に AWS アカウント 属している場合、 AWS Organizations ポリシーによって Amazon SQS リソースへのアクセスがブロックされる可能性があります。デフォルトでは、 AWS Organizations ポリシーは Amazon SQS へのリクエストをブロックしません。ただし、Amazon SQS キューへのアクセスをブロックするように AWS Organizations ポリシーが設定されていないことを確認してください。 AWS Organizations ポリシーを確認する方法については、「 AWS Organizations ユーザーガイド」の「すべてのポリシーの一覧表示」を参照してください。

さらに、メンバーアカウントのキューポリシーを誤って設定してAmazon SQSキューへのすべてのユーザーのアクセスを拒否した場合、IAM のメンバーアカウントの特権セッションを起動してキューをロック解除できます。特権セッションを起動したら、誤って設定されたキューポリシーを削除して、キューへのアクセスを回復できます。詳細については、IAM ユーザーガイドの AWS Organizations 「メンバーアカウントで特権タスクを実行する」を参照してください。

Amazon SQS でのポリシーの使用

このトピックでは、アカウント管理者がIAMアイデンティティ (ユーザー、グループ、ロール) へのアクセス権限ポリシーをアタッチする、アイデンティティベースのポリシーの例を示します。

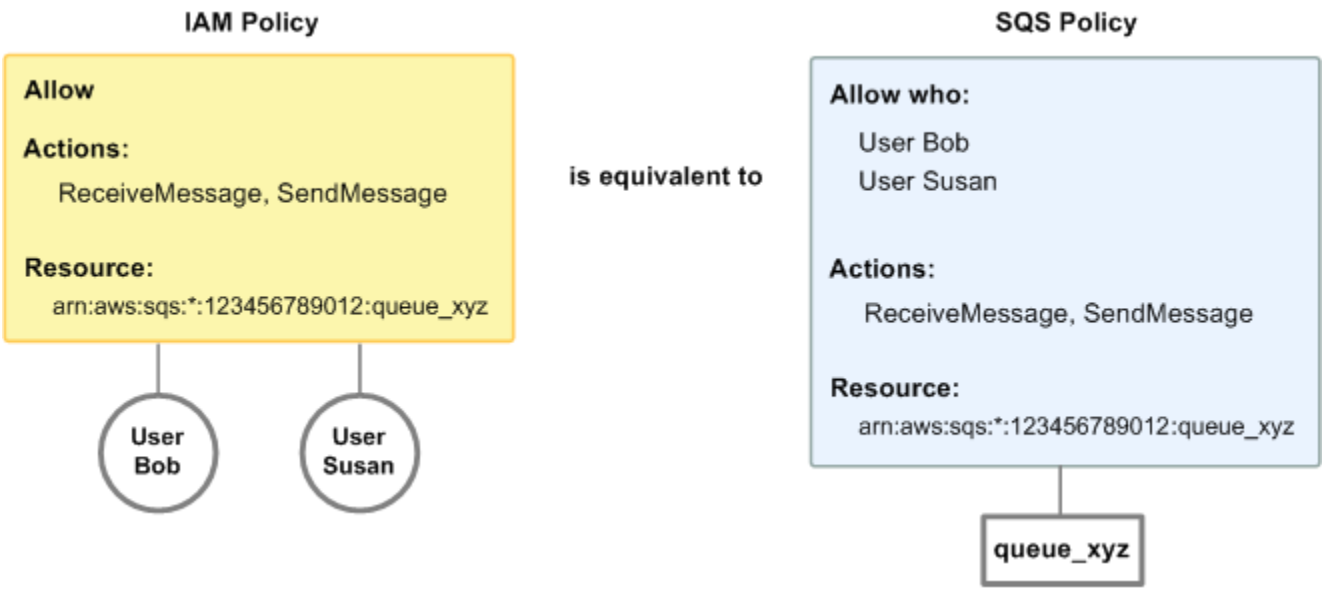
> **Important**
>
> 初めに Amazon Simpleキューサ - ビスリソースへのアクセスを管理するための基本概念と使用できるオプションについて説明する概要トピックをご覧になることをお勧めします。詳細については、「Amazon SQS でのアクセス管理の概要」を参照してください。
> ただし、`ListQueues`では、すべての Amazon SQSアクションがリソース-レベルのアクセス許可をサポートします。詳細については、「Amazon SQS (Amazon SQS)、 API のアクセス権限: アクションとリソースのリファレンスについて」を参照してください。

Amazon SQS ポリシーとIAMポリシーの使用

Amazon SQS リソースへのアクセス許可をユーザーに付与するには、Amazon SQS ポリシーシステム (リソースベースのポリシー) を使用する方法と IAMポリシーシステム (アイデンティティベースのポリシー) を使用する方法の 2 つがあります。 1 つまたは両方の方法を使用できます。ただし、`ListQueues` アクションについては、IAM ポリシーでのみ設定できるリージョンアクセス許可であるため、これを除きます。

例えば、以下の図は、IAMポリシーとそれに相当する Amazon SQS ポリシーを示しています。IAM ポリシーは、 AWS アカウントqueue_xyzで呼び出されたキューの Amazon SQS `ReceiveMessage`および `SendMessage` アクションに対する権限を付与し、ポリシーは Bob と Susan という名前のユーザーにアタッチされます (Bob と Susan にはポリシーに記載されているアクセス許可があります) 。この `ReceiveMessage`Amazon SQS ポリシーは、Bob と Susan に、同じキューの と `SendMessage` アクションへの権限も付与します。

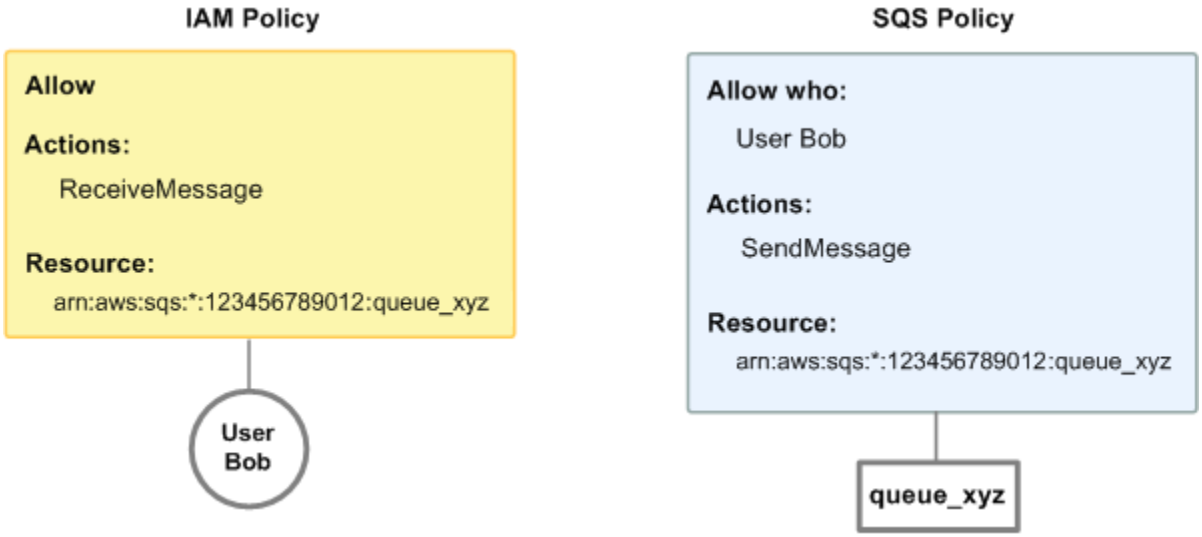
> **Note**
>
> この例は、条件のないシンプルなポリシーを示しています。どちらのポリシーでも特定の条件を指定して、同じ結果を得ることができます。



IAM ポリシーと Amazon SQS ポリシーには大きな違いが 1 つあります。Amazon SQS ポリシーシステムでは、他の AWS アカウントにアクセス許可を付与できますが、IAM では付与されません。

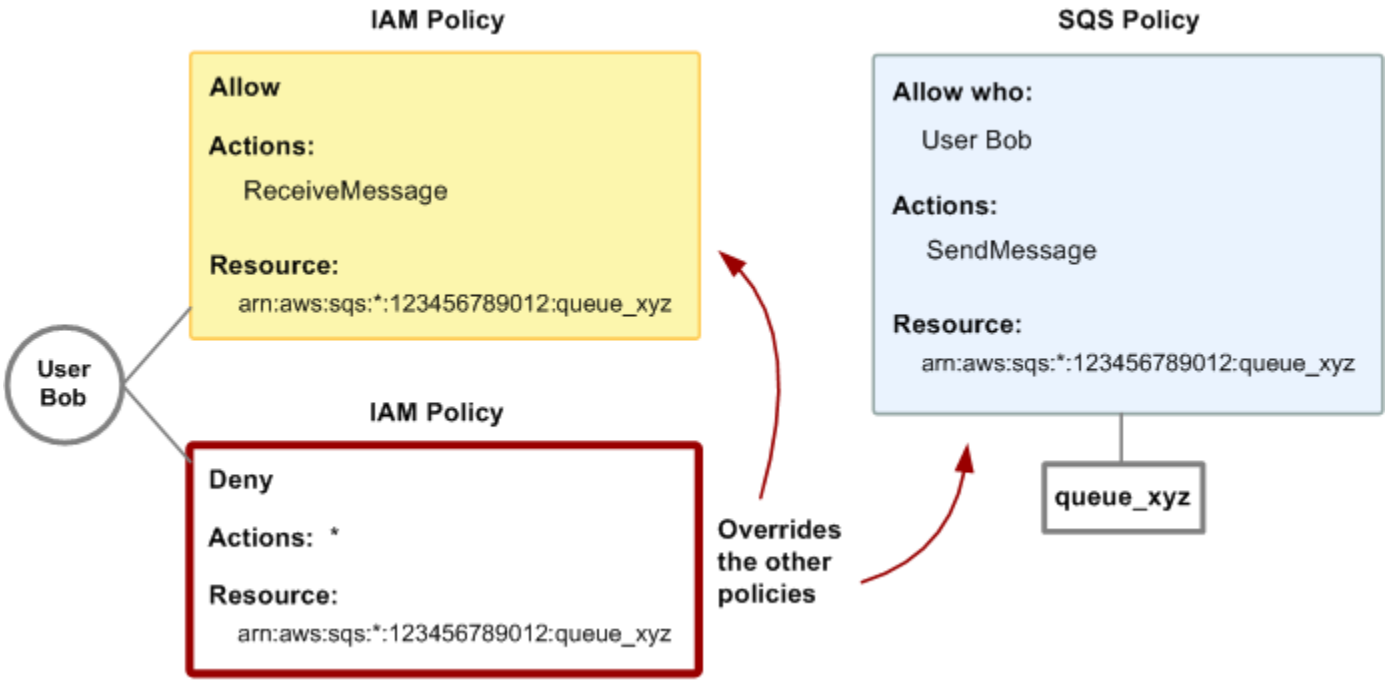
両方のシステムを同時に使用してどのようにアクセス許可を管理するかは、ご自身で決めてください。以下の例では、2つのポリシーシステムがどのように連携するかを示しています。

- 最初の例では、BobのアカウントにIAMポリシーとAmazon SQS ポリシーの両方が適用されています。`ReceiveMessage`IAMポリシーは、Amazon SQSでの`queue_xyz`アクションについて、Bob のアカウントにアクセス許可を付与しAmazon SQS 、 ポリシーは、同じキューの`SendMessage`アクションのアクセス許可をBobのアカウントに付与します。以下の図に、そのコンセプトを示します。



Bob がReceiveMessageリクエストをqueue_xyzに送信すると、IAMポリシーでそのアクションが許可されます。BobがSendMessageリクエストをqueue_xyz に送信すると Amazon SQSポリシーでそのアクションが許可されます。

- 2 番目の例では、Bob がqueue_xyzに対するアクセス許可を不正に使用したため、このキューへのBobのアクセス許可をすべて削除する必要があります。最も簡単な方法は、そのキューに対するBobのアクションをすべて拒否するようなポリシーを追加することです。明示的な denyは常にallowをオーバーライドするため、このポリシーは他の 2 つのポリシーをオーバーライドします。ポリシーの評価ロジックの詳細については、「Amazon SQSアクセスポリシー言語を使用したカスタムポリシーの使用」を参照してください。以下の図に、そのコンセプトを示します。



また、そのキューに対する Bob からのアクセスをすべて拒否するようなステートメントを Amazon SQSポリシーに追加することもできます。これは、そのキューに対する Bobのアクセスを拒否するIAMポリシーを追加するのと同じ効果を持ちます。Amazon SQS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、「ベーシックAmazon SQSポリシーの例」を参照してください。AmazonSQS記載 ポリシーの詳細については、「Amazon SQSアクセスポリシー言語を使用したカスタムポリシーの使用」を参照してください。

Amazon SQSコンソールの使用に必要なアクセス許可

Amazon SQSコンソールを使用して作業するユーザーには、ユーザーの AWS アカウントアカウントのAmazon SQSキューを使用するための最小限のアクセス権限が必要です。たとえば、ユーザーに

はキューをリスト表示するための `ListQueues` アクションを呼び出すアクセス権限、またはキューを作成するための `CreateQueue` アクションを呼び出すアクセス権限が必要です。Amazon SQS キューを Amazon SNSトピックに登録する Amazon SQS アクセス権限に加えて、コンソールには Amazon SNSアクション用のアクセス権限が必要です。

これらの最小限必要なアクセス権限よりも制限された IAMポリシーを作成している場合、そのIAMポリシーを使用するユーザーに対してコンソールは意図したとおりには機能しない場合があります。

AWS CLI または Amazon SQS アクション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。

Amazon SQS のアイデンティティベースのポリシー例

デフォルトでは、ユーザーとロールには Amazon SQS リソースを作成または変更するアクセス許可がありません。また、、 AWS Command Line Interface（AWS CLI）AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、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。その後、管理者はロールに IAM ポリシーを追加し、ユーザーはロールを引き継ぐことができます。

これらサンプルの 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 SQS 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 Queue 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。

> **Note**
>
> Amazon EC2 Auto Scaling のライフサイクルフックを構成する場合、メッセージをAmazon SQSキューに送信するためのポリシーを作成する必要はありません。詳細については、「Amazon EC2 ユーザーガイド」の「Amazon EC2 Auto Scaling のライフサイクルフック」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

アイデンティティベースのポリシーは、誰がユーザーのアカウントの Amazon SQS リソースを作成、アクセス、削除できるどうかを決定します。これらのアクションを実行すると、 AWS アカウン

トに料金が発生する可能性があります。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。例えば、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 AWS 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 Access Analyzer 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 Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、「IAM ユーザーガイド」の「IAM Access Analyzer でポリシーを検証する」を参照してください。
- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、MFA をオンにしてセキュリティを強化します。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、「IAM ユーザーガイド」の「MFA を使用した安全な API アクセス」を参照してください。

IAM でのベストプラクティスの詳細については、「IAM ユーザーガイド」の「IAM でのセキュリティのベストプラクティス」を参照してください。

Amazon SQS コンソールの使用

Amazon Simple Queue Service コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 の Amazon SQS 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 SQS 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 SQS `AmazonSQSReadOnlyAccess` 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。

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": "*"
        }
    ]
}
```

キューの作成をユーザーに許可する

次の例では、すべてのAmazon SQSアクションへのアクセスを Bobに許可するBob用のポリシーを作成しますが、名前のプレフィックスがリテラル文字列 `alice_queue_` になっているキューしか含まれていません。

Amazon SQSでは、キューの作成者に、そのキューを使用するアクセス権限が自動的に付与されません。したがって、IAM ポリシーの `CreateQueue` アクションに加えて、すべての Amazon SQS アクションを使用するアクセス権限を Bob に明示的に付与する必要があります。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
   }]
}
```

