Developer Guide

Amazon Simple Queue Service

Amazon Simple Queue Service: Developer Guide

Table of Contents

What is Amazon Simple Queue Service

Amazon Simple Queue Service (Amazon SQS) offers a secure, durable, and available hosted queue that lets you integrate and decouple distributed software systems and components. Amazon SQS offers common constructs such as dead-letter queues and cost allocation tags. It provides a generic web services API that you can access using any programming language that the AWS SDK supports.

Topics

Benefits of using Amazon SQS

- **Security** – You control who can send messages to and receive messages from an Amazon SQS queue. You can choose to transmit sensitive data by protecting the contents of messages in queues by using default Amazon SQS managed server-side encryption (SSE), or by using custom SSE keys managed in AWS Key Management Service (AWS KMS).
- **Durability** – For the safety of your messages, Amazon SQS stores them on multiple servers. Standard queues support at-least-once message delivery, and FIFO queues support exactly-once message processing and high-throughput mode.
- **Availability** – Amazon SQS uses redundant infrastructure to provide highly-concurrent access to messages and high availability for producing and consuming messages.
- **Scalability** – Amazon SQS can process each buffered request independently, scaling transparently to handle any load increases or spikes without any provisioning instructions.
- **Reliability** – Amazon SQS locks your messages during processing, so that multiple producers can send and multiple consumers can receive messages at the same time.
- **Customization** – Your queues don't have to be exactly alike—for example, you can set a default delay on a queue. You can store the contents of messages larger than 256 KB using Amazon Simple Storage Service (Amazon S3) or Amazon DynamoDB, with Amazon SQS holding a pointer to the Amazon S3 object, or you can split a large message into smaller messages.

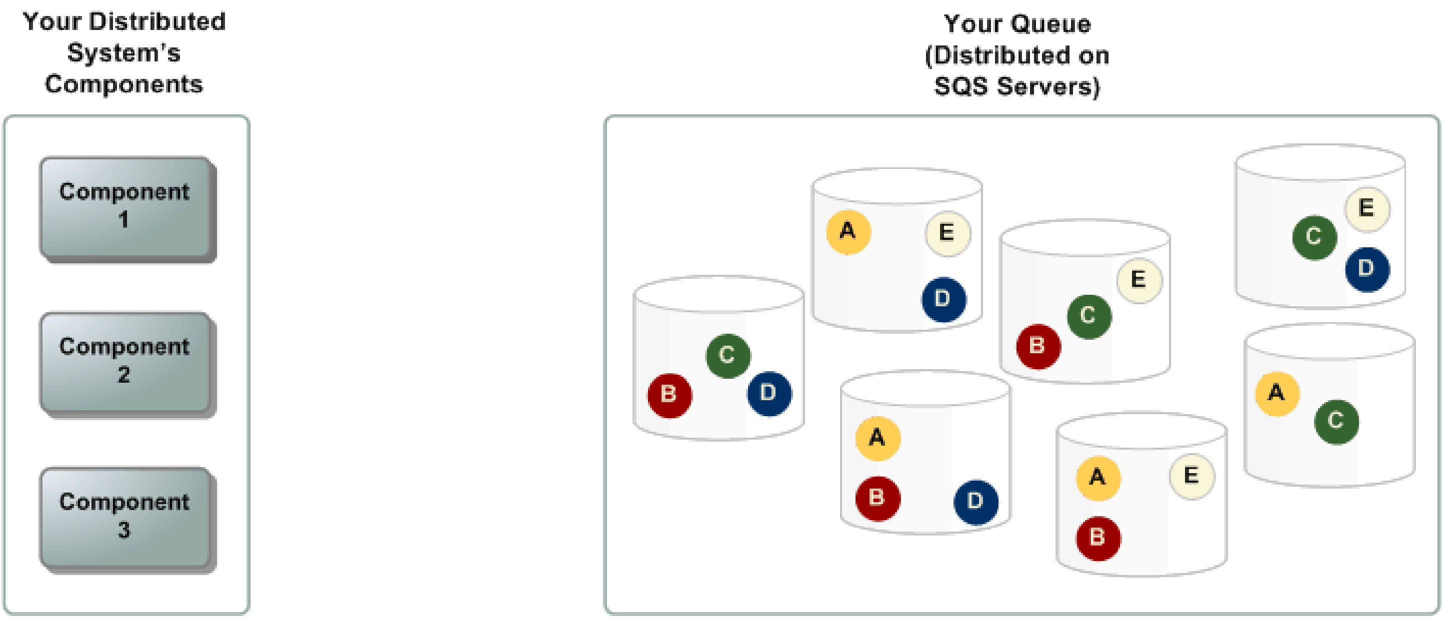
Basic Amazon SQS architecture

This section outlines the parts of a distributed messaging system and explains the lifecycle of an Amazon SQS message.

Distributed queues

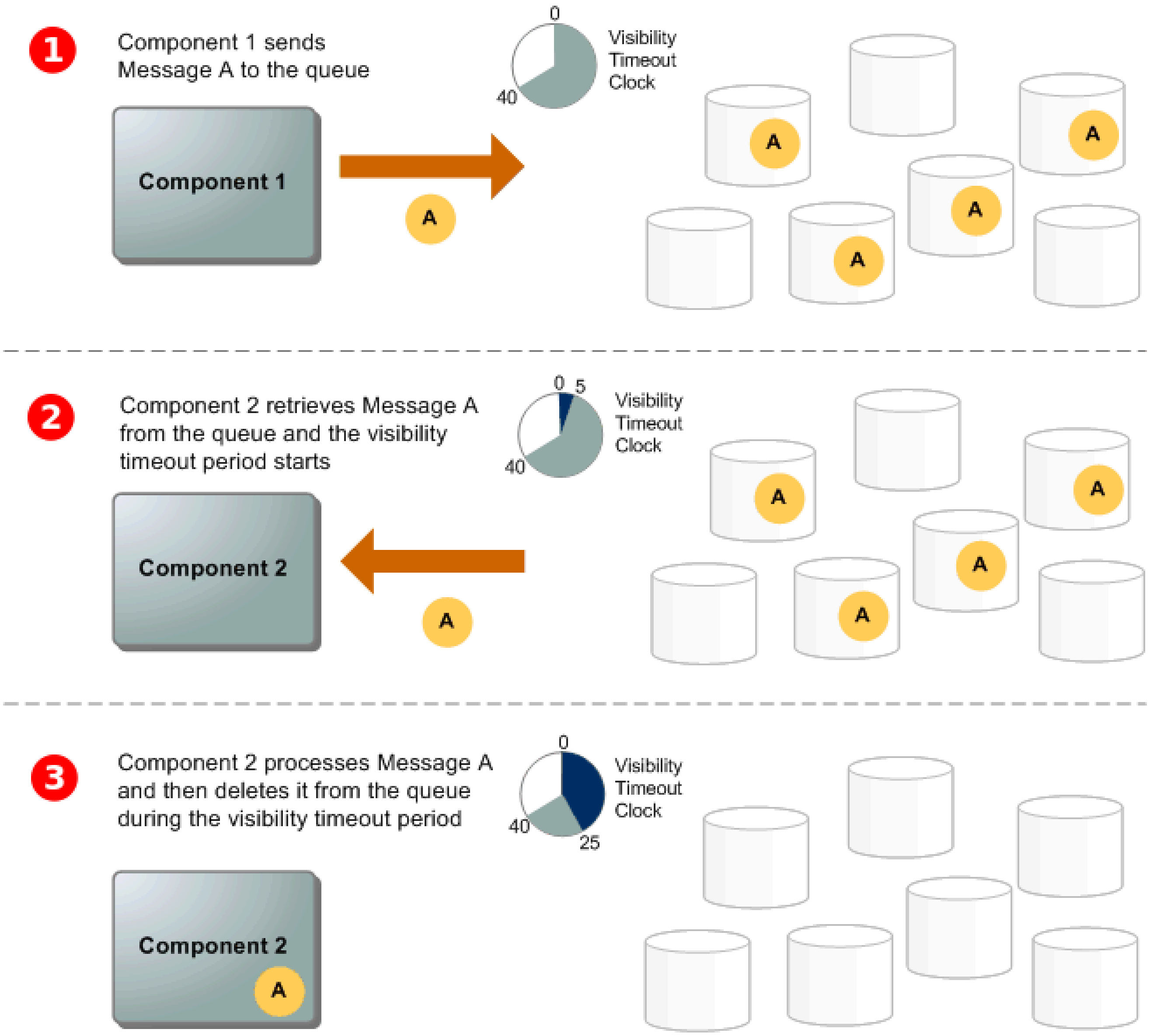
There are three main parts in a distributed messaging system: the components of your distributed system, your queue (distributed on Amazon SQS servers), and the messages in the queue.