共有キューにメッセージを書き込むことを開発者に許可する

次の例では、開発者用のグループを作成し、グループが Amazon SQS `SendMessage`アクションを使用できるようにするポリシーをアタッチしますが、指定された に属 AWS アカウント し、 という名前のキューでのみ使用します`MyCompanyQueue`。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"
   }]
}
```

* の代わりにSendMessageを使用してアクション ChangeMessageVisibility、DeleteMessage、GetQueueAttributes、GetQueueUrl、ReceiveMe およびSendMessageを共有キューのプリンシパルに許可できます。

> **Note**
>
> *には、他のアクセス許可タイプにより提供されるアクセスが含まれています、Amazon SQS はアクセス許可を個別に考慮します。たとえば、* により許可されるアクセスがSendMessageに含まれている場合でも、*とSendMessageの両方のアクセス許可をユーザーに付与できます。
> このコンセプトは、アクセス許可を削除するときも適用されます。プリンシパルに*アクセス許可しかない場合は、SendMessageアクセス許可の削除をリクエストしても、プリンシパルは「everything-but」アクセス許可を持つことにはなりません。プリンシパルは明示的なSendMessageアクセス許可を持っていないため、リクエストに効果はありません。プリンシパルがReceiveMessageアクセス許可だけを持つようにする場合、まずReceiveMessage アクセス許可を追加してから*アクセス許可を削除します。

キューの一般的なサイズを取得することをマネージャーに許可する

次の例では、マネージャー用のグループを作成し、グループが指定された AWS アカウントに属するすべてのキューで Amazon SQS GetQueueAttributesアクションを使用できるようにするポリシーをアタッチします。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:GetQueueAttributes",
      "Resource": "*"
   }]
}
```

特定のキューにメッセージを送信することをパートナーに許可する

このタスクは、Amazon SQSポリシーまたはIAMポリシーを使用して完了できます。パートナーにがある場合は AWS アカウント、Amazon SQS 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のユーザーは、キューにメッセージを送信できます。特定のユーザーまたはアプリケーションにアクセスを制限する場合は、パートナーを自社内のユーザーと同様に扱い、IAMポリシーではなく Amazon SQSポリシーを使用する必要があります。

この例は以下のアクションを実行します。

1. パートナー企業を表すためにWidgetCoというグループを作成します。
2. アクセスを必要としているパートナー会社の特定のユーザーまたはアプリケーション用のユーザーを作成します。
3. ユーザーを グループに追加します。
4. `SendMessage`というキューのみの`WidgetPartnerQueue`アクションのみへのグループのアクセス権限を付与するポリシーを添付します。

JSON

```
{
   "Version": "2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "sqs:SendMessage",
         "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
```

```
   }]
}
```

ベーシックAmazon SQSポリシーの例

このセクションでは、一般的なAmazon SQSユースケースのサンプルポリシーを示します。

コンソールを使用して、ユーザーにポリシーをアタッチしながら各ポリシーの効果を検証できます。最初は、ユーザーにアクセス権限が付与されてないため、コンソールを使用して何もできません。ユーザーにポリシーをアタッチすることで、ユーザーがコンソールで多様なアクションを実行できることを確認できます。

> **Note**
>
> 2 つのブラウザウィンドウを使用することをお勧めします。1 つはアクセス許可を付与し、もう 1 つはユーザーの認証情報 AWS Management Console を使用して にサインインし、ユーザーに付与するアクセス許可を検証します。

例 1: 1 つのアクセス許可を 1 つに付与する AWS アカウント

次のポリシー例では`111122223333`、米国東部 (オハイオ) リージョン`444455556666/queue1`でという名前のキューの`SendMessage`アクセス許可を AWS アカウント 番号に付与します。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_SendMessage",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue1"
```

```
   }]
}
```

例 2: 1 つの に 2 つのアクセス許可を付与する AWS アカウント

次のポリシー例では、 という名前のキューの SendMessageと アクセスReceiveMessage許可111122223333の両方に AWS アカウント 番号を付与します444455556666/queue1。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_Send_Receive",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:*:444455556666:queue1"
   }]
}
```

例 3: 2 つの にすべてのアクセス許可を付与する AWS アカウント

次のポリシー例では、Amazon SQS が米国東部 (111122223333オハイオ444455556666) リージョン123456789012/queue1で という名前のキューの共有アクセスを許可するすべてのアクションを使用するための 2 つの異なる AWS アカウント 番号 (と) のアクセス許可を付与します。 Amazon SQS

JSON

```
{
```

```
    "Version": "2012-10-17",
    "Id": "Queue1_Policy_UUID",
    "Statement": [{
       "Sid":"Queue1_AllActions",
       "Effect": "Allow",
       "Principal": {
          "AWS": [
             "111122223333",
             "444455556666"
          ]
       },
       "Action": "sqs:*",
       "Resource": "arn:aws:sqs:us-east-2:123456789012:queue1"
    }]
}
```

例4: ロールおよびユーザー名にクロスアカウントのアクセス許可を付与する

次のポリシー例では`role1`、Amazon SQS が米国東部 (オハイオ) リージョン`123456789012/queue1`で という名前のキューの共有アクセスを許可するすべてのアクションを使用するための`111122223333`クロスアカウントアクセス許可を および `username1` AWS アカウント に付与します。

クロスアカウント権限は、次のアクションには適用されません。

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTask
- ListQueues
- ListQueueTags
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue

- UntagQueue

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AllActions",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "arn:aws:iam::111122223333:role/role1",
            "arn:aws:iam::111122223333:user/username1"
         ]
      },
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:queue1"
   }]
}
```

例5:すべてのユーザーにアクセス権限を付与する

以下のサンプルポリシーは、すべてのユーザー (匿名ユーザー) に、111122223333/queue1 という名前のキューに対する ReceiveMessage アクセス権限を付与します。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_ReceiveMessage",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:ReceiveMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1"
   }]
}
```

例6:すべてのユーザーに時間制限付きのアクセス権限を付与する

以下のサンプルポリシーは、すべてのユーザー (匿名ユーザー) に、`ReceiveMessage`という名前のキューに対する`111122223333/queue1`アクセス権限を 2009 年 1 月 31 日の午後 12:00 (正午) から午後 3:00 の間のみ付与します。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_ReceiveMessage_TimeLimit",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:ReceiveMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "DateGreaterThan" : {
            "aws:CurrentTime":"2009-01-31T12:00Z"
         },
         "DateLessThan" : {
            "aws:CurrentTime":"2009-01-31T15:00Z"
         }
      }
   }]
}
```

例7:CIDR範囲のすべてのユーザーにすべてのアクセス権限を付与する

以下のサンプルポリシーは、すべてのユーザー (匿名ユーザー) に、 `111122223333/queue1`という名前のキューで共有できるすべてのAmazon SQS アクションを使用するアクセス権限を、リクエストが`192.0.2.0/24` CIDRの範囲から生成された場合のみ付与します。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
```

```
      "Sid":"Queue1_AnonymousAccess_AllActions_AllowlistIP",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"192.0.2.0/24"
         }
      }
   }]
}
```

例8:異なる CIDR 範囲のユーザーの許可リストとブロックリストのアクセス権限

以下のサンプルポリシーには、2つのステートメントが含まれています。

- 最初のステートメントは、192.0.2.0/24CIDR の範囲 (192.0.2.188 を除く) に存在するすべてのユーザー (匿名ユーザー) に、SendMessage/queue1 という名前のキューに対する111122223333アクションを使用するアクセス権限を付与します。
- 2 番目のステートメントは、12.148.72.0/23CIDR の範囲に存在するすべてのユーザー (匿名ユーザー) のキュー使用をブロックます。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_SendMessage_IPLimit",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"192.0.2.0/24"
         },
         "NotIpAddress" : {
            "aws:SourceIp":"192.0.2.188/32"
```