In the following scenario, your system has several *producers* (components that send messages to the queue) and *consumers* (components that receive messages from the queue). The queue (which holds messages A through E) redundantly stores the messages across multiple Amazon SQS servers.



Message lifecycle

The following scenario describes the lifecycle of an Amazon SQS message in a queue, from creation to deletion.



1

A producer (component 1) sends message A to a queue, and the message is distributed across the Amazon SQS servers redundantly.

2

When a consumer (component 2) is ready to process messages, it consumes messages from the queue, and message A is returned. While message A is being processed, it remains in the queue and isn't returned to subsequent receive requests for the duration of the visibility timeout.

3

The consumer (component 2) deletes message A from the queue to prevent the message from being received and processed again when the visibility timeout expires.

Note

Amazon SQS automatically deletes messages that have been in a queue for more than the maximum message retention period. The default message retention period is 4 days. However, you can set the message retention period to a value from 60 seconds to 1,209,600 seconds (14 days) using the `SetQueueAttributes` action.

Differences between Amazon SQS, Amazon MQ, and Amazon SNS

Amazon SQS, Amazon SNS, and Amazon MQ offer highly scalable and easy-to-use managed messaging services, each designed for specific roles within distributed systems. Here's an enhanced overview of the differences between these services:

Amazon SQS decouples and scales distributed software systems and components as a queue service. It processes messages through a single subscriber typically, ideal for workflows where order and loss prevention are critical. For wider distribution, integrating Amazon SQS with Amazon SNS enables a fanout messaging pattern, effectively pushing messages to multiple subscribers at once.

Amazon SNS allows publishers to send messages to multiple subscribers through topics, which serve as communication channels. Subscribers receive published messages using a supported endpoint type, such as Amazon Data Firehose, Amazon SQS, Lambda, HTTP, email, mobile push notifications, and mobile text messages (SMS). This service is ideal for scenarios requiring immediate notifications, such as real-time user engagement or alarm systems. To prevent message loss when subscribers are offline, integrating Amazon SNS with Amazon SQS queue messages ensures consistent delivery.

Amazon MQ fits best with enterprises looking to migrate from traditional message brokers, supporting standard messaging protocols like AMQP and MQTT, along with Apache ActiveMQ and RabbitMQ. It offers compatibility with legacy systems needing stable, reliable messaging without significant reconfiguration.

The following chart provides an overview of each services' resource type:

Resource type	Amazon SNS	Amazon SQS	Amazon MQ
Synchronous	No	No	Yes
Asynchronous	Yes	Yes	Yes
Queues	No	Yes	Yes
Publisher-subscriber messaging	Yes	No	Yes
Message brokers	No	No	Yes

Both Amazon SQS and Amazon SNS are recommended for new applications that can benefit from nearly unlimited scalability and simple APIs. They generally offer more cost-effective solutions for high-volume applications with their pay-as-you-go pricing. We recommend Amazon MQ for migrating applications from existing message brokers that rely on compatibility with APIs such as JMS or protocols such as Advanced Message Queuing Protocol (AMQP), MQTT, OpenWire, and Simple Text Oriented Message Protocol (STOMP).

Setting up Amazon SQS

Before you can use Amazon SQS for the first time, you must complete the following steps.

Topics

Step 1: Create an AWS account and IAM user

To access any AWS service, you first need to create an AWS account, an Amazon.com account that can use AWS products. You can use your AWS account to view your activity and usage reports and to manage authentication and access.

To avoid using your AWS account root user for Amazon SQS actions, it is a best practice to create an IAM user for each person who needs administrative access to Amazon SQS.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Step 2: Grant programmatic access

To use Amazon SQS actions (for example, using Java or through the AWS Command Line Interface), you need an access key ID and a secret access key.

> **Note**
>
> The access key ID and secret access key are specific to AWS Identity and Access Management. Don't confuse them with credentials for other AWS services, such as Amazon EC2 key pairs.

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
Workforce identity	Use temporary credentials to sign programmatic requests	Following the instructions for the interface that you want to use.

Which user needs programmatic access?	To	By
(Users managed in IAM Identity Center)	to the AWS CLI, AWS SDKs, or AWS APIs.	• For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Step 3: Get ready to use the example code

This guide includes examples that use the AWS SDK for Java. To run the example code, follow the set-up instructions in Getting Started with AWS SDK for Java 2.0.

You can develop AWS applications in other programming languages, such as Go, JavaScript, Python and Ruby. For more information, see Tools to Build on AWS.

> **Note**
>
> You can explore Amazon SQS without writing code with tools such as the AWS Command Line Interface (AWS CLI) or Windows PowerShell. You can find AWS CLI examples in the Amazon SQS section of the *AWS CLI Command Reference*. You can find Windows

PowerShell examples in the Amazon Simple Queue Service section of the *AWS Tools for PowerShell Cmdlet Reference*.

Next steps

You are now ready for Getting started with managing Amazon SQS queues and messages using the AWS Management Console.

Getting started with Amazon SQS

In this section, you'll learn how to create standard or FIFO queues using the Amazon SQS console.

Topics

- Prerequisites
- Understanding the Amazon SQS console
- Amazon SQS queue types
- Creating an Amazon SQS standard queue and sending a message
- Creating an Amazon SQS FIFO queue and sending a message
- Common tasks for getting started with Amazon SQS

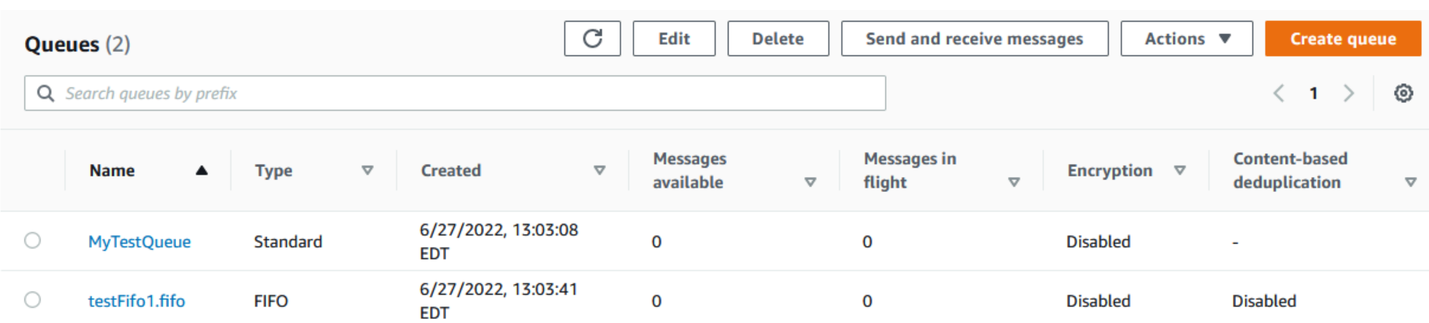
Prerequisites

Before you begin, complete the steps in Setting up Amazon SQS.

Understanding the Amazon SQS console

When you open the Amazon SQS console, choose **Queues** from the navigation pane. The **Queues** page provides information about all of your queues in the active region.

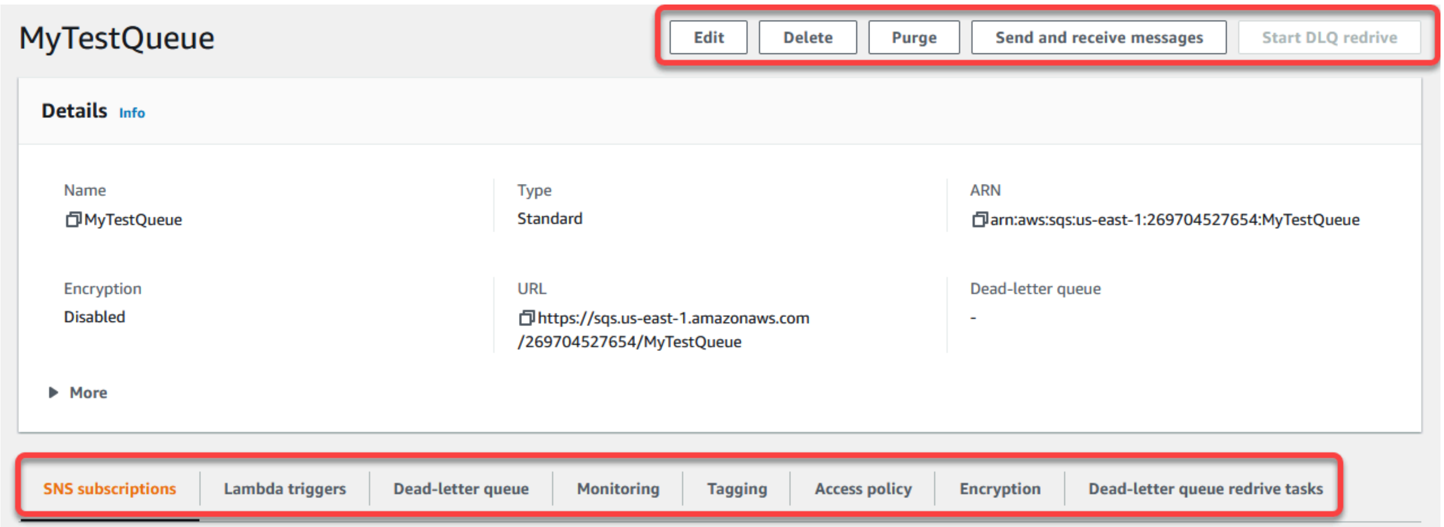
Each queue entry provides essential information about the queue, including its type and key attributes. Standard queues, optimized for maximum throughput and best-effort message ordering, are distinguished from First-In-First-Out (FIFO) queues, which prioritize message ordering and uniqueness for applications requiring strict message sequencing.



Interactive elements and actions

From the Queues page, you have multiple options for managing your queues:

1. **Quick Actions** – Adjacent to each queue name, a dropdown menu offers quick access to common actions such as sending messages, viewing or deleting messages, configuring triggers, and deleting the queue itself.
2. **Detailed View and Configuration** – Clicking on a queue name opens its Details page, where you can delve deeper into queue settings and configurations. Here, you can adjust parameters like message retention period, visibility timeout, and maximum message size to tailor the queue to your application's requirements.



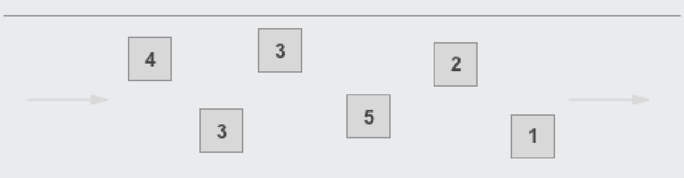
Region selection and resource tags

Ensure you're in the correct AWS Region to access and manage your queues effectively. Additionally, consider utilizing resource tags to organize and categorize your queues, enabling better resource management, cost allocation, and access control within your AWSshared environment.

By leveraging the features and functionalities offered within the Amazon SQS console, you can efficiently manage your messaging infrastructure, optimize queue performance, and ensure reliable message delivery for your applications.

Amazon SQS queue types

Amazon SQS supports two types of queues – **standard** queues and **FIFO** queues. Use the information from the following table to choose the right queue for your situation. To learn more about Amazon SQS queues, see [Amazon SQS standard queues](#) and [Amazon SQS FIFO queues](#).

Standard queues	FIFO queues
Unlimited Throughput – Standard queues support a nearly unlimited number of API calls per second, per API action (`SendMessage` , `ReceiveMessage` , or `DeleteMessage`). **At-Least-Once Delivery** – A message is delivered at least once, but occasionally more than one copy of a message is delivered. **Best-Effort Ordering** – Occasionally, messages are delivered in an order different from which they were sent.	**High Throughput** – If you use [batching](), FIFO queues support up to 3,000 messages per second, per API method (`SendMessageBatch` , `ReceiveMessage` , or `DeleteMessageBatch`). The 3,000 messages per second represent 300 API calls, each with a batch of 10 messages. To request a quota increase, [submit a support request](). Without batching, FIFO queues support up to 300 API calls per second, per API method (`SendMessage` , `ReceiveMessage` , or `DeleteMessage`). **Exactly-Once Processing** – A message is delivered once and remains available until a consumer processes and deletes it. Duplicates aren't introduced into the queue. **First-In-First-Out Delivery** – The order in which messages are sent and received is strictly preserved.
	
Send data between applications when the throughput is important, for example: • Decouple live user requests from intensive background work: let users upload media while resizing or encoding it.	Send data between applications when the order of events is important, for example: • Make sure that user-entered commands are run in the right order. • Display the correct product price by sending price modifications in the right order.

Standard queues	FIFO queues
• Allocate tasks to multiple worker nodes: process a high number of credit card validation requests. • Batch messages for future processing: schedule multiple entries to be added to a database.	• Prevent a student from enrolling in a course before registering for an account.

Implementing request-response systems in Amazon SQS

When implementing a request-response or remote procedure call (RPC) system, keep the following best practices in mind:

- **Create Reply Queues on Start-Up:** Instead of creating reply queues per message, create them on start-up, per producer. Use a correlation ID message attribute to map replies to requests efficiently.
- **Avoid Sharing Reply Queues Among Producers:** Ensure that each producer has its own reply queue. Sharing reply queues can result in a producer receiving response messages intended for another producer.

For more information about implementing the request-response pattern using the Temporary Queue Client, see Request-response messaging pattern (virtual queues).

Creating an Amazon SQS standard queue and sending a message

This is how to create a standard queue for Amazon SQS.

Create a queue using the Amazon SQS console

You can use the Amazon SQS console to create standard queues. The console provides default values for all settings except for the queue name.

Important

On August 17, 2022, default server-side encryption (SSE) was applied to all Amazon SQS queues.

Do not add personally identifiable information (PII) or other confidential or sensitive information in queue names. Queue names are accessible to many Amazon Web Services, including billing and CloudWatch logs. Queue names are not intended to be used for private or sensitive data.

To create an Amazon SQS standard queue

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Choose **Create queue**.
3. For **Type**, the **Standard** queue type is set by default.