```
            }
         }
      }, {
         "Sid":"Queue1_AnonymousAccess_AllActions_IPLimit_Deny",
         "Effect": "Deny",
         "Principal": "*",
         "Action": "sqs:*",
         "Resource": "arn:aws:sqs:*:111122223333:queue1",
         "Condition" : {
            "IpAddress" : {
               "aws:SourceIp":"12.148.72.0/23"
            }
         }
      }]
}
```

Amazon SQSアクセスポリシー言語を使用したカスタムポリシーの使用

AWS アカウント ID のみに基づいて基本的なアクセス許可 (SendMessageや などReceiveMessage) を付与するには、カスタムポリシーを記述する必要はありません。代わりに、Amazon SQS AddPermissionアクションを使用します。

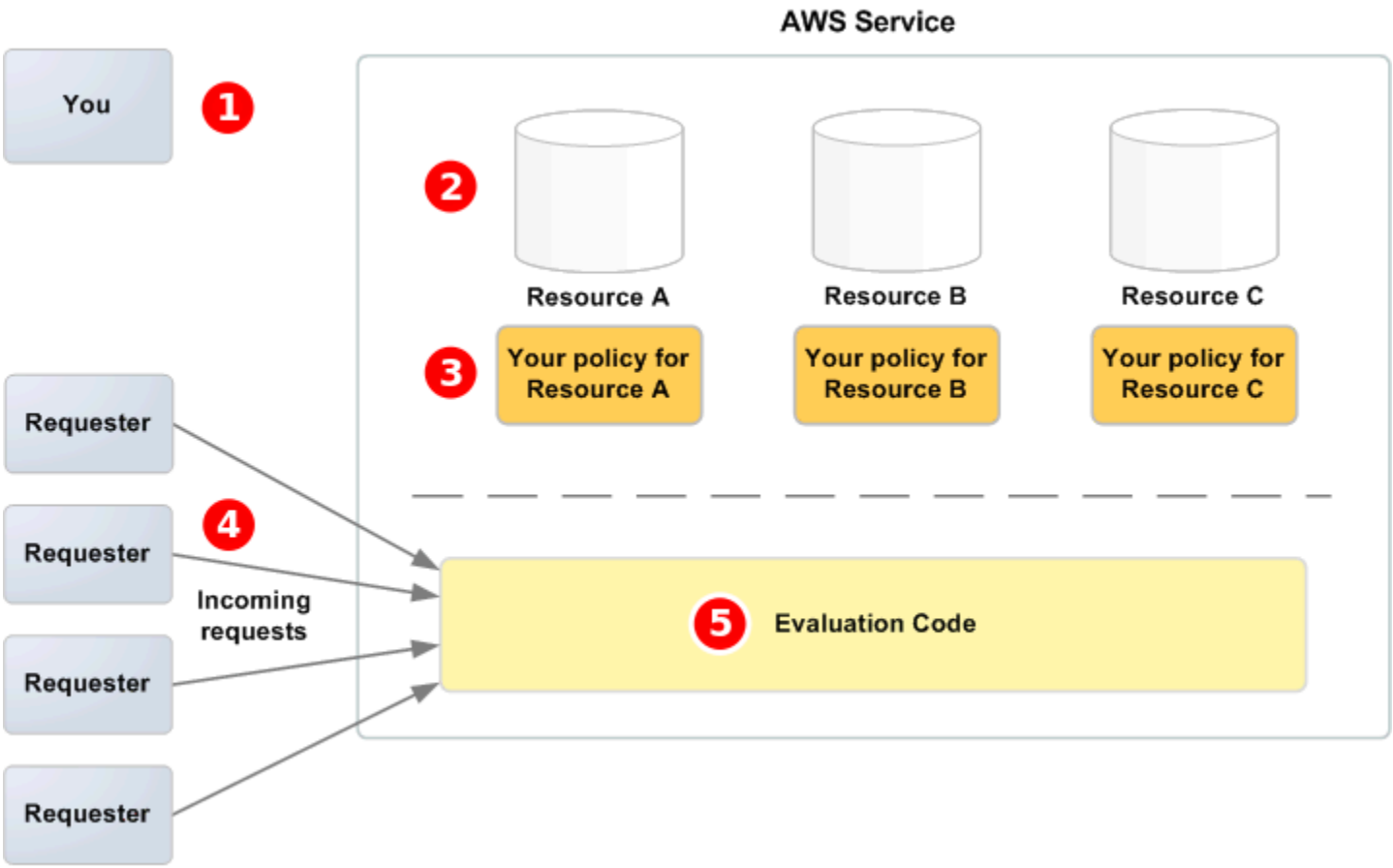
リクエスト時間やリクエスタの IP アドレスなどの特定の条件に基づいてアクセスを許可または拒否するには、カスタム Amazon SQS ポリシーを作成し、SetQueueAttributes アクションを使用してアップロードする必要があります。

トピック

- Amazon SQSのアクセスコントロールアーキテクチャ
- Amazon SQSアクセスコントロールプロセスのワークフロー
- Amazon SQSアクセスポリシー言語の主要な概念
- Amazon SQSアクセスポリシー言語評価ロジック
- Amazon SQSアクセスポリシー言語の明示的な拒否とデフォルトの拒否の関係
- Amazon SQS カスタムポリシーの制限
- カスタムの Amazon SQSアクセスポリシー言語の例

Amazon SQSのアクセスコントロールアーキテクチャ

以下の図は、Amazon SQSリソースのアクセスコントロールの説明です。



1 リソース所有者。

2 AWS サービスに含まれるリソース (Amazon SQS キューなど) 。

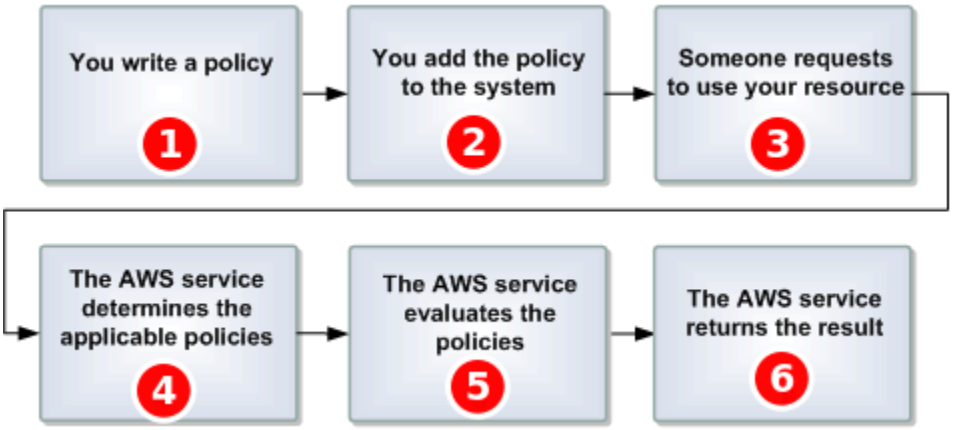
3 ポリシー。1 つのリソースごとに 1 つのポリシーを適用することをお勧めします。この AWS サービスには、ポリシーのアップロードと管理に使用する API が用意されています。

4 リクエスタ、および AWS サービスに対する受信リクエスト。

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。

Amazon SQSアクセスコントロールプロセスのワークフロー

以下の図は、 Amazon SQSアクセスポリシー言語と アクセスコントロールの全般的なワークフローを表しています。



1 キューの Amazon SQS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。この AWS サービスには、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。たとえば、特定の Amazon SQSキュー用のポリシーをアップロードするために Amazon SQS`SetQueueAttributes`アクションを使用します。

3 ある人物から、Amazon SQSキューの使用を求めるリクエストが送信されます。

4 Amazon SQS はすべての利用可能な Amazon SQSポリシーを分析し、該当するものを判断します。

5 Amazon SQSがポリシーを評価し、キューの使用許可をリクエスタに付与するかどうかを決定します。

6 ポリシーの評価結果に基づいて、Amazon SQSはリクエスタに `Access denied` エラーを返すか、リクエストの処理を続行します。

Amazon SQSアクセスポリシー言語の主要な概念

独自のポリシーを作成するには、JSON およびいくつかの重要な概念を理解している必要があります。

許可

Statementが[Effect] (効果)に設定されたallowの結果。

アクション

プリンシパルが実行アクセス権限を持っているアクティビティ。通常は AWSへのリクエスト。

Default-deny

Statementまたは許可設定を持たないExplicit-denyの結果。

条件

アクセス権限に関する制限または詳細。よく使用される条件は日時とIP アドレスに関連しています。

[Effect] (効果)

StatementのPolicyで評価時に返される結果。ポリシーステートメントを作成するときに、denyまたはallowの値を指定します。ポリシーの評価時に、Default-deny、許可、またはExplicit-denyの 3 つの結果が得られます。

Explicit-deny

Statementが[Effect] (効果)に設定されたdenyの結果。

評価

Amazon SQSが使用するプロセスでは、受信したリクエストを拒否または許可するかを、Policyに基づいて判断します。

Issuer

ユーザーはPolicyを記述して、リソースへのアクセス権限を付与します。定義上、発行者は常にリソース所有者です。Amazon SQS AWS ユーザーは、所有していないリソースのポリシーを作成できません。

キー

アクセス制限に使用される基本項目です。

アクセス権限

条件およびキー を使用して、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。

Policy

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です。

Amazon SQSはポリシーを使用して、リソースに関してユーザーにアクセス権限を付与するかどうかを決定します。

プリンシパル

アクセス権限でPolicyを受け取るユーザー。

[リソース]

プリンシパル によるリクエストがアクセスするオブジェクト。

Statement

より広範なドキュメントの一部としてPolicyで作成された1つのアクセス権限の正式な説明。

リクエスタ

[リソース] にアクセスするためにリクエストを送信するユーザー。

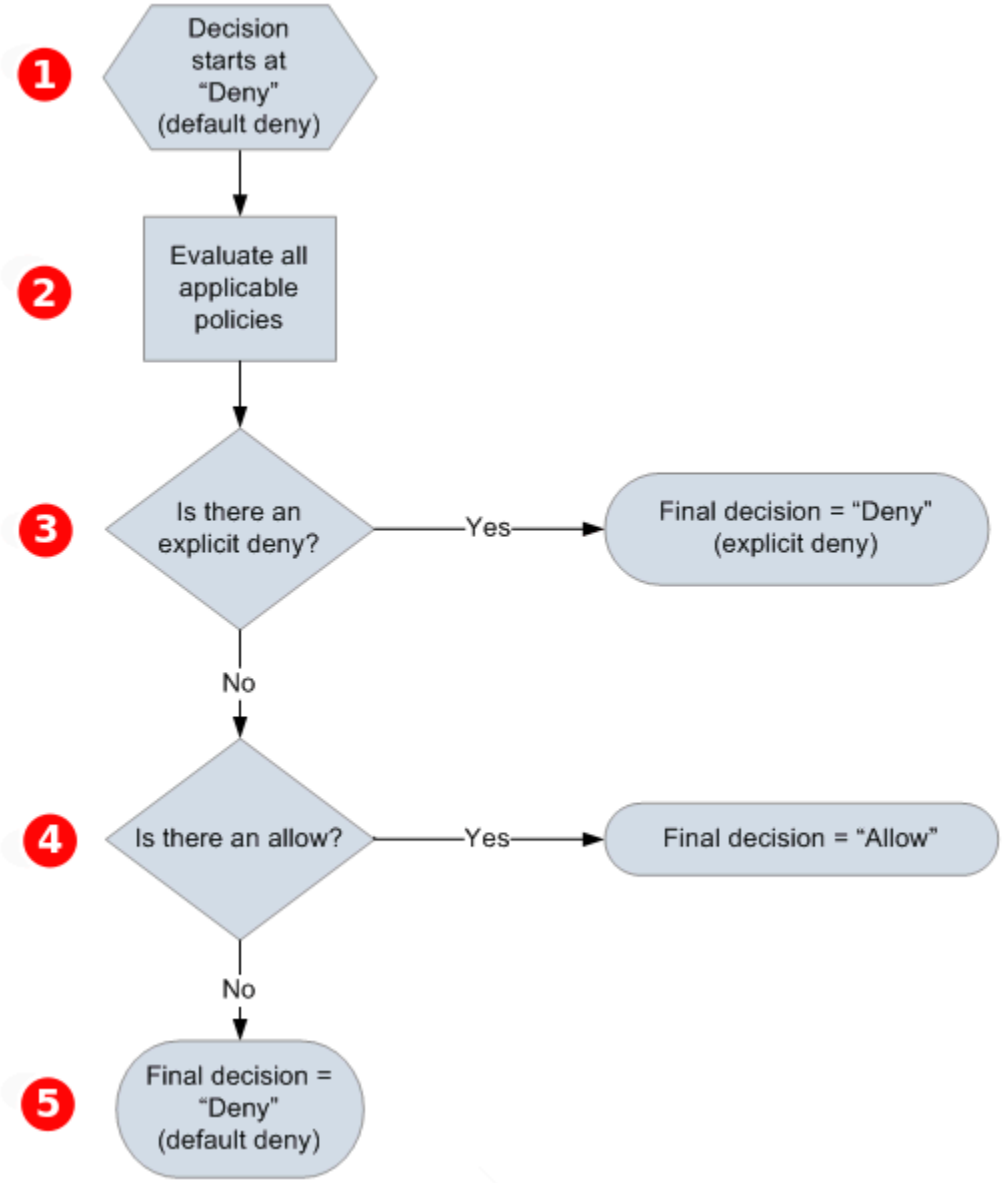
Amazon SQSアクセスポリシー言語評価ロジック

Amazon SQSは評価時に、リソース所有者以外の別の人物から与えられたリクエストに対し、許可するか拒否するかを決定します。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、リクエスタ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Default-denyをオーバーライドする。

- Explicit-deny はすべてのallowをオーバーライドする。
- ポリシー評価の順序は重要ではない。

次の図は、アクセス権限に関する決定事項をAmazon SQSがどのように評価するかに関する詳細を示しています。



1

決定はdefault-denyから始まります。

2

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

エンフォースメントコードは、リクエストに適用できる explicit-deny インストラクションを探します。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、プロセスは終了します。

4

explicit-deny が見つからなかった場合、リクエストに適応できるallow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)。

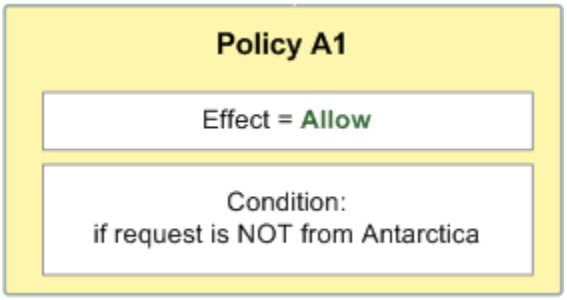
5

allowインストラクションが見つからなかった場合、最終決定は denyとなります(explicit-denyまたはallowが見つからない場合、default-denyとして見なされるためです)。

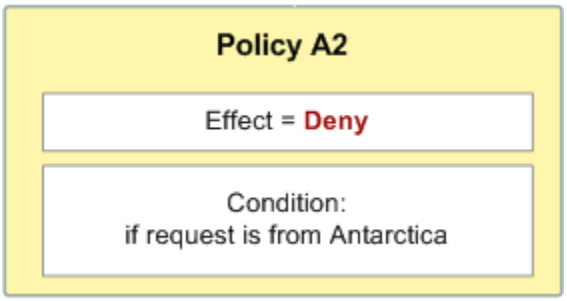
Amazon SQSアクセスポリシー言語の明示的な拒否とデフォルトの拒否の関係

Amazon SQS ポリシーがリクエストに直接適用されない場合、リクエストの結果は、Default-denyとなります。たとえば、ユーザーが Amazon SQSを使用するアクセス権限をリクエストしたが、そのユーザーに適用される唯一のポリシーではDynamoDBを使用できる場合、リクエストの結果はdefault-denyとなります。

ステートメントの条件が満たされない場合、リクエストの結果は default-denyになります。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 [Effect] (効果) エレメントの値に基づいて、リクエストの結果は 許可 または Explicit-deny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 default-deny となります。たとえば、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 A1を記述します。以下の図はAmazon SQSポリシーについて解説しています。

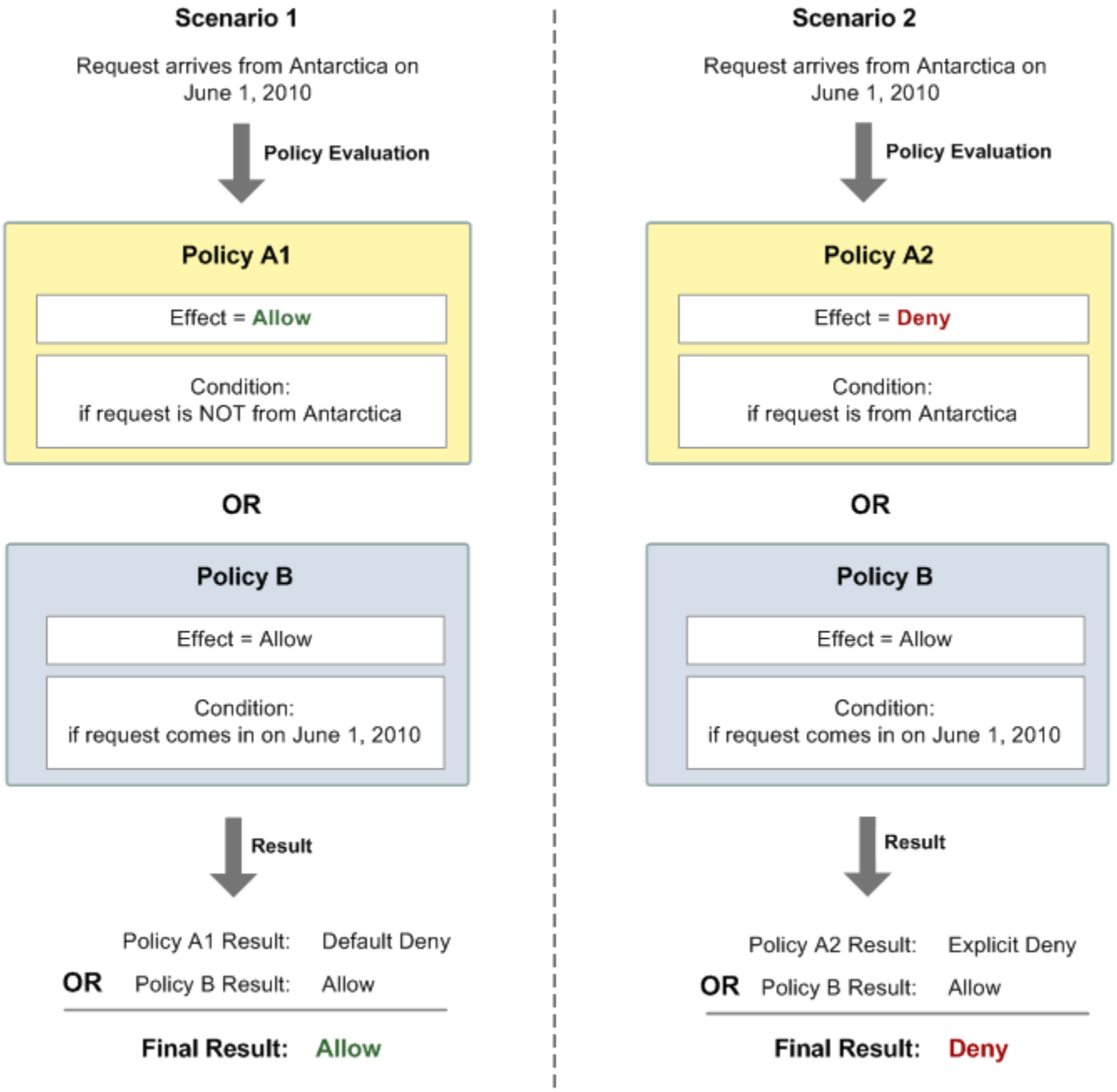


ユーザーがアメリカからリクエストを送信すると、条件が満たされ (リクエストは南極大陸から来ていない)、リクエストの結果は allow になります。ただし、ユーザーが南極からリクエストを送信した場合、条件は満たされず、リクエストはデフォルトで default-deny になります。南極大陸から来たリクエストを明示的に拒否するポリシー A2を作成して、結果を explicit-deny に変更することができます。以下の図はポリシーについて解説しています。



ユーザーが南極大陸からリクエストを送信すると、条件は満たされ、リクエストの結果はexplicit-denyとなります。

default-deny と explicit-deny の違いは重要です。allow は前者を上書きできますが、後者を上書きすることはできないためです。たとえば、ポリシー Bは、2010年6月1日に届いたリクエストを許可します。以下の図は、このポリシーをポリシー A1およびポリシー A2と組み合わせた場合を比較しています。



シナリオ 1では、ポリシー A1の結果はdefault-denyになり、ポリシー Bの結果はallowになります。これは、ポリシーで 2010年6月1日に来るリクエストが許可されるためです。ポリシー Bによるallowは、ポリシー A1の default-deny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。

シナリオ 2 で、ポリシー B2の結果は explicit-deny になり、ポリシー Bの結果は allow になります。ポリシー A2 によるexplicit-denyは、ポリシー Bのallow 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 SQS カスタムポリシーの制限

クロスアカウントアクセス

クロスアカウント権限は、次のアクションには適用されません。

- AddPermission
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTask
- ListQueues
- ListQueueTags
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

条件キー

現在、Amazon SQS はIAMで使用可能な条件キーの制限されたサブセットのみをサポートしています。詳細については、「Amazon SQS (Amazon SQS)、 API のアクセス権限: アクションとリソースのリファレンスについて」を参照してください。

カスタムの Amazon SQSアクセスポリシー言語の例

一般的な Amazon SQSアクセスポリシーの例を次に示します。

例1:1つのアカウントにアクセス権限を与える

次のAmazon SQSポリシー例では、 AWS アカウント 444455556666で所有されるqueue2から送受信するアクセス権限を AWS アカウント 111122223333に与えます。

JSON

```
{
```

```
   "Version": "2012-10-17",
   "Id": "UseCase1",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2"
   }]
}
```

例2:1つ以上のアカウントにアクセス権限を与える

次の Amazon SQS ポリシーの例では、特定の期間にアカウントが所有するキューへの 1 つ以上の AWS アカウント アクセスを許可します。このポリシーを作成し、Amazon SQSにアップロードするには、SetQueueAttributesのためアクションAddPermissionアクションでは、キューへのアクセスを許可するときに時間制限を指定することはできません。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase2",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333",
            "444455556666"
         ]
      },
      "Action": [
         "sqs:SendMessage",
```

```
            "sqs:ReceiveMessage"
         ],
         "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
         "Condition": {
            "DateLessThan": {
               "AWS:CurrentTime": "2009-06-30T12:00Z"
            }
         }
      }]
}
```

例3:Amazon EC2インスタンスからのリクエストに許可を与える

次の例では、Amazon SQSポリシーはAmazon EC2 インスタンスから来るリクエストへのアクセスを許可します。この例では、"例2:1つ以上のアカウントにアクセス権限を与える" の例を作成します。2009年6月30日正午 (UTC) 以前のアクセスを制限し、IP 範囲`203.0.113.0/24`へのアクセスを制限します。このポリシーを作成し、Amazon SQS にアップロードするには、`SetQueueAttributes`のためにアクション`AddPermission`アクションでは、キューへのアクセスを許可するときに IP アドレス制限を指定することはできません。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase3",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2",
      "Condition": {
         "DateLessThan": {
            "AWS:CurrentTime": "2009-06-30T12:00Z"
```

```
            },
            "IpAddress": {
               "AWS:SourceIp": "203.0.113.0/24"
            }
         }
      }]
}
```

例4:特定のアカウントへのアクセスを拒否する

次の Amazon SQS ポリシーの例では、キューへの特定の AWS アカウント アクセスを拒否します。この例では、例1:1つのアカウントにアクセス権限を与える「」の例に基づいて構築されています。指定された へのアクセスを拒否します AWS アカウント。このポリシーを作成し、Amazon SQS にアップロードするには、SetQueueAttributesのためにアクションAddPermissionアクションは、キューへのアクセスを拒否することを許可しません (キューへのアクセスのみを許可します) 。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase4",
   "Statement" : [{
      "Sid": "1",
      "Effect": "Deny",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-2:444455556666:queue2"
   }]
}
```

例5:VPC エンドポイントからではない場合はアクセスを拒否する

次のqueue1111122223333Amazon SQSポリシー例では、SendMessage へのアクセスを制限します。ReceiveMessage が vpce-1a2b3c4d および アクションを実行できるのは、VPC エンドポイント ID (aws:sourceVpce 条件を使用して指定) からのみです。詳細については、「Amazon SQS のAmazon Virtual Private Cloud エンドポイント」を参照してください。

> **Note**
>
> - aws:sourceVpce 条件では、VPC エンドポイントID のみが必要で、VPC エンドポイントリソースのARNは必要ありません。
> - 次の例を変更し、2番目のステートメントですべてのアクション (sqs:*) を拒否して、特定の VPC エンドポイントにすべての Amazon SQS アクションを制限できます。ただし、このようなポリシーステートメントにより、すべてのアクション (キューのアクセス権限を変更するために必要な管理アクションを含む) が、ポリシーで定義された特定の VPC エンドポイントを通じて行われる必要があることが規定されます。この場合、ユーザーは今後キューのアクセス権限を変更できなくなる可能性があります。

JSON

```
{
   "Version": "2012-10-17",
   "Id": "UseCase5",
   "Statement": [{
      "Sid": "1",
      "Effect": "Allow",
      "Principal": {
         "AWS": [
            "111122223333"
         ]
      },
      "Action": [
         "sqs:SendMessage",
         "sqs:ReceiveMessage"
      ],
         "Resource": "arn:aws:sqs:us-east-2:111122223333:queue1"
      },
      {
```

```
            "Sid": "2",
            "Effect": "Deny",
            "Principal": "*",
            "Action": [
               "sqs:SendMessage",
               "sqs:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:us-east-2:111122223333:queue1",
            "Condition": {
               "StringNotEquals": {
                  "aws:sourceVpce": "vpce-1a2b3c4d"
               }
            }
         }
      ]
   }
```

Amazon SQSで一時的なセキュリティ認証情報を使用する

IAM では、独自のセキュリティ認証情報を使用してユーザーを作成するだけでなく、任意のユーザーに一時的なセキュリティ認証情報を付与できるため、ユーザーは AWS サービスとリソースにアクセスできます。 AWS アカウントを持つユーザーを管理できます。また、 を持たないシステムのユーザー AWS アカウント（フェデレーティッドユーザー) を管理することもできます。さらに、AWS リソースにアクセスするために作成したアプリケーションは、「ユーザー」とみなすこともできます。

Amazon SQS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。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。失効した証明書を使用してリクエストを送信した場合、Amazon SQS はリクエストを拒否します。

> **Note**
>
> 一時的な認証情報に基づいてポリシーを設定することはできません。

前提条件

1. 一時的なセキュリティ認証情報を作成するには、IAMを使用します。
   - セキュリティトークン

- アクセスキー ID
- シークレットアクセスキー

2. 一時アクセスキー IDとセキュリティトークンで署名対象のリクエスト文字列を準備します。
3. 独自のシークレットアクセスキーの代わりに一時シークレットアクセスキーを使用して、クエリ API リクエストに署名します。

> **Note**
>
> 署名付きのクエリAPI リクエストを送信するときは、独自のアクセスキー ID の代わりに一時アクセスキー IDを使用して、セキュリティトークンを含めます。一時的なセキュリティ認証情報の IAM サポートの詳細については、IAM ユーザーガイドのAWS 「リソースへの一時的なアクセスの付与」を参照してください。

一時的なセキュリティ認証情報を使用して Amazon SQSクエリ API アクションを呼び出すには

1. を使用して一時的なセキュリティトークンをリクエストします AWS Identity and Access Management。詳細については、IAM ユーザーガイドの「一時的なセキュリティ認証情報を使用して、IAMユーザーのアクセスを可能にする」 を参照してください。

   IAMにより、セキュリティトークン、アクセスキー ID、シークレットアクセスキーが返信されます。
2. クエリは、独自のアクセスキー IDの代わりに一時アクセスキー IDを使用し、セキュリティトークンを含めて準備します。独自のシークレットアクセスキーの代わりに一時シークレットアクセスキーを使用してリクエストに署名します。
3. 一時アクセスキー IDとセキュリティトークンを含む署名付きクエリ文字列を送信します。

   以下の例は、一時的なセキュリティ認証情報を使用してAmazon SQS リクエストを認証する方法を示しています。*AUTHPARAMS*の構造はAPIリクエストの署名によって異なります。詳細については、Amazon Web Services 全般のリファレンスの AWS API リクエストの署名を参照してください。

```
https://sqs.us-east-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Attribute.1.Name=VisibilityTimeout
```

```
&Attribute.1.Value=40
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

以下の例は、一時的なセキュリティ認証情報を使用し、SendMessageBatch アクションで 2 つのメッセージを送信する方法を示しています。

```
https://sqs.us-east-2.amazonaws.com/
?Action=SendMessageBatch
&SendMessageBatchRequestEntry.1.Id=test_msg_001
&SendMessageBatchRequestEntry.1.MessageBody=test%20message%20body%201
&SendMessageBatchRequestEntry.2.Id=test_msg_002
&SendMessageBatchRequestEntry.2.MessageBody=test%20message%20body%202
&SendMessageBatchRequestEntry.2.DelaySeconds=60
&Expires=2020-12-18T22%3A52%3A43PST
&SecurityToken=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
&AWSAccessKeyId=AKIAI44QH8DHBEXAMPLE
&Version=2012-11-05
&AUTHPARAMS
```

最小特権ポリシーによる暗号化された Amazon SQS キューのアクセス管理

Amazon SQS では、AWS Key Management Service (KMS) と統合されたサーバー側の暗号化 (SSE) を使用して、アプリケーション間で機密データを交換できます。Amazon SQS と の統合により AWS KMS、Amazon SQS を保護するキーと、他の AWS リソースを保護するキーを一元管理できます。

複数の AWS サービスが Amazon SQS にイベントを送信するイベントソースとして機能します。イベントソースが暗号化された Amazon SQS キューにアクセスできるようにするには、カスタマーマネージド AWS KMS キーを使用してキューを設定する必要があります。次に、 キーポリシーを使用して、サービスに必要な AWS KMS API メソッドの使用を許可します。サービスには、キューがイベントを送信できるようにするためのアクセス認証のアクセス権限も必要です。これを実現するには、Amazon SQS ポリシーを使用します。Amazon SQS ポリシーは、Amazon SQS キューとそのデータへのアクセスを制御するために使用できるリソースベースのポリシーです。

以下のセクションでは、Amazon SQS ポリシーと AWS KMS キーポリシーを使用して、暗号化された Amazon SQS キューへのアクセスを制御する方法について説明します。このガイドのポリシーは、最小特権を達成するのに役立ちます。

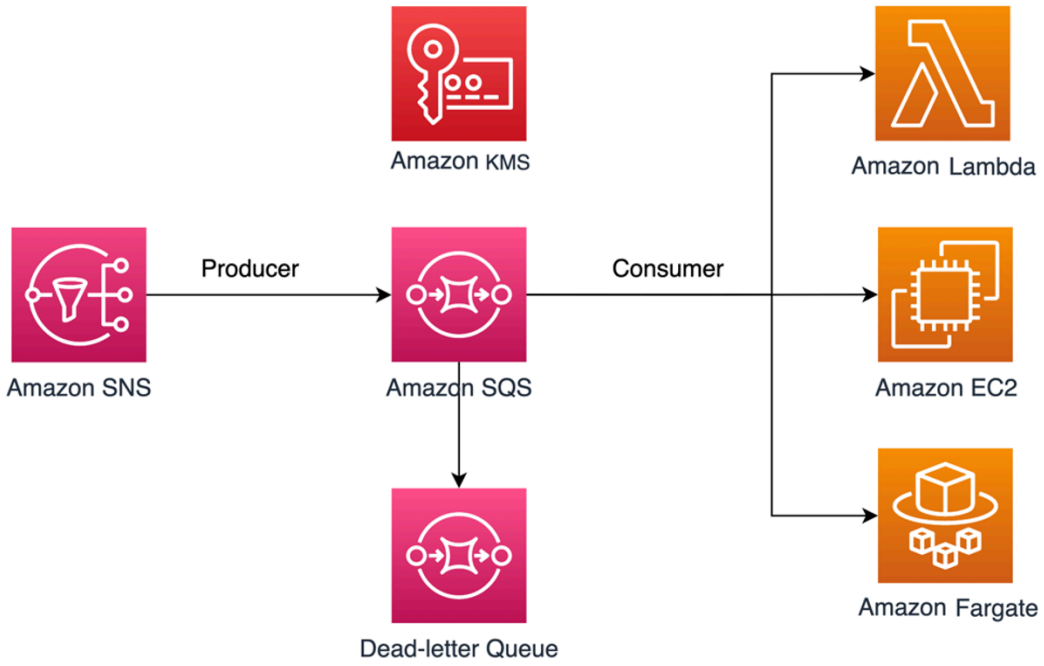
また、このガイドでは、`aws:SourceArn`、`aws:SourceAccount`、および `aws:PrincipalOrgID` グローバル IAM 条件コンテキストキーを使用して、リソースベースのポリシーで混乱した使節の問題に対処する方法についても説明します。

トピック

- 概要
- Amazon SQS の最小特権キーポリシー
- デッドレターキューの Amazon SQS ポリシーステートメント
- サービス間での混乱した使節問題を防止する
- IAM Access Analyzer を使用して、クロスアカウントアクセスを確認します。

概要

このトピックでは、一般的なユースケースを順を追って説明し、キーポリシーと Amazon SQS キューポリシーを構築する方法について説明します。このユースケースは次の画像に示されています。



この例では、メッセージプロデューサーは Amazon Simple Notification Service (SNS) トピックで、暗号化された Amazon SQS キューにメッセージをファンアウトするように設定されています。メッセージコンシューマーは、AWS Lambda 関数、Amazon Elastic Compute Cloud (EC2) インスタンス、AWS Fargate コンテナなどのコンピューティングサービスです。Amazon SQS キューは、失敗したメッセージをデッドレターキュー (DLQ) に送信するように設定されます。DLQ は、未使用のメッセージを分離して、処理が成功しない理由を調べることができるため、アプリケーションやメッセージングシステムのデバッグに役立ちます。このトピックで定義されているソリューションでは、Lambda 関数などのコンピューティングサービスを使用して Amazon SQS キューに保存されたメッセージを処理します。メッセージコンシューマーが仮想プライベートクラウド (VPC) に配置されている場合、このガイドに含まれる `DenyReceivingIfNotThroughVPCE` ポリシーステートメントにより、メッセージの受信をその特定の VPC に制限できます。

> **Note**
>
> このガイドには、必要な IAM アクセス許可のみがポリシーステートメントの形式で記載されています。ポリシーを作成するには、Amazon SQS ポリシーまたは AWS KMS キーポリシーにステートメントを追加する必要があります。このガイドでは、Amazon SQS キューまたは AWS KMS キーを作成する方法については説明していません。これらのリソースの作成方法については、「Amazon SQS キューを作成する」および「キーの作成」を参照してください。
>
> このガイドで定義されている Amazon SQS ポリシーでは、メッセージを同じ Amazon SQS キューまたは別の Amazon SQS キューに直接リドライブすることはサポートされません。

Amazon SQS の最小特権キーポリシー

このセクションでは、Amazon SQS キューの暗号化に使用するカスタマーマネージドキー AWS KMS の で必要な最小特権のアクセス許可について説明します。これらのアクセス許可により、最小特権を実装しながら、アクセスを目的のエンティティのみに制限できます。キーポリシーは、以下のポリシーステートメントで構成されている必要があります。詳細については以下で説明します。

- AWS KMS キーに管理者権限を付与する
- キーメタデータへの読み取り専用アクセスを付与する
- キューにメッセージを発行するために、Amazon SNS KMS のアクセス許可を Amazon SNS に付与する
- コンシューマーがキューからのメッセージを復号化することを許可する

AWS KMS キーに管理者権限を付与する

AWS KMS キーを作成するには、 AWS KMS キーのデプロイに使用する IAM ロールに AWS KMS 管理者権限を付与する必要があります。これらの管理者権限は、以下の `AllowKeyAdminPermissions` ポリシーステートメントで定義されています。このステートメントを AWS KMS キーポリシーに追加するときは、*<admin-role ARN>* を、 AWS KMS キーのデプロイ、 AWS KMS キーの管理、またはその両方に使用される IAM ロールの Amazon リソースネーム (ARN) に置き換えてください。これは、デプロイパイプラインの IAM ロールでも、AWS Organizations 内の組織の管理者ロールでもかまいません。

```
{
  "Sid": "AllowKeyAdminPermissions",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<admin-role ARN>"
    ]
  },
  "Action": [
    "kms:Create*",
    "kms:Describe*",
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:TagResource",
    "kms:UntagResource",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

> **Note**
>
> AWS KMS キーポリシーでは、 `Resource`要素の値は である必要があります。これは`*`「この AWS KMS キー」を意味します。アスタリスク (*) は、 AWS KMS キーポリシーがアタッチされているキーを識別します。

キーメタデータへの読み取り専用アクセスを付与する

他の IAM ロールにキーメタデータへの読み取り専用アクセス権を付与するには、その `AllowReadAccessToKeyMetaData` ステートメントをキーポリシーに追加します。たとえば、次のステートメントでは、監査目的でアカウント内のすべての AWS KMS キーを一覧表示できます。このステートメントは、 AWS ルートユーザーにキーメタデータへの読み取り専用アクセスを許可します。したがって、アカウント内の IAM プリンシパルは、その ID ベースのポリシーに次のステートメント (`kms:Describe*`、`kms:Get*`、`kms:List*`) に記載されているアクセス権限が付与されていれば、キーメタデータにアクセスできます。*`<account-ID>`* をユーザー自身の情報に必ず置き換えます。

```
{
  "Sid": "AllowReadAcesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
  },
  "Action": [
    "kms:Describe*",
    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
}
```

キューにメッセージを発行するために、Amazon SNS KMS のアクセス許可を Amazon SNS に付与する

Amazon SNS トピックが、暗号化された Amazon SQS キューにメッセージを発行できるようにするには、`AllowSNSToSendToSQS` ポリシーステートメントをキーポリシーに追加します。このステー

トメントは、 AWS KMS キーを使用して Amazon SNSAmazon SQSに付与します。*<account-ID>* をユーザー自身の情報に必ず置き換えます。

> **Note**
>
> ステートメントConditionの は、同じ AWS アカウントの Amazon SNS サービスのみへのアクセスを制限します。

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
  }
}
```

コンシューマーがキューからのメッセージを復号化することを許可する

次の `AllowConsumersToReceiveFromTheQueue` ステートメントは、暗号化された Amazon SQS キューから受信したメッセージを復号化するために必要なアクセス許可を Amazon SQS メッセージコンシューマーに付与します。ポリシーステートメントを添付するときは、*<consumer's runtime role ARN>* をメッセージコンシューマーの IAM ランタイムロール ARN に置き換えます。

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
```

```
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

Amazon SQS の最小特権ポリシー

このセクションでは、このガイドの対象となるユースケース (例えば Amazon SNS から Amazon SQS へ) の最小特権の Amazon SQS キューポリシーについて説明します。定義されたポリシーは、`Deny` および `Allow` ステートメントの両方を組み合わせて使用することで、意図しないアクセスを防ぐように設計されています。`Allow` ステートメントは、目的の 1 つ以上のエンティティへのアクセスを許可します。`Deny` ステートメントは、ポリシー条件の範囲内で目的のエンティティを除外しながら、意図しない他のエンティティが Amazon SQS キューにアクセスするのを防ぎます。

Amazon SQS ポリシーには以下のステートメントが含まれています。詳細については以下で説明します。

- Amazon SQS の管理権限を制限する
- 指定された組織からの Amazon SQS キューアクションを制限する
- Amazon SQS のアクセス許可をコンシューマーに付与する
- 転送時の暗号化を強制する
- メッセージ送信を特定の Amazon SNS トピックに制限する
- (オプション) 特定の VPC エンドポイントに対するメッセージの受信を制限する

Amazon SQS の管理権限を制限する

次の `RestrictAdminQueueActions` ポリシーステートメントは、Amazon SQS の管理権限を、キューのデプロイ、キューの管理、またはその両方に使用する 1 つまたは複数の IAM ロールのみに制限します。*<placeholder values>* をユーザー自身の情報に必ず置き換えます。Amazon SQS キューのデプロイに使用する IAM ロールの ARN と、Amazon SQS 管理権限が必要なすべての管理者ロールの ARN を指定します。

```
{
  "Sid": "RestrictAdminQueueActions",
```

```
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes"
  ],
  "Resource": "<SQS Queue ARN>",
  "Condition": {
    "StringNotLike": {
      "aws:PrincipalARN": [
        "arn:aws:iam::<account-id>:role/<deployment-role-name>",
        "<admin-role ARN>"
      ]
    }
  }
}
```

指定された組織からの Amazon SQS キューアクションを制限する

Amazon SQS リソースを外部アクセス (AWS 組織外のエンティティによるアクセス) から保護するには、次のステートメントを使用します。このステートメントは、Amazon SQS キューアクセスを `Condition` で指定した組織に制限します。必ず *`<SQS queue ARN>`* Amazon SQS キューのデプロイに使用した IAM ロールの ARN に、*`<org-id>`* を組織 ID に置き換えてください。

```
{
  "Sid": "DenyQueueActionsOutsideOrg",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:AddPermission",
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteQueue",
    "sqs:RemovePermission",
    "sqs:SetQueueAttributes",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
```