   Note

   You can't change the queue type after you create the queue.
4. Enter a **Name** for your queue.
5. (Optional) The console sets default values for the queue configuration parameters. Under **Configuration**, you can set new values for the following parameters:
   a. For **Visibility timeout** , enter the duration and units. The range is from 0 seconds to 12 hours. The default value is 30 seconds.
   b. For **Message retention period**, enter the duration and units. The range is from 1 minute to 14 days. The default value is 4 days.
   c. For **Delivery delay**, enter the duration and units. The range is from 0 seconds to 15 minutes. The default value is 0 seconds.
   d. For **Maximum message size**, enter a value. The range is from 1 KB to 256 KB. The default value is 256 KB.
   e. For **Receive message wait time**, enter a value. The range is from 0 to 20 seconds. The default value is 0 seconds, which sets short polling. Any non-zero value sets long polling.
6. (Optional) Define an **Access policy**. The access policy defines the accounts, users, and roles that can access the queue. The access policy also defines the actions (such as `SendMessage`,

`ReceiveMessage`, or `DeleteMessage`) that the users can access. The default policy allows only the queue owner to send and receive messages.

To define the access policy, do one of the following:

- Choose **Basic** to configure who can send messages to the queue and who can receive messages from the queue. The console creates the policy based on your choices and displays the resulting access policy in the read-only JSON panel.
- Choose **Advanced** to modify the JSON access policy directly. This allows you to specify a custom set of actions that each principal (account, user, or role) can perform.

7. For **Redrive allow policy**, choose **Enabled**. Select one of the following: **Allow all**, **By queue**, or **Deny all**. When choosing **By queue**, specify a list of up to 10 source queues by the Amazon Resource Name (ARN).
8. Amazon SQS provides managed server-side encryption by default. To choose an encryption key type, or to disable Amazon SQS managed server-side encryption, expand **Encryption**. For more on encryption key types, see Configuring server-side encryption for a queue using SQS-managed encryption keys and Configuring server-side encryption for a queue using the Amazon SQS console.

> **Note**
>
> With SSE enabled, anonymous `SendMessage` and `ReceiveMessage` requests to the encrypted queue will be rejected. Amazon SQS security best practises recommend against using anonymous requests. If you wish to send anonymous requests to an Amazon SQS queue, make sure to disable SSE.

9. (Optional) To configure a dead-letter queue to receive undeliverable messages, expand **Dead-letter queue**.
10. (Optional) To add tags to the queue, expand **Tags**.
11. Choose **Create queue**. Amazon SQS creates the queue and displays the queue's **Details** page.

Amazon SQS propagates information about the new queue across the system. Because Amazon SQS is a distributed system, you might experience a slight delay before the console displays the queue on the **Queues** page.

Send a message

After you create your queue, you can send a message to it.

1. From the left navigation pane, choose **Queues**. From the queue list, select the queue that you created.
2. From **Actions**, choose **Send and receive messages**.

   The console displays the **Send and receive messages** page.
3. In the **Message body**, enter the message text.
4. For a standard queue, you can enter a value for **Delivery delay** and choose the units. For example, enter `60` and choose **seconds**. For more information, see Amazon SQS message timers.
5. Choose **Send message**.

   When your message is sent, the console displays a success message. Choose **View details** to display information about the sent message.

Creating an Amazon SQS FIFO queue and sending a message

This is how to create a FIFO queue for Amazon SQS.

Create a queue

You can use the Amazon SQS console to create FIFO queues. The console provides default values for all settings except for the queue name.

> **Important**
>
> On August 17, 2022, default server-side encryption (SSE) was applied to all Amazon SQS queues.
> Do not add personally identifiable information (PII) or other confidential or sensitive information in queue names. Queue names are accessible to many Amazon Web Services, including billing and CloudWatch logs. Queue names are not intended to be used for private or sensitive data.

To create an Amazon SQS FIFO queue

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Choose **Create queue**.
3. For **Type**, the **Standard** queue type is set by default. To create a FIFO queue, choose **FIFO**.

   > **Note**
   >
   > You can't change the queue type after you create the queue.

4. Enter a **Name** for your queue.

   The name of a FIFO queue must end with the `.fifo` suffix. The suffix counts towards the 80-character queue name quota. To determine whether a queue is FIFO, you can check whether the queue name ends with the suffix.

5. (Optional) The console sets default values for the queue configuration parameters. Under **Configuration**, you can set new values for the following parameters:

   a. For **Visibility timeout** , enter the duration and units. The range is from 0 seconds to 12 hours. The default value is 30 seconds.

   b. For **Message retention period**, enter the duration and units. The range is from 1 minute to 14 days. The default value is 4 days.

   c. For **Delivery delay**, enter the duration and units. The range is from 0 seconds to 15 minutes. The default value is 0 seconds.

   d. For **Maximum message size**, enter a value. The range is from 1 KB to 256 KB. The default value is 256 KB.

   e. For **Receive message wait time**, enter a value. The range is from 0 to 20 seconds. The default value is 0 seconds, which sets short polling. Any non-zero value sets long polling.

   f. For a FIFO queue, choose **Content-based deduplication** to enable content-based deduplication. The default setting is disabled.

   g. (Optional) For a FIFO queue to enable higher throughput for sending and receiving messages in the queue, choose **Enable high throughput FIFO**.

      Choosing this option changes the related options (**Deduplication scope** and **FIFO throughput limit**) to the required settings for enabling high throughput for FIFO queues. If you change any of the settings required for using high throughput FIFO, normal throughput is in effect for the queue, and deduplication occurs as specified. For more information, see High throughput for FIFO queues in Amazon SQS and Amazon SQS message quotas.

6. (Optional) Define an **Access policy**. The access policy defines the accounts, users, and roles that can access the queue. The access policy also defines the actions (such as `SendMessage`, `ReceiveMessage`, or `DeleteMessage`) that the users can access. The default policy allows only the queue owner to send and receive messages.

   To define the access policy, do one of the following:

   - Choose **Basic** to configure who can send messages to the queue and who can receive messages from the queue. The console creates the policy based on your choices and displays the resulting access policy in the read-only JSON panel.
   - Choose **Advanced** to modify the JSON access policy directly. This allows you to specify a custom set of actions that each principal (account, user, or role) can perform.

7. For **Redrive allow policy**, choose **Enabled**. Select one of the following: **Allow all**, **By queue**, or **Deny all**. When choosing **By queue**, specify a list of up to 10 source queues by the Amazon Resource Name (ARN).

8. Amazon SQS provides managed server-side encryption by default. To choose an encryption key type, or to disable Amazon SQS managed server-side encryption, expand **Encryption**. For more on encryption key types, see Configuring server-side encryption for a queue using SQS-managed encryption keys and Configuring server-side encryption for a queue using the Amazon SQS console.

   > **Note**
   >
   > With SSE enabled, anonymous `SendMessage` and `ReceiveMessage` requests to the encrypted queue will be rejected. Amazon SQS security best practises recommend against using anonymous requests. If you wish to send anonymous requests to an Amazon SQS queue, make sure to disable SSE.

9. (Optional) To configure a dead-letter queue to receive undeliverable messages, expand **Dead-letter queue**.

10. (Optional) To add tags to the queue, expand **Tags**.

11. Choose **Create queue**. Amazon SQS creates the queue and displays the queue's **Details** page.

Amazon SQS propagates information about the new queue across the system. Because Amazon SQS is a distributed system, you might experience a slight delay before the console displays the queue on the **Queues** page.

After creating a queue, you can send messages to it, and receive and delete messages. You can also edit any of the queue configuration settings except the queue type.

Send a message

After you create your queue, you can send a message to it.