```
  "Condition": {
    "StringNotEquals": {
      "aws:PrincipalOrgID": [
        "<org-id>"
      ]
    }
  }
}
```

Amazon SQS のアクセス許可をコンシューマーに付与する

Amazon SQS キューからメッセージを受信するには、メッセージコンシューマーに必要なアクセス許可を与える必要があります。次のポリシーステートメントは、Amazon SQS キューからのメッセージを消費するために必要なアクセス許可を、指定したコンシューマーに付与します。Amazon SQS ポリシーにステートメントを追加する際は、必ず *<consumer's IAM runtime role ARN>* をコンシューマーが使用する IAM ランタイムロールの ARN に置き換え、*<SQS queue ARN>* を Amazon SQS キューをデプロイするために使用される IAM ロールの ARN に置き換えてください。

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": "<consumer's IAM execution role ARN>"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteMessage",
    "sqs:GetQueueAttributes",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>"
}
```

他のエンティティが Amazon SQS キューからメッセージを受信しないようにするには、Amazon SQS キューポリシーに `DenyOtherConsumersFromReceiving` ステートメントを追加します。このステートメントは、メッセージの消費を指定したコンシューマーに制限します。つまり、そのコンシューマーの ID アクセス許可によってアクセスが許可される場合でも、他のコンシューマーにはアクセスできなくなります。*<SQS queue ARN>* と *<consumer's runtime role ARN>* をユーザー自身の情報に必ず置き換えてください。

```
{
  "Sid": "DenyOtherConsumersFromReceiving",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ChangeMessageVisibility",
    "sqs:DeleteMessage",
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotLike": {
      "aws:PrincipalARN": "<consumer's execution role ARN>"
    }
  }
}
```

転送時の暗号化を強制する

以下の DenyUnsecureTransport ポリシーステートメントは、コンシューマーとプロデューサーが安全なチャネル (TLS 接続) を使用して Amazon SQS キューからメッセージを送受信することを強制します。*<SQS queue ARN>* を Amazon SQS キューのデプロイに使用した IAM ロールの ARN に必ず置き換えてください。

```
{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "Bool": {
      "aws:SecureTransport": "false"
    }
```

```
  }
}
```

メッセージ送信を特定の Amazon SNS トピックに制限する

以下の AllowSNSToSendToTheQueue ポリシーステートメントは、Amazon SNS トピックが指定された Amazon SQS キューへメッセージを送信できるようにします。*<SQS queue ARN>* を Amazon SQS キューのデプロイに使用した IAM ロールの ARN に、*<SNS topic ARN>* を Amazon SNS トピック ARN に必ず置き換えてください。

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

次の DenyAllProducersExceptSNSFromSending ポリシーステートメントは、他のプロデューサーがキューにメッセージを送信できないようにします。*<SQS queue ARN>* と*<SNS topic ARN>* をユーザー自身の情報に置き換えてください。

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
```

```
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

(オプション) 特定の VPC エンドポイントに対するメッセージの受信を制限する

メッセージの受信を特定の VPC エンドポイントのみに制限するには、以下のポリシーステートメントを Amazon SQS キューポリシーに追加します。このステートメントは、メッセージが目的の VPC エンドポイントからのものでない限り、メッセージコンシューマーがキューからメッセージを受信することを防ぎます。*<SQS queue ARN>* を Amazon SQS キューのデプロイに使用した IAM ロールの ARN に、および*<vpce_id>* を VPC エンドポイントの ID に置き換えてください。

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

デッドレターキューの Amazon SQS ポリシーステートメント

ステートメント ID で識別される以下のポリシーステートメントを DLQ アクセスポリシーに追加します。

- `RestrictAdminQueueActions`
- `DenyQueueActionsOutsideOrg`
- `AllowConsumersToReceiveFromTheQueue`
- `DenyOtherConsumersFromReceiving`
- `DenyUnsecureTransport`

前述のポリシーステートメントを DLQ アクセスポリシーに追加することに加えて、次のセクションで説明するように、Amazon SQS キューへのメッセージ送信を制限するステートメントも追加する必要があります。

Amazon SQS キューへのメッセージの送信を制限する

同じアカウントの Amazon SQS キューのみへのアクセスを制限するには、DLQ キューポリシーに次の `DenyAnyProducersExceptSQS` ポリシーステートメントを追加します。DLQ はメインキューを作成する前にデプロイする必要があるため、このステートメントでは特定のキューへのメッセージ送信を制限しません。DLQ の作成時に Amazon SQS ARN を知ることができないためです。1 つの Amazon SQS キューのみにアクセスを制限する必要がある場合、`Condition` 内の `aws:SourceArn` を Amazon SQS ソースキューの ARN で変更します。

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

Important

このガイドで定義されている Amazon SQS キューポリシーは、`sqs:PurgeQueue` アクションを特定の IAM ロールに制限しません。`sqs:PurgeQueue` アクションにより、Amazon SQS キューからのすべてのメッセージの削除が可能になります。このアクションを使用して、Amazon SQS キューを置き換えずにメッセージ形式を変更することもできます。アプリケーションをデバッグするときに、Amazon SQS キューをクリアして、エラーの可能性のあるメッセージを削除できます。アプリケーションをテストするときは、Amazon SQS キューに大量のメッセージを送り、本番環境に入る前にキューを消去して最初からやり直すことができます。このアクションを特定のロールに制限しない理由は、Amazon SQS キューをデプ

ロイする際にこのロールがわからない可能性があるためです。キューを削除するには、このアクセス許可をロールの ID ベースのポリシーに追加する必要があります。

サービス間での混乱した使節問題を防止する

「混乱した代理」問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。これを防ぐために、 は、アカウント内のリソースへのアクセスをサードパーティー (クロスアカウント) または他の AWS サービス (クロスサービス) に許可する場合に、アカウントを保護するのに役立つツール AWS を提供します。このセクションのポリシーステートメントは、サービス間の混乱した使節の問題を防止するのに役立ちます。

サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスが操作され、それ自体のアクセス許可を通じて、別の顧客のリソースに対して本来アクセス許可が付与されるべきではない形で働きかけが行われることがあります。この問題を防ぐため、この記事で定義されているリソースベースのポリシーでは、`aws:SourceArn`、`aws:SourceAccount`、`aws:PrincipalOrgID` およびグローバル IAM 条件コンテキストキーを使用しています。これにより、サービスが持つアクセス許可が、 AWS Organizations の特定のリソース、特定のアカウント、または特定の組織に制限されます。

IAM Access Analyzer を使用して、クロスアカウントアクセスを確認します。

AWS IAM Access Analyzer を使用して Amazon SQS キューポリシーと AWS KMS キーポリシーを確認し、Amazon SQS キューまたは AWS KMS キーが外部エンティティへのアクセスを許可すると警告できます。IAM Access Analyzer の機能は、信頼ゾーン外のエンティティと共有されている組織とアカウントのリソースを識別するのに役立ちます。この信頼ゾーンは、IAM Access Analyzer を有効にするときに指定する AWS アカウントまたは AWS Organizations 内の組織です。

IAM Access Analyzer は、ロジックベースの推論を使用して AWS 環境内のリソースベースのポリシーを分析することで、外部プリンシパルと共有されているリソースを識別します。信頼ゾーン外で共有されているリソースのインスタンスごとに、Access Analyzer は結果を生成します。結果には、アクセスと付与される外部プリンシパルに関する情報が含まれます。結果を確認して、アクセスが意図的で安全なものであるか、またはアクセスが意図しないものであるか、セキュリティ上のリスクであるかを判断します。意図しないアクセスについては、影響を受けるポリシーを確認して修正します。 AWS IAM Access Analyzer が AWS リソースへの意図しないアクセスを識別する方法の詳細については、このブログ記事を参照してください。

AWS IAM Access Analyzer の詳細については、AWS IAM Access Analyzer のドキュメントを参照してください。

Amazon SQS (Amazon SQS)、 API のアクセス権限: アクションとリソースのリファレンスについて

アクセスコントロール をセットアップし、IAMアイデンティティにアタッチできるアクセス権限ポリシーを作成するときは、以下の表をリファレンスとして使用できます。には、各 Amazon Simple Queue Service アクション、アクションを実行するためのアクセス許可を付与できる対応するアクション、およびアクセス許可を付与できる AWS リソースが含まれます。

ポリシーの`Action`フィールドでアクションを指定し、ポリシーの`Resource`フィールドでリソースの値を指定します。アクションを指定するには、`sqs:`プレフィックスに続けて アクション名を使用します (例:`sqs:CreateQueue`)。

現在、Amazon SQS は IAM で使用可能なグローバル条件コンテキストキーをサポートしています。

Amazon Simple キューサービス API およびアクション用のアクセス権限が必要です。

AddPermission

アクション:`sqs:AddPermission`

リソース: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

ChangeMessageVisibility

アクション:`sqs:ChangeMessageVisibility`

リソース: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

ChangeMessageVisibilityBatch

アクション:`sqs:ChangeMessageVisibilityBatch`

リソース: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

CreateQueue

アクション:`sqs:CreateQueue`

リソース: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

DeleteMessage

アクション:`sqs:DeleteMessage`

リソース: `arn:aws:sqs:region:account_id:queue_name`

DeleteMessageBatch

アクション:`sqs:DeleteMessageBatch`

リソース: `arn:aws:sqs:region:account_id:queue_name`

DeleteQueue

アクション:`sqs:DeleteQueue`

リソース: `arn:aws:sqs:region:account_id:queue_name`

GetQueueAttributes

アクション:`sqs:GetQueueAttributes`

リソース: `arn:aws:sqs:region:account_id:queue_name`

GetQueueUrl

アクション:`sqs:GetQueueUrl`

リソース: `arn:aws:sqs:region:account_id:queue_name`

ListDeadLetterSourceQueues

アクション:`sqs:ListDeadLetterSourceQueues`

リソース: `arn:aws:sqs:region:account_id:queue_name`

ListQueues

アクション:`sqs:ListQueues`

リソース: `arn:aws:sqs:region:account_id:queue_name`

ListQueueTags

アクション:`sqs:ListQueueTags`

リソース: `arn:aws:sqs:region:account_id:queue_name`

PurgeQueue

アクション:`sqs:PurgeQueue`

リソース: `arn:aws:sqs:region:account_id:queue_name`

ReceiveMessage

アクション:`sqs:ReceiveMessage`

リソース: `arn:aws:sqs:region:account_id:queue_name`

RemovePermission

アクション:`sqs:RemovePermission`

リソース: `arn:aws:sqs:region:account_id:queue_name`

SendMessage および SendMessageBatch

アクション:`sqs:SendMessage`

リソース: `arn:aws:sqs:region:account_id:queue_name`

SetQueueAttributes

アクション:`sqs:SetQueueAttributes`

リソース: `arn:aws:sqs:region:account_id:queue_name`

TagQueue

アクション:`sqs:TagQueue`

リソース: `arn:aws:sqs:region:account_id:queue_name`

UntagQueue

アクション:`sqs:UntagQueue`

リソース: `arn:aws:sqs:region:account_id:queue_name`

Amazon SQSのロギングとモニタリング

Amazon Simple Queue Service 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、Amazon SQS 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。キャプチャされた呼び出しには、Amazon SQS コンソールからの呼び出しとAmazon SQSオペレーションへのコード

呼び出しが含まれます。CloudTrail で収集された情報を使用して、Amazon SQS に対するリクエスト、リクエスト元の IP アドレス、リクエスト日時など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。

Amazon CloudWatch アラーム

指定した期間にわたって 1 つのメトリクスをモニタリングし、複数の期間にわたって定義されたしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。例えば、Amazon SNS トピックに通知を送信するか、Amazon SQS キューにメッセージを送信するアクションをトリガーするように CloudWatch アラームを設定できます。CloudWatch アラームは、特定の状態にあるという理由だけではアクションを実行しません。状態は変更され、定義された期間にわたってその状態を維持する必要があります。

詳細については、「Amazon SQSメトリクスの CloudWatchアラームを作成する」および「Amazon CloudWatch を使用したデッドレターキューのアラームの作成」を参照してください。

Amazon CloudWatch Logs

CloudWatch Logs にAmazon SQS に関連するログファイルを監視、保存、およびアクセスします。これらのログを使用して、メッセージ処理の分析、問題のデバッグ、Amazon SQS ワークフローのパフォーマンスのモニタリングを行うことができます。

詳細については、「を使用した Amazon Simple Queue Service API コールのログ記録 AWS CloudTrail」を参照してください。

Amazon CloudWatch Events

Amazon CloudWatch Events を使用して、 AWS 環境内の変更や特定のイベントを検出し、Amazon SQS キューにルーティングします。これにより、イベントデータをキャプチャしたり、ワークフローをトリガーしたり、後で処理するためにイベントを保存したりできます。

詳細については、このガイドの「Amazon EventBridge を使用して AWS サービスから Amazon SQS への通知を自動化する」および「Amazon EventBridge ユーザーガイド」の「EventBridge is the evolution of Amazon CloudWatch Events」を参照してください。

AWS CloudTrail ログ

CloudTrail は、ユーザー、ロール、または によって Amazon SQS で実行されたアクションの詳細なレコードをキャプチャします AWS のサービス。これらのログにより、、`SendMessage`、`ReceiveMessage`などの API コールを追跡し`DeleteQueue`、リクエストの実行者、発生日時、送信元の IP アドレスなどの主要な詳細を提供できます。

詳細については、「を使用した Amazon Simple Queue Service API コールのログ記録 AWS CloudTrail」を参照してください。

AWS Trusted Advisor

Trusted Advisor は AWS 、顧客へのサービス提供から開発されたベストプラクティスを使用して、Amazon SQS の使用を最適化します。Amazon SQS キューを確認し、セキュリティを強化し、メッセージ処理の信頼性を向上させ、コストを削減するための実用的な推奨事項を提供します。たとえば、デッドレターキューを有効にするか、キューアクセスポリシーを改善して安全なオペレーションを確保することをお勧めします。

詳細については、サポート ユーザーガイドの AWS Trusted Advisorを参照してください。

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 Management Console 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 で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タ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タが制御します。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。

を使用した Amazon Simple Queue Service API コールのログ記録 AWS CloudTrail

CloudTrail では、データイベントと管理イベントの 2 つのイベントタイプを使用して Amazon SQS オペレーションをログに記録し、モニタリングできます。これにより、アカウント内の Amazon SQS アクティビティを簡単に追跡および監査できます。

CloudTrail の Amazon SQS データイベント

データイベントは、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に関する情報を提供します (Amazon SQS オブジェクトへのメッセージの送信など) 。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、Amazon SQS 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 Management Console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。

CloudTrail で Amazon SQS データイベントをログに記録するには、高度なイベントセレクタを使用して、ログに記録する特定の Amazon SQS リソースまたはアクションを設定する必要があります。リソースタイプを含めて`AWS::SQS::Queue`、キュー関連のアクションをキャプチャします。などのフィルター `eventName` (`SendMessage`イベントなど) を使用して、ログ記録の設定をさらに絞り込むことができます。詳細については、CloudTrail API リファレンスのAdvancedEventSelector」を参照してください。

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
Amazon SQS キュー	`AWS::SQS::Queue`	• ChangeMessageVisibility • ChangeMessageVisibilityBatch • DeleteMessage • DeleteMessageBatch • GetQueueAttributes • GetQueueUrl • ListDeadLetterSourceQueues • ListQueues • ListQueueTags • ReceiveMessage • SendMessage • SendMessageBatch

高度なイベントセレクタを使用してフィールドをフィルタリングし、重要なイベントのみをログに記録します。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。

CloudTrail での Amazon SQS 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。

Amazon SQS は、以下の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
- CancelMessageMoveTask
- CreateQueue
- DeleteQueue
- ListMessageMoveTasks
- PurgeQueue
- RemovePermission
- SetQueueAttributes
- StartMessageMoveTask
- TagQueue
- UntagQueue

Amazon SQS 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。

次の例は、`SendMessage` オペレーションを示す CloudTrail イベントを示しています。

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EXAMPLE_PRINCIPAL_ID",
    "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed/SessionName",
    "accountId": "123456789012",
```

```
    "accessKeyId": "ACCESS_KEY_ID",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:sts::123456789012:assumed-role/RoleToBeAssumed",
        "accountId": "123456789012",
        "userName": "RoleToBeAssumed"
      },
      "attributes": {
        "creationDate": "2023-11-07T22:13:06Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2023-11-07T23:59:11Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SendMessage",
  "awsRegion": "ap-southeast-4",
  "sourceIPAddress": "10.0.118.80",
  "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": {
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue",
    "messageBody": "HIDDEN_DUE_TO_SECURITY_REASONS",
    "messageDeduplicationId": "MsgDedupIdSdk1ae1958f2-bbe8-4442-83e7-4916e3b035aa",
    "messageGroupId": "MsgGroupIdSdk16"
  },
  "responseElements": {
    "mD5OfMessageBody": "9a4e3f7a614d9dd9f8722092dbda17a2",
    "mD5OfMessageSystemAttributes": "f88f0587f951b7f5551f18ae699c3a9d",
    "messageId": "93bb6e2d-1090-416c-81b0-31eb1faa8cd8",
    "sequenceNumber": "18881790870905840128"
  },
  "requestID": "c4584600-fe8a-5aa3-a5ba-1bc42f055fae",
  "eventID": "98c735d8-70e0-4644-9432-b6ced4d791b1",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "type": "AWS::SQS::Queue",
      "ARN": "arn:aws:sqs:ap-southeast-4:123456789012:MyQueue"
    }
```