1. From the left navigation pane, choose **Queues**. From the queue list, select the queue that you created.
2. From **Actions**, choose **Send and receive messages**.

   The console displays the **Send and receive messages** page.
3. In the **Message body**, enter the message text.
4. For a First-In-First-Out (FIFO) queue, enter a **Message group ID**. For more information, see FIFO queue delivery logic in Amazon SQS.
5. (Optional) For a FIFO queue, you can enter a **Message deduplication ID**. If you enabled content-based deduplication for the queue, the message deduplication ID isn't required. For more information, see FIFO queue delivery logic in Amazon SQS.
6. FIFO queues does not support timers on individual messages. For more information, see Amazon SQS message timers.
7. Choose **Send message**.

   When your message is sent, the console displays a success message. Choose **View details** to display information about the sent message.

Common tasks for getting started with Amazon SQS

Now that you've created a queue and learned how to send, receive, and delete messages and how to delete a queue, you might want to try the following:

- To trigger a Lambda function, see Configuring an Amazon SQS queue to trigger an AWS Lambda function.
- Learn how to configure queues, including SSE and other features.

- Learn how to send a message with attributes.
- Learn how to send a message from a VPC.
- To discover the functionality and architecture of Amazon SQS, see Amazon SQS queue types and Basic Amazon SQS architecture.
- To find out the guidelines and caveats that will help you make the most of Amazon SQS, see Amazon SQS best practices.
- Explore the Amazon SQS examples for one of the AWS SDKs, such as the AWS SDK for Java 2.x Developer Guide.
- To learn about Amazon SQS AWS CLI commands, see the *AWS CLI Command Reference*.
- To learn about Amazon SQS actions, see the *Amazon Simple Queue Service API Reference*.
- Learn how to interact with Amazon SQS programmatically: Read Working with APIs and explore the AWS Development Center:
  - Java
  - JavaScript
  - PHP
  - Python
  - Ruby
  - Windows & .NET

- Learn about keeping an eye on costs and resources in the Troubleshooting issues in Amazon SQS section.
- Learn about protecting your data and access to it in the Security section.
- Learn more about the Amazon SQS workflow in the Amazon SQS access control process workflow section.

Managing an Amazon SQS queue

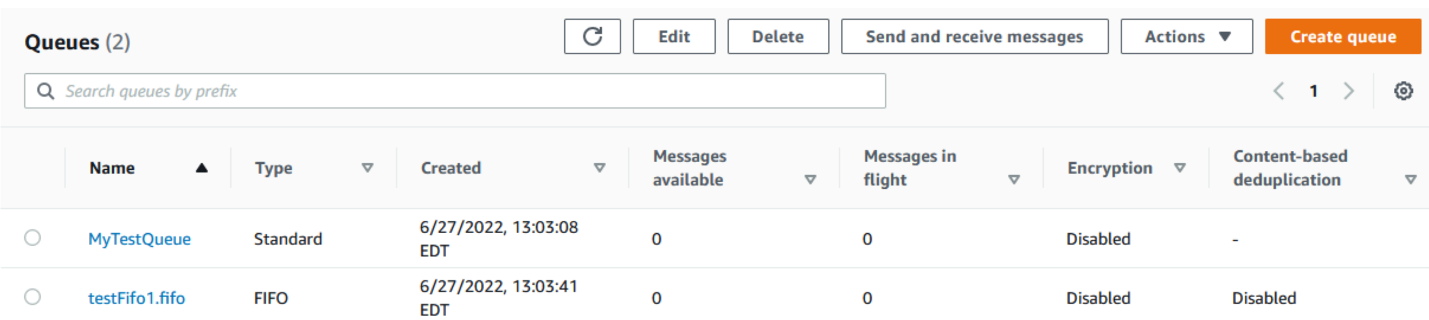
This section helps you become more familiar with Amazon SQS by showing you how to manage queues and messages using the Amazon SQS console.

Prerequisites

Before you begin, complete the steps in Setting up Amazon SQS.

Understanding the Amazon SQS console

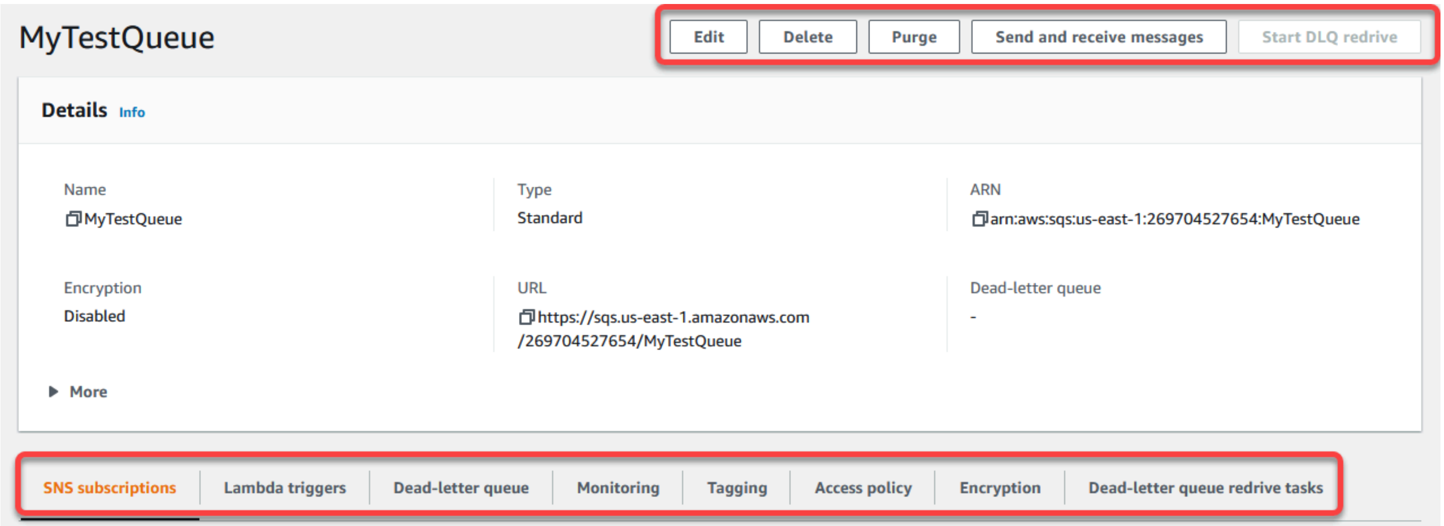
When you open the console, choose **Queues** from the navigation pane to display the **Queues** page. The **Queues** page provides information about all of your queues in the active region.



The entry for each queue shows the queue type and other information about the queue. The **Type** column helps you distinguish standard queues from First-In-First Out (FIFO) queues at a glance.

From the **Queues** page, there are two ways to perform actions on a queue. You can choose the option next to the queue name and then choose the action you want to perform on the queue.

You can also choose the queue name, which open the **Details** page for the queue. The **Details** page includes the same actions as the **Queues** page. In addition, you can choose one of the tabs below the **Details** section to view additional configuration details and actions.



Editing an Amazon SQS queue using the console

You can use the Amazon SQS console to edit any queue configuration parameters (except the queue type) and add or remove queue features.

To edit an Amazon SQS queue (console)

1. Open the Queues page of the Amazon SQS console.
2. Select a queue, and then choose **Edit**.
3. (Optional) Under **Configuration**, update the queue's configuration parameters.
4. (Optional) To update the access policy, under **Access policy**, modify the JSON policy.
5. (Optional) To update a dead-letter queue redrive allow policy, expand **Redrive allow policy**.
6. (Optional) To update or remove encryption, expand **Encryption**.
7. (Optional) To add, update, or remove a dead-letter queue (which allows you to receive undeliverable messages), expand **Dead-letter queue**.
8. (Optional) To add, update, or remove the tags for the queue, expand **Tags**.
9. Choose **Save**.