```
  ],
  "eventType": "AwsApiCall",
  "managementEvent": false,
  "recipientAccountId": "123456789012",
  "eventCategory": "Data",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "sqs.ap-southeast-4.amazonaws.com"
  }
```

> **Note**
>
> `ListQueues` オペレーションは特定のリソースに対して動作しないため、一意のケースです。その結果、ARN フィールドにはキュー名が含まれず、代わりにワイルドカード (*) が使用されます。

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 SQSキューのモニタリング

Amazon SQSとAmazon CloudWatchが統合されているため、CloudWatchを使用して Amazon SQSキューのメトリクスを表示および分析できます。キューのメトリックスを表示および分析するには、Amazon SQS コンソールとすると、CloudWatchコンソールと定義し、AWS CLI、またはCloudWatch APIを使用します。また、Amazon SQSメトリクスに対するCloudWatchアラームの設定の場合。

Amazon SQSキューの CloudWatchメトリクスは、1 分間隔で自動的に収集され、CloudWatchにプッシュされます これらのメトリックスは、CloudWatchガイドラインを満たすすべてのキューで収集され、アクティブになります。CloudWatch では、キュー内にメッセージがある場合や、いずれかのアクションがキューにアクセスしている場合、そのキューはアクティブであると見なされます。

Amazon SQS キューが 6 時間以上非アクティブになると、Amazon SQS サービスはスリープ状態と見なされ、CloudWatch サービスへのメトリクスの配信を停止します。Amazon SQS キューが非アクティブだった期間の Amazon SQS の CloudWatch メトリクスでは、欠落しているデータやゼロを表すデータを視覚化することはできません。

> **Note**
>
> - Amazon SQS キューは、キューに対して API を呼び出すユーザーが承認されず、リクエストが失敗すると、アクティブ化される場合があります。
> - Amazon SQS コンソールは、キューのページが開くと、`GetQueueAttributes` API コールを実行します。`GetQueueAttributes` API リクエストにより、キューがアクティブ化されます。
> - キューが非アクティブ状態からアクティブ化される場合、CloudWatch メトリクスで最大 15 分の遅延が発生します。
> - CloudWatch で報告される Amazon SQS メトリクスに料金はかかりません。これらは Amazon SQSサービスの一部として提供されます。
> - 標準キューと FIFOキューの両方で CloudWatch メトリクスがサポートされています。

Amazon SQS向けのCloudWatchメトリクスへのアクセス

Amazon SQSとAmazon CloudWatchが統合されているため、CloudWatchを使用して Amazon SQSキューのメトリクスを表示および分析できます。キューのメトリックスを表示および分析するには、Amazon SQS コンソールとすると、CloudWatchコンソールと定義し、AWS CLI、またはCloudWatch APIを使用します。また、Amazon SQSメトリクスに対するCloudWatchアラームの設定の場合。

Amazon SQS コンソールの使用

Amazon SQS コンソールを使用して、最大 10 個の Amazon SQS キューのメトリクスにアクセスして分析します。

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、メトリクスにアクセスするキューのチェックボックスを選択 (オンに) します。最大10個のキューのメトリクスを表示できます。
3. モニタリングタブを選択します。

   さまざまなグラフが [SQS metrics] セクションに表示されます。
4. 特定のグラフが表す内容を理解するには、目的のグラフの横の

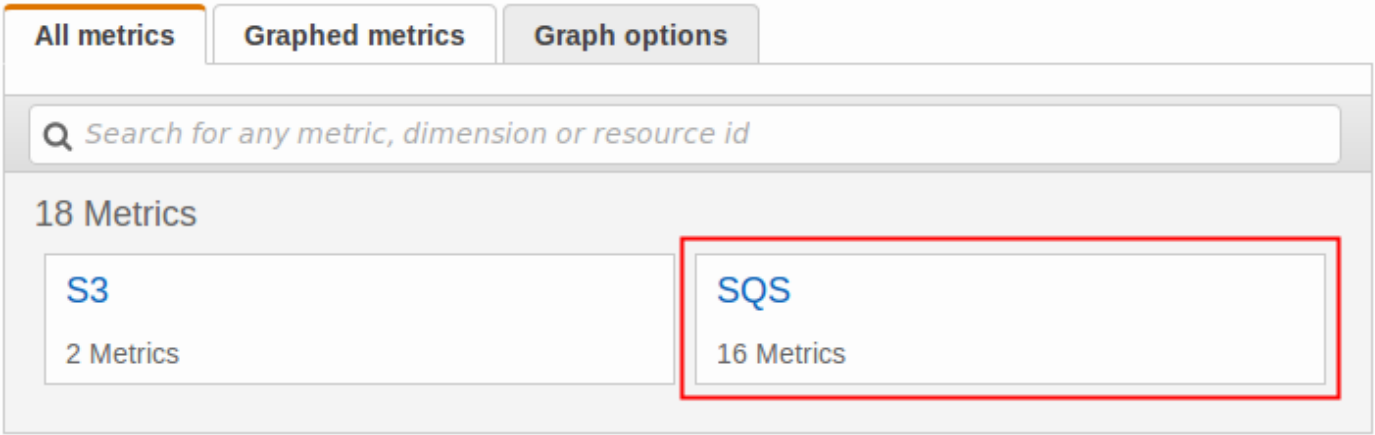
にマウスを移動するか、「Amazon SQS で利用可能な CloudWatchメトリクス」を参照してください。

5. すべてのグラフの時間範囲を同時に変更するには、[Time Range] で、目的の時間範囲 (たとえば、[Last Hour]) を選択します。
6. 個別のグラフの追加の統計を表示するには、そのグラフを選択します。
7. CloudWatchモニタリングディテール ダイアログボックスで、統計を選択します (たとえば 合計)。サポートされている統計のリストについては、「Amazon SQS で利用可能な CloudWatch メトリクス」を参照してください。
8. 個別のグラフで表示される時間の範囲と間隔を変更するには (たとえば、過去 5 分間ではなく過去 24 時間の範囲を表示したり、5 分ごとではなく 1 時間ごとの期間を表示するなど)、グラフのダイアログボックスが表示された状態で、[Time Range] の目的の時間範囲 (たとえば、[Last 24 Hours]) を選択します。[Period] で、指定された時間範囲 (たとえば、[1Hour]) 内で目的の期間を選択します。グラフの表示を終了するときは、[Close] を選択します。
9. (オプション) その他の の機能を使用するには、{モニタリング }タブで [すべてのCloudWatch メトリックスを見る]を選択し、(Amazon CloudWatch コンソールの使用)の手順に従います。

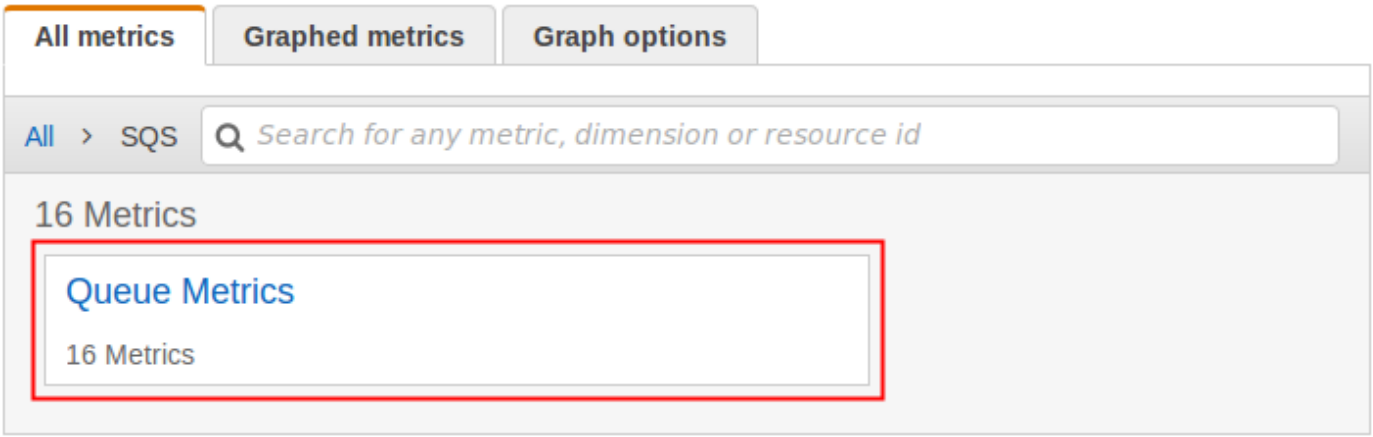
Amazon CloudWatch コンソールの使用

CloudWatch コンソールを使用して、Amazon SQS メトリクスにアクセスして分析します。

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. [SQS] メトリクスの名前空間を選択します。

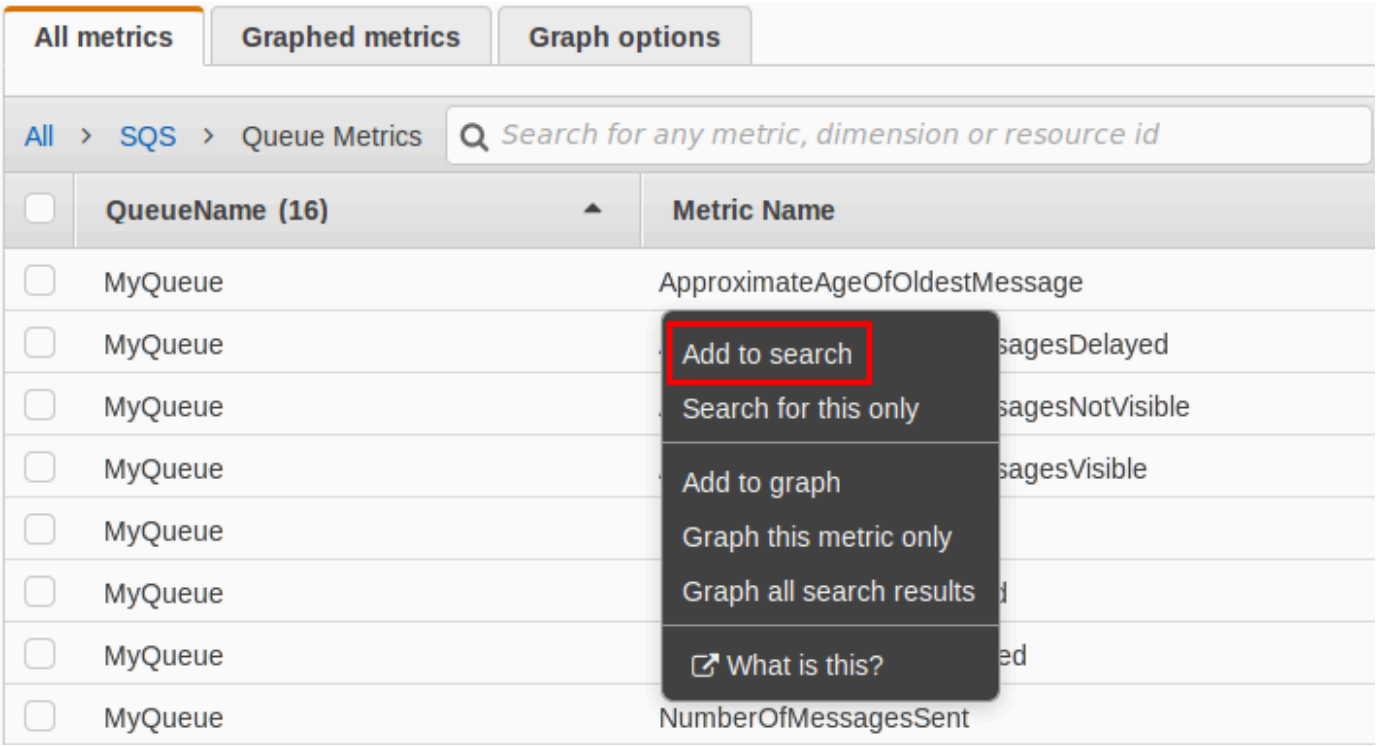


4. [Queue Metrics] メトリクスディメンションを選択します。



5. これで、Amazon SQSメトリクスを調べることができるようになりました。

- メトリクスを並べ替えるには、列見出しを使用します。
- メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。
- メトリクスでフィルタするには、メトリクスの名前を選択し、[Add to search] を選択します。



詳細およびその他のオプションについては、「Amazon CloudWatch ユーザーガイドのメトリクスをグラフ化そしてAmazon CloudWatch ダッシュボードの使用」を参照してください。

の使用 AWS Command Line Interface

を使用して Amazon SQS メトリクスにアクセスするには AWS CLI、 `get-metric-statistics` コマンドを実行します。

詳細については、Amazon CloudWatch ユーザーガイドの「メトリクスの統計の取得」を参照してください。

CloudWatch API の使用

CloudWatch APIを使用して Amazon SQSメトリクスにアクセスするには、`GetMetricStatistics` アクションを使用します。

詳細については、Amazon CloudWatch ユーザーガイドの「メトリクスの統計の取得」を参照してください。

Amazon SQSメトリクスの CloudWatchアラームを作成する

CloudWatch では、メトリクスが指定されたしきい値に達したときにアラームをトリガーできます。たとえば、`NumberOfMessagesSent` メトリクスのアラームを作成できます。たとえば、1時間以内に100件以上のメッセージが `MyQueue` キューに送信された場合、Eメール通知が送信されます。詳細については、Amazon CloudWatchユーザーガイド の「Amazon CloudWatchアラームの作成」を参照してください。

1. にサインイン AWS Management Console 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。
3. [アラームの作成] ダイアログボックスの [メトリクスの選択] セクションで、[メトリクスの参照]、[SQS] を選択します。
4. [SQS > Queue Metrics] で、アラームを設定する [QueueName] および [メトリクス名] を選択して、[次へ] を選択します。使用可能なメトリクスのリストについては、Amazon SQS で利用可能な CloudWatchメトリクス を参照してください。

   次の例では、`NumberOfMessagesSent`キューの`MyQueue`メトリクスのアラームが選択されています。送信されたメッセージの数が100を超えるとアラームがトリガーされます。
5. [アラームの作成] ダイアログボックスの、[アラームの定義] セクションで、次の操作を行います。

   a. [アラームのしきい値] で、アラームの [名前] と [説明] を入力します。

b. [is] を [> 100] に設定します。

c. [for] を [1 out of 1 datapoints (1つのデータポイントにつき 1 つ)] に設定します。

d. [アラームのプレビュー] で、[間隔] を [1時間] に設定します。

e. [統計] を [スタンダード]、[合計] に設定します。

f. [アクション] の [アラームが次の時] に、[状態: 警告] を設定します。

アラームがトリガーされたときに CloudWatchが通知を送信する場合は、既存のAmazon SNSトピックを選択するか、{新しいリスト} を選択して、E メールアドレスをカンマで区切って入力します。

> **Note**
> 新しいAmazon SNSトピックを作成する場合、E 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メールアドレスが検証される前にアラーム状態が変更されると、通知は配信されません。

6. [Create Alarm] (アラームの作成) を選択します。

アラームが作成されます。

Amazon SQS で利用可能な CloudWatchメトリクス

Amazon SQS は、次のメトリクスをCloudWatchに送信します。

> **Note**
> 一部のメトリクスでは、Amazon SQS の分散アーキテクチャにより、結果は概算になります。ほとんどの場合、カウントはキュー内の実際のメッセージ数に近い数値になります。

Amazon SQSメトリクス

Amazon SQS は、 `AWS/SQS`名前空間の Amazon CloudWatch に運用メトリクスを自動的に発行します。これらのメトリクスは、キューのヘルスとパフォーマンスをモニタリングするのに役立ちます。SQS は分散型であるため、多くの値は概算ですが、ほとんどの運用上の決定には十分正確です。

> **Note**
>
> - すべてのメトリクスは、キューがアクティブな場合にのみ負以外の値を出力します。
> - 一部のメトリクス (などSentMessageSize) は、少なくとも 1 つのメッセージが送信されるまで出力されません。

メトリクス	説明	単位	レポート動作	キーノート
ApproximateAgeOfOldestMessage	キュー内の最も古い未処理メッセージの経過時間。	[秒]	キューにアクティブなメッセージが少なくとも 1 つ含まれている場合に報告されます。	• 標準キューの場合、メッセージが 3 回以上受信され、削除されない場合、SQS はキューの背面に移動します。次に、メトリクスは受信しきい値を超えていない次のメッセージの経過時間を反映します。この順序変更は、リドライブポリシーが設定されている場合でも発生します。 • ポイズンピルメッセージ (繰り返し受信されたが削除されていないメッセージ) は、正常に処理されるまでこのメトリクスから除外されます。 • を超えた後にメッセージが DLQ に移動するとmaxReceiveCount 、経過時間はリセットされます。この場合、DLQ の メトリクスには、メッセージが最初に送信された時刻ではなく、メッセージが移動された時刻が反映されます。

メトリクス	説明	単位	レポート動作	キーノート
				• FIFO キューは、順序を維持するためにメッセージの順序を変更しません。失敗したメッセージは、メッセージグループが削除されるか期限切れになるまでブロックします。DLQ が設定されている場合、メッセージは受信しきい値が満たされた後に送信されます。
`ApproximateNumberOfGroupsWithInflightMessages`	FIFO のみ。1 つ以上の処理中のメッセージを含むメッセージグループの数。	カウント	FIFO キューがアクティブである場合に報告されます。	• メッセージは、コンシューマーがキューから受信したが、まだ削除または期限切れになっていない場合、処理中と見なされます。 • このメトリクスは、FIFO キューのスループットのトラブルシューティングと最適化に役立ちます。通常、高い値は強い同時実行数を示します。 • キューのバックログが大きく、この値が小さい場合は、コンシューマーをスケーリングするか、アクティブなメッセージグループの数を増やすことを検討してください。 • スループットと処理中の制限については、「」を参照してください[Amazon SQS のクォータ](#)。

メトリクス	説明	単位	レポート動作	キーノート
ApproximateNumberOfMessagesDelayed	キュー内で遅延し、すぐに取得できないメッセージの数。	カウント	遅延メッセージがキューに存在する場合に報告されます。	• デフォルトの遅延で設定されたキューと、 DelaySeconds パラメータで送信された個々のメッセージに適用されます。 • 遅延メッセージは、遅延期間が終了するまでコンシューマーから非表示のままになり、認識されたキューのバックログまたはスループットに影響する可能性があります。
ApproximateNumberOfMessagesNotVisible	受信されたが、まだ削除または期限切れになっていない処理中のメッセージの数。	カウント	処理中のメッセージが存在する場合に報告されます。	• メッセージは、ReceiveMessage API 経由でコンシューマーに送信されると、処理中の状態になります。 • これらのメッセージは、可視性タイムアウトウィンドウ中に他のコンシューマーから一時的に非表示になります。 • このメトリクスを使用して、メッセージ処理の遅延やコンシューマーのスタックを追跡します。

メトリクス	説明	単位	レポート動作	キーノート
ApproximateNumberOfMessagesVisible	現在取得と処理に使用できるメッセージの数。	カウント	キューがアクティブの場合に報告されます。	• キュー内の現在の処理バックログを反映します。 • 蓄積できるメッセージの数にはハード制限はありませんが、キューで設定された保持期間が適用されます。 • 値が一貫して高い場合は、プロビジョニング不足のコンシューマーや処理ロジックがスタックしている可能性があります。
NumberOfEmptyReceives 1	メッセージが返されなかった ReceiveMessage API コールの数。	カウント	受信オペレーション中に報告されます。	• このメトリクスは、ポーリング動作の非効率性や十分に活用されていないコンシューマーインスタンスを特定するのに役立ちます。 • キューが空の場合、コンシューマーがショートポーリングを使用する場合、またはメッセージが生成されたよりも速く処理されている場合、高い値が発生することがあります。 • これはキューの状態を正確に示すものではありません。サービス側の動作を反映し、再試行が含まれる場合があります。

メトリクス	説明	単位	レポート動作	キーノート
NumberOfDeduplicatedSentMessages	FIFO のみ。重複排除され、キューに追加されなかった送信メッセージの数。	カウント	重複したMessageDeduplicationId 値またはコンテンツが検出された場合に報告されます。	• SQS は、 MessageDeduplicationId または コンテンツベースのハッシュ (有効になっている場合) に基づいてメッセージを重複排除します。 • 値が大きいと、プロデューサーが 5 分間の重複排除ウィンドウ内で同じメッセージを繰り返し送信している可能性があります。 • このメトリクスを使用して冗長なプロデューサーロジックをトラブルシューティングするか、重複排除が意図したとおりに機能していることを確認します。

メトリクス	説明	単位	レポート動作	キーノート

NumberOfMessagesDeleted 1	キューから正常に削除されたメッセージの数。	カウント	有効な受信ハンドルを持つ削除リクエストごとにレポートされます。	• このメトリクスは、同じメッセージが複数回削除された場合でも、成功したすべての削除オペレーションをカウントします。
				• higher-than-expected一般的な理由は次のとおりです。
				  • 可視性タイムアウトが期限切れになり、メッセージが再度受信された後、異なる受信ハンドルを使用して同じメッセージを複数回削除します。
				  • 同じ受信ハンドルを使用した重複削除は、成功ステータスを返し、メトリクスを増分します。
				• このメトリクスを使用してメッセージ処理の成功を追跡しますが、一意の削除済みメッセージの正確な数として扱わないでください。

メトリクス	説明	単位	レポート動作	キーノート
NumberOfMessagesReceived 1	ReceiveMessage API によって返されたメッセージの数。	カウント	受信オペレーション中に報告されます。	• これには、可視性タイムアウトの有効期限が切れたために後でキューに返されるメッセージを含む、コンシューマーに返されるすべてのメッセージが含まれます。 • 削除しない場合、1 つのメッセージを複数回受信できます。これにより、このメトリクスが送信されたメッセージの数を超える可能性があります。 • これを使用してコンシューマーのアクティビティを追跡しますが、処理された一意のメッセージの数として扱いません。

メトリクス	説明	単位	レポート動作	キーノート
NumberOfMessagesSent 1	キューに正常に追加されたメッセージの数。	カウント	手動送信が成功するたびにレポートされます。	• DLQ を直接ターゲットとする呼び出しを含め、SendMessage または への手動呼び出しSendMessageBatch がカウントされます。 • を超えた後に DLQ に自動的に移動されるメッセージmaxReceiveCount は、このメトリクスに含まれません。 • その結果、 は よりも低いNumberOfMessagesSent 場合がありますNumberOfMessagesReceived 。特に、リドライブポリシーが多くのメッセージを背後にある DLQsに移動している場合です。
SentMessageSize 1	キューに正常に送信されたメッセージのサイズ。	バイト	少なくとも 1 つのメッセージが送信されるまで出力されません。	• このメトリクスは、キューが最初のメッセージを受信するまで CloudWatch コンソールに表示されません。 • このメトリクスを使用して、各メッセージのサイズをバイト単位で追跡します。これは、ペイロードの傾向の分析やスループットコストの見積もりに役立ちます。 • SQS の最大メッセージサイズは 1 MiB です。

メトリクス	説明	単位	レポート動作	キーノート
ApproximateNumberOfNoisyGroups	公平キューでノイズが多いと見なされるメッセージグループの数。ノイズの多いメッセージグループは、マルチテナントキューのノイズの多い近隣テナントを表します。	カウント	キューがアクティブの場合、負以外の値がレポートされます。	• 不均衡なリソースを消費するメッセージグループを追跡することで、マルチテナント環境で発生する可能性のあるノイズの多い近隣問題を特定するのに役立ちます。 • このメトリクスを使用して、ノイズの多いグループの数が許容しきい値を超えたときにトリガーされるアラームを設定します。これは、キューの公平性に関する潜在的な問題を示します。
ApproximateNumberOfMessagesVisibleInQuietGroups	ノイズの多いメッセージグループからのメッセージを除外して表示されるメッセージの数。	カウント	キューがアクティブな場合、負以外の値がレポートされます。	• ノイズの多い近隣からのメッセージを除く、標準レートのメッセージグループのキューバックログを可視化します。 • ノイズの多い近隣の影響を除外することで、一般的なメッセージグループの真の処理バックログを特定するのに役立ちます。

メトリクス	説明	単位	レポート動作	キーノート
ApproximateNumberOfMessagesNotVisibleInQuietGroups	ノイズの多いメッセージグループからのメッセージを除く、進行中のメッセージの数。	カウント	キューがアクティブな場合、負以外の値がレポートされます。	• 適切に動作しているメッセージグループからの処理中のメッセージ (処理されているが、まだ削除されていない) を追跡します。 • このメトリクスを使用して、通常のメッセージグループの処理スループットをモニタリングし、ノイズの多い近隣によって引き起こされない処理のボトルネックを検出します。
ApproximateNumberOfMessagesDelayedInQuietGroups	遅延したノイズの多いメッセージグループからのメッセージを除外し、すぐに読み取ることができないメッセージの数。遅延メッセージは、キューが遅延キューとして設定されている場合、または遅延パラメータを使用してメッセージを送信した場合に発生します。	カウント	キューがアクティブな場合、負以外の値が報告されます。	• (大量のグループやノイズの多いグループではなく) 通常のスループットパターンまたは予想されるスループットパターンを持つメッセージグループからの遅延メッセージバックログをモニタリングするのに役立ちます。 • 一般的なワークロードの将来の処理要件と容量計画を理解するのに役立ちます。

メトリクス	説明	単位	レポート動作	キーノート
`ApproximateAgeOfOldestMessageInQuietGroups`	ノイズの多いメッセージグループからのメッセージを除く、キュー内の削除されていない最も古いメッセージの経過時間。	[秒]	キューがアクティブな場合、負以外の値がレポートされます。	• SLA コンプライアンスをモニタリングし、通常のスループットパターンまたは予想されるスループットパターンを持つメッセージグループの処理ボトルネックを検出するために使用されます (メトリクスを歪める可能性のある大量のメッセージグループやノイズの多いメッセージグループとは異なります)。 • このメトリクスを使用して、ノイズの多い近隣からの人為的に古くなったメッセージを無視するメッセージ処理タイムアウトのアラームを設定します。

1 これらのメトリクスはシステムレベルのアクティビティを反映し、再試行、重複、遅延メッセージが含まれる場合があります。raw カウントを使用して、メッセージライフサイクルの動作を考慮せずにリアルタイムキューの状態を推定しないでください。

デッドレターキュー (DLQs) と CloudWatch メトリクス

DLQs を使用する場合は、Amazon SQS メトリクスの動作を理解することが重要です。

- **`NumberOfMessagesSent`** – このメトリクスの動作は DLQsによって異なります。
  - 手動送信 – DLQ に手動で送信されたメッセージは、このメトリクスによってキャプチャされます。
  - 自動再処理 – 処理の失敗によって DLQ に自動的に移動されたメッセージは、このメトリクスではキャプチャされません。その結果、 `NumberOfMessagesSent`および `NumberOfMessagesReceived`メトリクスに DLQsの不一致が表示される場合があります。

- DLQs の推奨メトリクス – DLQ の状態をモニタリングするには、`ApproximateNumberOfMessagesVisible`メトリクスを使用します。このメトリクスは、DLQ で現在処理可能なメッセージの数を示します。

フェアキューと CloudWatch メトリクス

公平キューを使用すると、Amazon SQS は次の追加のメトリクスを出力します。

- `ApproximateNumberOfNoisyGroups`
- `ApproximateNumberOfMessagesVisibleInQuietGroups`
- `ApproximateNumberOfMessagesNotVisibleInQuietGroups`
- `ApproximateNumberOfMessagesDelayedInQuietGroups`
- `ApproximateAgeOfOldestMessageInQuietGroups`

> **Note**
>
> 各`QuietGroup`メトリクスは同等の標準キューレベルの`Approximate`メトリクスのサブセットですが、ノイズの多い近隣グループからメッセージを除外します。

ノイズの多いグループ

ノイズの多いメッセージグループは、マルチテナントキューのノイズの多い近隣テナントを表します。

クワイエットグループ

ノイズの多いグループを除くメッセージグループ。

SQS 公平キューの動作を観察する

Amazon SQS 公平キューの効果をモニタリングするには、`Approximate..InQuietGroups`メトリクスを標準のキューレベルのメトリクスと比較します。特定のテナントのトラフィックが急増すると、一般的なキューレベルのメトリクスで、バックログの増加やメッセージ経過時間が古くなる可能性があります。ただし、クワイエットグループを個別に見ると、ノイズのないメッセージグループやテナントのほとんどが影響を受けていないことを特定し、影響を受けるメッセージグループの総数の見積もりを提供できます。

これらの新しいメトリクスは、Amazon SQS 公平キューの動作に関する適切な概要を提供しますが、どの特定のテナントが負荷を引き起こしているかを理解することは有益です。Amazon CloudWatch 寄稿者インサイトを使用すると、上位 1 位の寄稿者、一意の寄稿者の総数、およびその使用状況に関するメトリクスを表示できます。これは、従来のメトリクスを出力するときにカーディナリティデータ (およびコスト) が高くなる数千のテナントを扱うシナリオで特に役立ちます。

公平キューの設定をモニタリングする例については、GitHub のサンプルを参照してください。

Amazon SQS メトリクスのディメンション

CloudWatch の Amazon SQS メトリクスは、単一のディメンション を使用します**QueueName**。すべてのメトリクスデータは、キューの名前でグループ化およびフィルタリングされます。

モニタリングのヒント

キーメトリクスと CloudWatch アラームを使用して SQS を効果的にモニタリングし、キューのバックログを検出し、パフォーマンスを最適化して、サービスの制限内を維持します。

- に基づいて CloudWatch アラームを設定`ApproximateNumberOfMessagesVisible`して、バックログの増加をキャッチします。
- をモニタリング`NumberOfEmptyReceives`してポーリング頻度を調整し、API コストを削減します。
- FIFO キュー`ApproximateNumberOfGroupsWithInflightMessages`で を使用してスループット制限を診断します。
- SQS クォータを確認して、メトリクスのしきい値とサービスの制限を理解します。

Amazon SQSのコンプライアンス検証

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、AWS のサービス 「コンプライアンスプログラムによる範囲内」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 「 Compliance ProgramsAssurance」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。 は、コンプライアンスに役立つ以下のリソース AWS を提供します。

- セキュリティのコンプライアンスとガバナンス – これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする手順を示します。
- HIPAA 対応サービスのリファレンス – HIPAA 対応サービスの一覧が提供されています。すべての AWS のサービス が HIPAA の対象となるわけではありません。
- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドは、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールを保護し、そのガイダンスに AWS のサービス マッピングするためのベストプラクティスをまとめたものです。
- 「 デベロッパーガイド」の「ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – 不審なアクティビティや悪意のあるアクティビティがないか環境をモニタリングすることで AWS アカウント、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件に対応できます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon SQSの耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、高冗長ネットワークで接続された複数の物理的に分離されたアベイラビリティーゾーンと分離されたアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビ

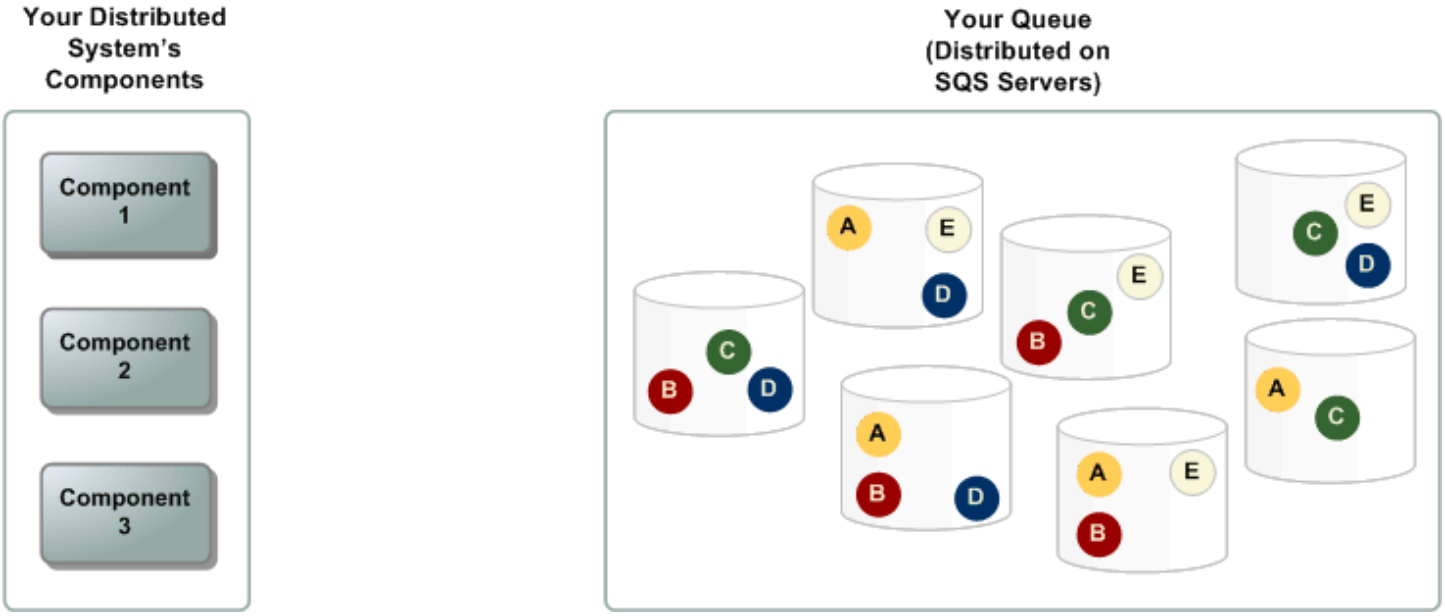
リティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。 AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「 グローバルインフラストラクチャ」を参照してください。

AWS グローバルインフラストラクチャに加えて、Amazon SQS は分散キューを提供します。

分散キュー

分散メッセージングシステムには、分散システムのコンポーネント、キュー (Amazon SQS サーバーで分散) 、キュー内のメッセージの 3 つの主要部分があります。

次のシナリオでは、システムには複数のプロデューサー (キューにメッセージを送信するコンポーネント) とコンシューマー (キューからメッセージを受信するコンポーネント) があります。キュー (A から E までのメッセージを保持) は、複数のAmazon SQS サーバー全体にまたがって冗長的にメッセージを保存します。



Amazon SQS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 SQS は、ホワイトペーパー「Amazon Web Services: セキュリティプロセスの概要」に記載されている AWS グローバルネットワークセキュリティ手順で保護されています。

AWS 公開された API アクションを使用して、ネットワーク経由で Amazon SQS 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 SQS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 SQSポリシーを使用して、特定の AmazonVPC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、ネットワーク内の特定の VPC からのみ、特定の Amazon SQS キューへの AWS ネットワークアクセスを効果的に分離できます。詳細については、「例5:VPC エンドポイントからではない場合はアクセスを拒否する」を参照してください。

Amazon SQSのセキュリティベストプラクティス

AWS には Amazon SQS 用の多くのセキュリティ機能が用意されています。これは、独自のセキュリティポリシーのコンテキストで確認する必要があります。以下に、Amazon SQS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。

> **Note**
>
> 提供される特定の実装ガイダンスは、一般的なユースケースと実装用です。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。

キューがパブリックアクセス可能でないことの確認

インターネット上のユーザーが Amazon SQS キューを読み書きできるように明示的に要求しない限り、キューにパブリックにアクセスできない (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) ことを確認する必要があります。

- `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

アクセス許可を付与する場合、アクセス許可を受け取るユーザー、アクセス許可の対象となるキュー、およびこれらのキューに対して許可する特定の APIアクションを決定します。最小権限を実装することは、セキュリティリスクを軽減し、エラーや悪意のあるインテントの影響を軽減するために重要です。

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。これで、セキュリティポリシーの組み合わせを使用して実装できます。

Amazon SQSはプロデューサー-コンシューマーモデルを使用し、次の3種類のユーザーアカウントアクセスを使用します。

- 管理者-キューの作成、変更、削除にアクセスします。管理者は、キューポリシーも制御します。
- プロデューサー-キューへのメッセージの送信にアクセスします。
- コンシューマー-キューからのメッセージの受信および削除にアクセスします。

詳細については、次のセクションを参照してください。

- Amazon SQSでの Identity and Access Management
- Amazon SQS (Amazon SQS)、 API のアクセス権限: アクションとリソースのリファレンスについて
- Amazon SQSアクセスポリシー言語を使用したカスタムポリシーの使用

Amazon SQS 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 SQS

Amazon EC2 などのアプリケーションや AWS サービスが Amazon SQS キュー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。

IAM ロールを使用して、Amazon SQS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。ロールを使用する場合、長期的な認証情報 (ユーザー名、パスワード、アクセスキーなど) を EC2 インスタンスや AWS サービスに配布する必

要はありません AWS Lambda。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 SQS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、アクセスする際にメッセージを復号します。SSE は で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、お客様がアクセス許可を持っていれば、キュー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、「Amazon SQS での保管中の暗号化」および「Amazon SQS のキー管理」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

HTTPS(TLS)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。キューポリシーの `aws:SecureTransport` 条件を使用して、HTTPS(TLS) 経由の暗号化された接続のみを許可し、リクエストに SSL の使用を強制します。

VPC エンドポイントを使用して Amazon SQSにアクセスすることを検討する場合には

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キューがある場合は、VPC エンドポイントを使用して、特定の VPC 内のホストへのアクセスのみをキューに入れます。キュー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からのキューへのアクセスを制御できます。

Amazon SQS 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。

- キューポリシーを使用して、どのVPCまたは VPC エンドポイントがキューにアクセスできるかを制御できます。

詳細については、「Amazon SQSのAmazon Virtual Private Cloud エンドポイント」および「Amazon SQS用のAmazon 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。

Amazon SQSの関連リソース

次の表は、本サービスを利用する際に役立つと思われる関連リソースの一覧です。

リソース	説明
Amazon Simple Queue Service API リファレンス	アクション、パラメータ、データタイプに関する説明と、サービスから返されるエラーのリスト。
AWS CLI コマンドリファレンスで見る Amazon SQS	キューの操作に使用できる AWS CLI コマンドの説明。
のリージョンとエンドポイント	Amazon SQSのリージョンとエンドポイントに関する情報
製品ページ	Amazon SQS に関する情報の主要なウェブページ。
フォーラム	Amazon SQSに関連する技術的な質問について話し合うデベロッパーのためのコミュニティベースのフォーラムです。
AWS プレミアムサポート情報	AWS インフラストラクチャサービスでのアプリケーションの構築と実行に役立つ one-on-one の高速応答サポートチャネルである AWS プレミアムサポートに関する情報のプライマリウェブページ。

ドキュメント履歴

次の表は、 Amazon Simple Queue Service デベロッパーガイドの2019年1月以降の重要な変更点をまとめたものです。このドキュメントの更新に関する通知については、「RSS フィード」を参照してください。

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
Fair Queues のサポート	Fair Queues のサポートが標準キューに追加されました。	2025 年 7 月 21 日
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 SQS はデュアルスタック (IPv4 および IPv6) エンドポイントをサポートするようになり、両方の IP プロトコルを介してキューにアクセスできるようになりました。	2025 年 4 月 17 日
すべての Amazon SQS APIs の CloudTrail 統合 Amazon SQS	CloudTrail 統合がすべての Amazon SQS API APIs。	2025年1月10日
SQSUnlockQueuePolicy	Amazon SQS は、 という新しい AWS管理ポリシーを追加してキューを`SQSUnlockQueuePolicy` ロック解除し、すべてのプリンシパルが Amazon SQS キューにアクセスすることを拒否する誤って設定されたキューポリシーを削除しました。	2024 年 11 月 15 日

AWS kms:Decrypt	Amazon SQS では `SendMessage` API 用の `kms:Decrypt` アクセス許可が不要になりました。キューの暗号化に必要なのは、KMS キーに対する `kms:GenerateDataKey` アクセス許可のみとなりました。ただし、`ReceiveMessage` を呼び出すには引き続き `kms:Decrypt` アクセス許可が必要です。	2024 年 7 月 24 日
FIFO メトリクスの更新	Amazon SQS の FIFO メトリクスに `NumberOfDeduplicatedSentMessages` と `ApproximateNumberOfGroupsWithInflightMessages` のサポートを追加しました。	2024 年 7 月 3 日
AmazonSQSReadOnlyAccess マネージドポリシーで `ListQueueTags` アクションをサポート	AmazonSQSReadOnlyAccess マネージドポリシーは、指定した Amazon SQS キューに関連するすべてのタグを取得するための `ListQueueTags` をサポートするようになりました。	2024 年 5 月 2 日
AWS JSON プロトコル	JSON プロトコルを使用して API AWS リクエストを行います。	2023 年 7 月 27 日

Amazon SQS の AWS 管理ポリシーとこれらのポリシーの更新について説明する新しいセクション	Amazon SQS には、特定のソースキューにある最新のメッセージ移動タスク (最大 10 個) を一覧表示できる新しいアクションが追加されました。このアクションは `ListMessageMoveTasks` API 演算に関連付けられています。	2023 年 6 月 7 日
API を使用したデッドレターキューの再処理	Amazon SQS API を使用してデッドレターキューの再処理を設定します。	2023 年 6 月 7 日
Amazon SQS 用 ABAC	柔軟でスケーラブルなアクセス許可が行える、キューのタグを使用した属性ベースのアクセス制御 (ABAC)。	2022 年 11 月 10 日
FIFO の高スループット上限の引き上げ	商業リージョンの FIFO 高スループットモードのデフォルトクォータの引き上げと、FIFO 高スループットドキュメントの最適化を行いました。	2022 年 10 月 20 日
デフォルトのサーバー側の暗号化が利用可能	デフォルトで SQS 所有の暗号化 (SSE-SQS) を使用するサーバー側の暗号化。	2022 年 9 月 26 日
Amazon SQS の混乱した代理の保護サポートが利用可能	混乱した代理の保護を使用すると、リクエストに新しいヘッダーを指定できます。そのヘッダーは、Amazon SQS マネージド SSE を使用する際に KMS ポリシーの条件と照合されます。	2021 年 12 月 29 日

マネージド SSE が利用可能	Amazon SQS マネージド SSE (SSE-SQS)は、Amazon SQS 所有の暗号化キーを使用して、メッセージキューを介して送信される機密データを保護する、マネージドサーバー側の暗号化です。	2021 年 11 月 23 日
デッドレターキューの再処理が利用可能に	Amazon SQS は、標準キューに対してデッドレターキューの再処理をサポートしています。	2021 年 11 月 10 日
FIFOキューのメッセージに対する高スループットが利用可能です。	Amazon SQS FIFOキューに対する高スループットは、FIFO キューの1 秒あたりのトランザクション(TPS)数を増加することができます。スループットクォータの詳細については、「メッセージに関連するクォータ」を参照してください。	2021年5月27日
FIFO キューのメッセージの高スループットがプレビューリリースで利用可能に	Amazon SQS FIFOキューに対する高スループットはプレビューリリースであり、変更される可能性があります。この機能は、FIFO キューのメッセージに対して、1 秒あたりのトランザクション (TPS) の数を増加することができます。スループットクォータの詳細については、「メッセージに関連するクォータ」を参照してください。	2020年12月17日

新Amazon SQSコンソールのデザイン	開発および本番ワークフローを簡素化するために、Amazon SQS コンソールには新しいユーザーエクスペリエンスがあります。	2020年7月8日
Amazon SQSは、listキューおよび listDeadLetterSourceキューに対するページ割りをサポートします。	listキューまたはlistDeadLetterSourceキューリクエストで返される結果の最大数を指定できます。	2020年6月22日
Amazon SQS は、 GovCloud (米国) AWS リージョンを除くすべてのリージョンで 1 分間の AWS Amazon CloudWatch メトリクスをサポートします。	Amazon SQS に対する1分間CloudWatchメトリクスは、AWS GovCloud (US) リージョンを除き、すべてのリージョンで利用可能になりました。	2020年1月9日
Amazon SQSは1分間Amazon CloudWatchメトリクスをサポートします。	Amazon SQSに対する1分間CloudWatchメトリクスは現在、米国東部 (オハイオ)、欧州(アイルランド)、欧州 (ストックホルム) 、アジアパシフィック(東京)の各リージョンでのみ利用可能です。	2019年11月25日
AWS Lambda Amazon SQS FIFO キューの トリガーが利用可能に	FIFO キューに到着するメッセージを Lambda 関数トリガーとして設定できます。	2019年11月25日
Amazon SQS のサーバー側の暗号化 (SSE)が、中国リージョンで利用可能に	Amazon SQSに対するSSEは、中国リージョンで利用可能です。	2019年11月13日
FIFOキューは、中東 (バーレーン)リージョンで利用可能です。	FIFOキューは、中東 (バーレーン)リージョンで利用可能です。	2019年10月10日

Amazon SQS 用の Amazon Virtual Private Cloud (Amazon VPC) エンドポイントは、AWS GovCloud (米国東部) および AWS GovCloud (米国西部) リージョンで利用できます。	AWS GovCloud (米国東部) および GovCloud AWS GovCloud (米国西部) リージョンの Amazon VPC から Amazon SQS キューにメッセージを送信できます。	2019年9月5日
Amazon SQS では、メッセージシステム属性 AWS X-Ray を使用してキューのトラブルシューティングを行うことができます。	X-Rayを使用して、Amazon SQS キュー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。このリリースでは、`MessageSystemAttribute` リクエストパラメータに`SendMessage` と`SendMessageBatch` APIオペレーション、`ReceiveMessage` APIオペレーションの`AWSTraceHeader` 属性と`MessageSystemAttributeValue` データタイプが（ Amazon SQS)を通してX-Rayトレースヘッダーへ送信できる）が追加されます。	2019年8月28日

Amazon SQSキュー作成時にタグ付けが可能になります。	単一の Amazon SQS API コール、 AWS SDK 関数、または AWS Command Line Interface （ AWS CLI) コマンドを使用して、キューを同時に作成し、タグを指定できます。さらに、Amazon SQS は `aws:TagKeys` および `aws:RequestTag` AWS Identity and Access Management (IAM) キーをサポートしています。	2019年8月22日
Amazon SQS に対する一時キュークライアントは、現在利用可能です。	一時キューは、リクエストとレスポンスのような一般的なメッセージパターンを使用する場合に、開発時間と開発コストを削減するのに役立ちます。一時キュークライアントを使用すると、高スループットで費用対効果の高いアプリケーションマネージドの一時キューを作成ができます。	2019年7月25日
SSE for Amazon SQS が AWS GovCloud (米国東部) リージョンで利用可能に	Amazon SQS のサーバー側の暗号化 (SSE) は、 AWS GovCloud (米国東部) リージョンで利用できます。	2019 年 6 月 20 日
FIFO キューは、アジアパシフィック (香港) 、中国 (北京) 、 AWS GovCloud (米国東部) 、 AWS GovCloud (米国西部) の各リージョンで利用できます。	FIFO キューは、アジアパシフィック (香港) 、中国 (北京) 、 AWS GovCloud (米国東部) 、 AWS GovCloud (米国西部) の各リージョンで利用できます。	2019年5月15日

Amazon SQSに対するAmazon VPC エンドポイントポリシーは利用可能です。	Amazon SQSに対するAmazon VPC エンドポイントポリシーを作成できます。	2019年4月4日
FIFOキューは、欧州(ストックホルム) および中国 (寧夏) リージョンで利用可能です。	FIFOキューは、欧州(ストックホルム) および中国 (寧夏) リージョンで利用可能です。	2019年3月14日
FIFO キューは、Amazon SQS が利用可能なすべてのリージョンで利用可能です。	FIFOキューは、米国東部（オハイオ）、米国東部（北バージニア）、米国西部（北カリフォルニア）、米国西部（オレゴン）、アジアパシフィック（ムンバイ）、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（パリ）、および南米国（サンパウロ）の各リージョンで利用可能です。	2019年2月7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。