   The console displays the **Details** page for the queue.

Receiving and deleting a message in Amazon SQS

After you send messages to an Amazon SQS queue, you have the option to receive and delete them. When requesting messages from a queue, you cannot specify individual messages. Instead, you determine the maximum number of messages you wish to retrieve, up to a limit of 10.

Amazon SQS operates as a distributed system, which can occasionally result in an empty response when retrieving messages from a queue with few messages. If this happens, simply rerun your request. To optimize message retrieval and minimize empty responses, consider using long polling. Long polling delays the response until a message becomes available or the poll times out, reducing unnecessary polling costs and improving efficiency.

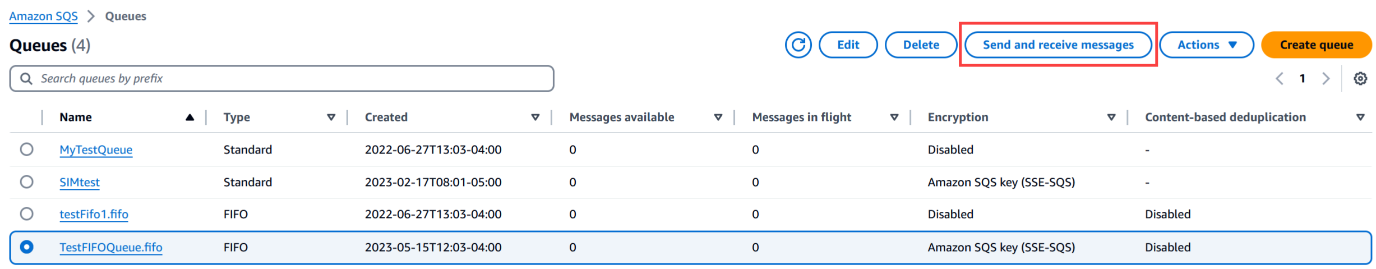
Messages are not automatically deleted after retrieval because Amazon SQS ensures that you do not lose access to a message due to processing failures, such as issues with your application or network disruptions. To permanently remove a message from the queue, you must explicitly send a delete request after processing the message to confirm successful receipt and handling.

When messages are retrieved via the Amazon SQS console, they are immediately made visible again for re-retrieval. This default behavior ensures messages are not inadvertently lost during manual operations but can lead to repeated processing. In automated environments, adjust the visibility timeout setting to control how long a message remains invisible to other consumers after being retrieved. This setting is crucial for coordinating message processing across multiple consumers and ensuring that messages are processed only once.

For more detailed operations on receiving and deleting messages, see the Amazon SQS API Reference Guide. This guide offers comprehensive information on API endpoints, including parameters that manage complex message handling scenarios effectively.

To receive and delete a message using the console

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. On the Queues page, select a **queue**, and then choose **Send and receive messages**.



4. On the Send and receive messages page, choose **Poll for messages**.

   Amazon SQS begins to poll for messages in the queue. The progress bar on the right side of the Receive messages section displays the duration of polling.

   The Messages section displays a list of the received messages. For each message, the list displays the message ID, Sent date, Size, and Receive count.

5. To delete messages, choose the **messages** that you want to delete, and then choose **Delete**.
6. In the Delete Messages dialog box, choose **Delete**.

Confirming that an Amazon SQS queue is empty

In most cases, you can use [long polling](#) to determine if a queue is empty. In rare cases, you might receive empty responses even when a queue still contains messages, especially if you specified a low value for **Receive message wait time** when you created the queue. This section describes how to confirm that a queue is empty.

To confirm that a queue is empty (console)

1. Stop all producers from sending messages.
2. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
3. In the navigation pane, choose **Queues**.
4. On the **Queues** page, choose a queue.
5. Choose the **Monitoring** tab.
6. At the top right of the Monitoring dashboards, choose the down arrow next to the Refresh symbol. From the dropdown menu, choose **Auto refresh**. Leave the **Refresh interval** at **1 Minute**.
7. Observe the following dashboards:
   - Approximate Number Of Messages Delayed

- Approximate Number Of Messages Not Visible
- Approximate Number Of Messages Visible

When all of them show `0` values for several minutes, the queue is empty.

To confirm that a queue is empty (AWS CLI, AWS API)

1. Stop all producers from sending messages.
2. Repeatedly run one of the following commands:
   - AWS CLI: `get-queue-attributes`
   - AWS API: `GetQueueAttributes`
3. Observe the metrics for the following attributes:
   - `ApproximateNumberOfMessagesDelayed`
   - `ApproximateNumberOfMessagesNotVisible`
   - `ApproximateNumberOfMessagesVisible`

   When all of them are `0` for several minutes, the queue is empty.

If you rely on Amazon CloudWatch metrics, make sure that you see multiple consecutive zero data points before considering that queue empty. For more information on CloudWatch metrics, see Available CloudWatch metrics for Amazon SQS.

Deleting an Amazon SQS queue

If you no longer use an Amazon SQS queue and don't foresee using it in the near future, we recommend deleting it.

> **Tip**
>
> If you want to verify that a queue is empty before you delete it, see Confirming that an Amazon SQS queue is empty.

You can delete a queue even when it isn't empty. To delete the messages in a queue but not the queue itself, purge the queue.

To delete a queue (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. On the **Queues** page, choose the queue to delete.
4. Choose **Delete**.
5. In the **Delete queue** dialog box, confirm the deletion by entering **`delete`**.
6. Choose **Delete**.

To delete a queue (AWS CLI and API)

You can use one of the following commands to delete a queue:

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

Purging messages from an queue using the Amazon SQS console

If you don't want to delete an Amazon SQS queue but need to delete all of the messages from it, purge the queue. The message deletion process takes up to 60 seconds. We recommend waiting for 60 seconds regardless of your queue's size.

> **Important**
>
> When you purge a queue, you can't retrieve any of the deleted messages.

To purge a queue (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. On the **Queues** page, choose the queue to purge.

4. From **Actions**, choose **Purge**.
5. In the **Purge queue** dialog box, confirm the purge by entering **`purge`** and choosing **Purge**.

   All messages are purged from the queue. The console displays a confirmation banner.

Amazon SQS standard queues

Amazon SQS provides standard queues as the default queue type, supporting a nearly unlimited number of API calls per second for actions like `SendMessage`, `ReceiveMessage`, and `DeleteMessage`. Standard queues ensure at-least-once message delivery, but due to the highly distributed architecture, more than one copy of a message might be delivered, and messages may occasionally arrive out of order. Despite this, standard queues make a best-effort attempt to maintain the order in which messages are sent.

When you send a message using `SendMessage`, Amazon SQS redundantly stores the message in multiple availability zones (AZs) before acknowledging it. This redundancy ensures that no single computer, network, or AZ failure can render the messages inaccessible.

You can create and configure queues using the Amazon SQS console. For detailed instructions, see Create a queue using the Amazon SQS console. For Java-specific examples, see Amazon SQS Java SDK examples.

Use cases for standard queues

Standard message queues are suitable for various scenarios, as long as your application can handle messages that might arrive more than once or out of order. Examples include:

- **Decoupling live user requests from intensive background work** – Users can upload media while the system resizes or encodes it in the background.
- **Allocating tasks to multiple worker nodes** – For example, handling a high volume of credit card validation requests.
- **Batching messages for future processing** – Scheduling multiple entries to be added to a database at a later time.

For information on quotas related to standard queues, see Amazon SQS standard queue quotas.

For best practices of working with standard queues, see Amazon SQS best practices.

Amazon SQS at-least-once delivery

Amazon SQS stores copies of your messages on multiple servers for redundancy and high availability. On rare occasions, one of the servers that stores a copy of a message might be unavailable when you receive or delete a message.

If this occurs, the copy of the message isn't deleted on the server that is unavailable, and you might get that message copy again when you receive messages. Design your applications to be *idempotent* (they should not be affected adversely when processing the same message more than once).

Amazon SQS queue and message identifiers

This section describes the identifiers of standard and FIFO queues. These identifiers can help you find and manipulate specific queues and messages.

Identifiers for Amazon SQS standard queues

For more information about the following identifiers, see the [*Amazon Simple Queue Service API Reference*].

Queue name and URL

When you create a new queue, you must specify a queue name unique for your AWS account and region. Amazon SQS assigns each queue you create an identifier called a *queue URL* that includes the queue name and other Amazon SQS components. Whenever you want to perform an action on a queue, you provide its queue URL.

The following is the queue URL for a queue named `MyQueue` owned by a user with the AWS account number `123456789012`.

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

You can retrieve the URL of a queue programmatically by listing your queues and parsing the string that follows the account number. For more information, see `ListQueues`.

Message ID

Each message receives a system-assigned *message ID* that Amazon SQS returns to you in the `SendMessage` response. This identifier is useful for identifying messages. The maximum length of a message ID is 100 characters.

Receipt handle

Every time you receive a message from a queue, you receive a *receipt handle* for that message. This handle is associated with the action of receiving the message, not with the message itself. To delete the message or to change the message visibility, you must provide the receipt handle (not the message ID). Thus, you must always receive a message before you can delete it (you can't put a message into the queue and then recall it). The maximum length of a receipt handle is 1,024 characters.

> **Important**
>
> If you receive a message more than once, each time you receive it, you get a different receipt handle. You must provide the most recently received receipt handle when you request to delete the message (otherwise, the message might not be deleted).

The following is an example of a receipt handle (broken across three lines).

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

Amazon SQS FIFO queues

FIFO (First-In-First-Out) queues have all the capabilities of the standard queues, but are designed to enhance messaging between applications when the order of operations and events is critical, or where duplicates can't be tolerated.

The most important features of FIFO queues are *FIFO (First-In-First-Out) delivery* and *exactly-once processing*:

- The order in which messages are sent and received is strictly preserved and a message is delivered once and remains unavailable until a consumer processes and deletes it.
- Duplicates aren't introduced into the queue.

Additionally, FIFO queues support *message groups* that allow multiple ordered message groups within a single queue. There is no quota to the number of message groups within a FIFO queue.

Examples of situations where you might use FIFO queues include the following:

1. E-commerce order management system where order is critical
2. Integrating with a third-party systems where events need to be processed in order
3. Processing user-entered inputs in the order entered
4. Communications and networking – Sending and receiving data and information in the same order
5. Computer systems – Making sure that user-entered commands are run in the right order
6. Educational institutes – Preventing a student from enrolling in a course before registering for an account
7. Online ticketing system – Where tickets are distributed on a first come first serve basis

Note

FIFO queues also provide exactly-once processing, but have a limited number of transactions per second (TPS). You can use Amazon SQS **high throughput** mode with your FIFO queue to increase your transaction limit. For details on using high throughput mode, see High throughput for FIFO queues in Amazon SQS. For information on throughput quotas, see the section called “Message quotas”.

Amazon SQS FIFO queues are available in all Regions where Amazon SQS is available.

For more on using FIFO queues with complex ordering, see Solving Complex Ordering Challenges with Amazon SQS FIFO Queues.

For information about how to create and configure queues using the Amazon SQS console, see Create a queue using the Amazon SQS console. For Java examples, see Amazon SQS Java SDK examples.

For best practices for working with FIFO queues, see Amazon SQS best practices.

Amazon SQS FIFO queue key terms

The following key terms can help you better understand the functionality of FIFO queues. For more information, see the *Amazon Simple Queue Service API Reference*.

Clients

The Amazon SQS Buffered Asynchronous Client doesn't currently support FIFO queues.

Message deduplication ID

The token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute deduplication interval.

> **Note**
>
> Amazon SQS continues to keep track of the message deduplication ID even after the message is received and deleted.

Message group ID

The tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are always processed one by one, in a strict order relative to the message group (however, messages that belong to different message groups might be processed out of order).

Receive request attempt ID

The token used for deduplication of `ReceiveMessage` calls.

Sequence number

The large, non-consecutive number that Amazon SQS assigns to each message.

Services

If your application uses multiple AWS services, or a mix of AWS and external services, it is important to understand which service functionality doesn't support FIFO queues.

Some AWS or external services that send notifications to Amazon SQS might not be compatible with FIFO queues, despite allowing you to set a FIFO queue as a target.

The following features of AWS services aren't currently compatible with FIFO queues:

- Amazon S3 Event Notifications
- Auto Scaling Lifecycle Hooks
- AWS IoT Rule Actions
- AWS Lambda Dead-Letter Queues

For information about compatibility of other services with FIFO queues, see your service documentation.

FIFO queue delivery logic in Amazon SQS

The following concepts can help you better understand the sending of messages to and receiving messages from FIFO.

Sending messages

If multiple messages are sent in succession to a FIFO queue, each with a distinct message deduplication ID, Amazon SQS stores the messages and acknowledges the transmission. Then, each message can be received and processed in the exact order in which the messages were transmitted.

In FIFO queues, messages are ordered based on message group ID. If multiple hosts (or different threads on the same host) send messages with the same message group ID to a FIFO queue, Amazon SQS stores the messages in the order in which they arrive for processing. To make sure that Amazon SQS preserves the order in which messages are sent and received, each producer should use a unique message group ID to send all its messages.

FIFO queue logic applies only per message group ID. Each message group ID represents a distinct ordered message group within an Amazon SQS queue. For each message group ID, all messages are sent and received in strict order. However, messages with different message group ID values might be sent and received out of order. You must associate a message group ID with a message. If you don't provide a message group ID, the action fails. If you require a single group of ordered messages, provide the same message group ID for messages sent to the FIFO queue.

Receiving messages

You can't request to receive messages with a specific message group ID.

When receiving messages from a FIFO queue with multiple message group IDs, Amazon SQS first attempts to return as many messages with the same message group ID as possible. This allows other consumers to process messages with a different message group ID. When you receive a message with a message group ID, no more messages for the same message group ID are returned unless you delete the message or it becomes visible.

> **Note**
>
> It is possible to receive up to 10 messages in a single call using the `MaxNumberOfMessages` request parameter of the `ReceiveMessage` action. These messages retain their FIFO order and can have the same message group ID. Thus, if there are fewer than 10 messages available with the same message group ID, you might receive messages from another message group ID, in the same batch of 10 messages, but still in FIFO order.

Retrying multiple times

FIFO queues allow the producer or consumer to attempt multiple retries:

- If the producer detects a failed `SendMessage` action, it can retry sending as many times as necessary, using the same message deduplication ID. Assuming that the producer receives at least one acknowledgement before the deduplication interval expires, multiple retries neither affect the ordering of messages nor introduce duplicates.
- If the consumer detects a failed `ReceiveMessage` action, it can retry as many times as necessary, using the same receive request attempt ID. Assuming that the consumer receives at least one acknowledgement before the visibility timeout expires, multiple retries don't affect the ordering of messages.

- When you receive a message with a message group ID, no more messages for the same message group ID are returned unless you delete the message or it becomes visible.

Exactly-once processing in Amazon SQS

Unlike standard queues, FIFO queues don't introduce duplicate messages. FIFO queues help you avoid sending duplicates to a queue. If you retry the `SendMessage` action within the 5-minute deduplication interval, Amazon SQS doesn't introduce any duplicates into the queue.

To configure deduplication, you must do one of the following:

- Enable content-based deduplication. This instructs Amazon SQS to use a SHA-256 hash to generate the message deduplication ID using the body of the message—but not the attributes of the message. For more information, see the documentation on the `CreateQueue`, `GetQueueAttributes`, and `SetQueueAttributes` actions in the *Amazon Simple Queue Service API Reference*.
- Explicitly provide the message deduplication ID (or view the sequence number) for the message. For more information, see the documentation on the `SendMessage`, `SendMessageBatch`, and `ReceiveMessage` actions in the *Amazon Simple Queue Service API Reference*.

Moving from a standard queue to a FIFO queue in Amazon SQS

If your existing application uses standard queues and you want to take advantage of the ordering or exactly-once processing features of FIFO queues, you need to configure both the queue and your application correctly.

Key considerations

- **Creating a FIFO Queue:** You cannot convert an existing standard queue into a FIFO queue. You must either create a new FIFO queue for your application or delete the existing standard queue and recreate it as a FIFO queue.
- **Delay Parameter:** FIFO queues do not support per-message delays, only per-queue delays. If your application sets the `DelaySeconds` parameter on each message, you must modify it to set `DelaySeconds` on the entire queue instead.
- **Message Group ID:** Provide a message group ID for every sent message. This ID enables parallel processing of messages while maintaining their respective order. Use a granular business

dimension for the message group ID to better scale with FIFO queues. The more message group IDs you distribute messages to, the greater the number of messages available for consumption.

- **High Throughput Mode:** Use the recommended high throughput mode for FIFO queues to achieve increased throughput. For more information on messaging quotas, see Amazon SQS message quotas.

Checklist for moving to FIFO queues

Before sending messages to a FIFO queue, confirm the following:

1. **Configure delay settings**
   - Modify your application to remove per-message delays.
   - Set the `DelaySeconds` parameter on the entire queue.
2. **Set message group IDs**
   - Organize messages into message groups by specifying a message group ID based on a business dimension.
   - Use more granular business dimensions to improve scalability.
3. **Handle message deduplication**
   - If your application can send messages with identical message bodies, provide a unique message deduplication ID for each message.
   - If your application sends messages with unique message bodies, enable content-based deduplication.
4. **Configure the consumer**
   - Generally, no code changes are needed for the consumer.
   - If processing messages takes a long time and the visibility timeout is set high, consider adding a receive request attempt ID to each `ReceiveMessage` action. This helps retry receive attempts in case of networking failures and prevents queues from pausing due to failed receive attempts.

By following these steps, you can ensure your application works correctly with FIFO queues, taking full advantage of their ordering and exactly-once processing features. For more detailed information, see the *Amazon Simple Queue Service API Reference*.

Amazon SQS FIFO queue and Lambda concurrency behavior

By using a FIFO (First-In-First-Out) queue with Lambda, you can ensure ordered processing of messages within each message group. The Lambda function will not run multiple instances for the same message group simultaneously, thereby maintaining the order. However, it can scale up to handle multiple message groups in parallel, ensuring efficient processing of your queue's workload. The following points describe the behavior of Lambda functions when processing messages from an Amazon SQS FIFO queue with respect to message group IDs:

- **Single instance per message group:** At any point in time, only one Lambda instance will be processing messages from a specific message group ID. This ensures that messages within the same group are processed in order, maintaining the integrity of the FIFO sequence.
- **Concurrent processing of different groups:** Lambda can concurrently process messages from different message group IDs using multiple instances. This means that while one instance of the Lambda function is handling messages from one message group ID, other instances can simultaneously handle messages from other message group IDs, leveraging the concurrency capabilities of Lambda to process multiple groups in parallel.

FIFO queue message grouping

FIFO queues ensure that messages are processed in the exact order they are sent. They use a **message group ID** to group messages that should be processed sequentially.

Messages within the same message group are processed in order, and only one message from each group is processed at a time to maintain this order.

Lambda concurrency with FIFO queues

After you create your queue, you can send a message to it.

When you set up a Lambda function to process messages from an Amazon SQS FIFO queue, Lambda respects the ordering guarantees provided by the FIFO queue. The following points describe the behavior of Lambda functions in terms of concurrency and scaling when processing messages from an Amazon SQS FIFO queue when using message group IDs.

- **Concurrency within message groups:** Only one Lambda instance processes messages for a particular message group ID at a time. This ensures that messages within a group are handled sequentially.

- **Scaling and multiple message groups:**While Lambda can scale up to process messages concurrently, this scaling occurs across different message groups. If you have multiple message groups, Lambda can process multiple groups in parallel, with each group being handled by a separate Lambda instance.

For more information, see Scaling and concurrency in Lambda in the *AWS Lambda Operator Guide*.

Use case example

Suppose your FIFO queue receives messages with the same message group ID, and your Lambda function has a high concurrency limit (up to 1000).

If a message from group ID 'A' is being processed and another message from group ID 'A' arrives, the second message will not trigger a new Lambda instance until the first message is fully processed.

However, if messages from group IDs 'A' and 'B' arrive, both messages can be processed concurrently by separate Lambda instances.

High throughput for FIFO queues in Amazon SQS

High throughput FIFO queues in Amazon SQS efficiently manage high message throughput while maintaining strict message order, ensuring reliability and scalability for applications processing numerous messages. This solution is ideal for scenarios demanding both high throughput and ordered message delivery.

Amazon SQS high throughput FIFO queues are not necessary in scenarios where strict message ordering is not crucial and where the volume of incoming messages is relatively low or sporadic. For instance, if you have a small-scale application that processes infrequent or non-sequential messages, the added complexity and cost associated with high throughput FIFO queues may not be justified. Additionally, if your application does not require the enhanced throughput capabilities provided by high throughput FIFO queues, opting for a standard Amazon SQS queue might be more cost-effective and simpler to manage.

To enhance request capacity in high throughput FIFO queues, increasing the number of message groups is recommended. For more information on high throughput message quotas, see Amazon SQS service quotas in the *Amazon Web Services General Reference*.

For information per-queue quotas and data distribution strategies, see Amazon SQS message quotas and Partitions and data distribution for high throughput for SQS FIFO queues.

Topics

Use cases for high throughput for Amazon SQS FIFO queues

The following use cases highlight the diverse applications of high throughput FIFO queues, showcasing their effectiveness across industries and scenarios:

1. **Real-time data processing:** Applications dealing with real-time data streams, such as event processing or telemetry data ingestion, can benefit from high throughput FIFO queues to handle the continuous influx of messages while preserving their order for accurate analysis.
2. **E-commerce order processing:** In e-commerce platforms where maintaining the order of customer transactions is critical, high throughput FIFO queues ensure that orders are processed sequentially and without delays, even during peak shopping seasons.
3. **Financial services:** Financial institutions handling high-frequency trading or transactional data rely on high throughput FIFO Queues to process market data and transactions with minimal latency while adhering to strict regulatory requirements for message ordering.
4. **Media streaming:** Streaming platforms and media distribution services utilize high throughput FIFO queues to manage the delivery of media files and streaming content, ensuring smooth playback experiences for users while maintaining the correct order of content delivery.

Partitions and data distribution for high throughput for SQS FIFO queues

Amazon SQS stores FIFO queue data in partitions. A *partition* is an allocation of storage for a queue that is automatically replicated across multiple Availability Zones within an AWS Region. You don't manage partitions. Instead, Amazon SQS handles partition management.

For FIFO queues, Amazon SQS modifies the number of partitions in a queue in the following situations:

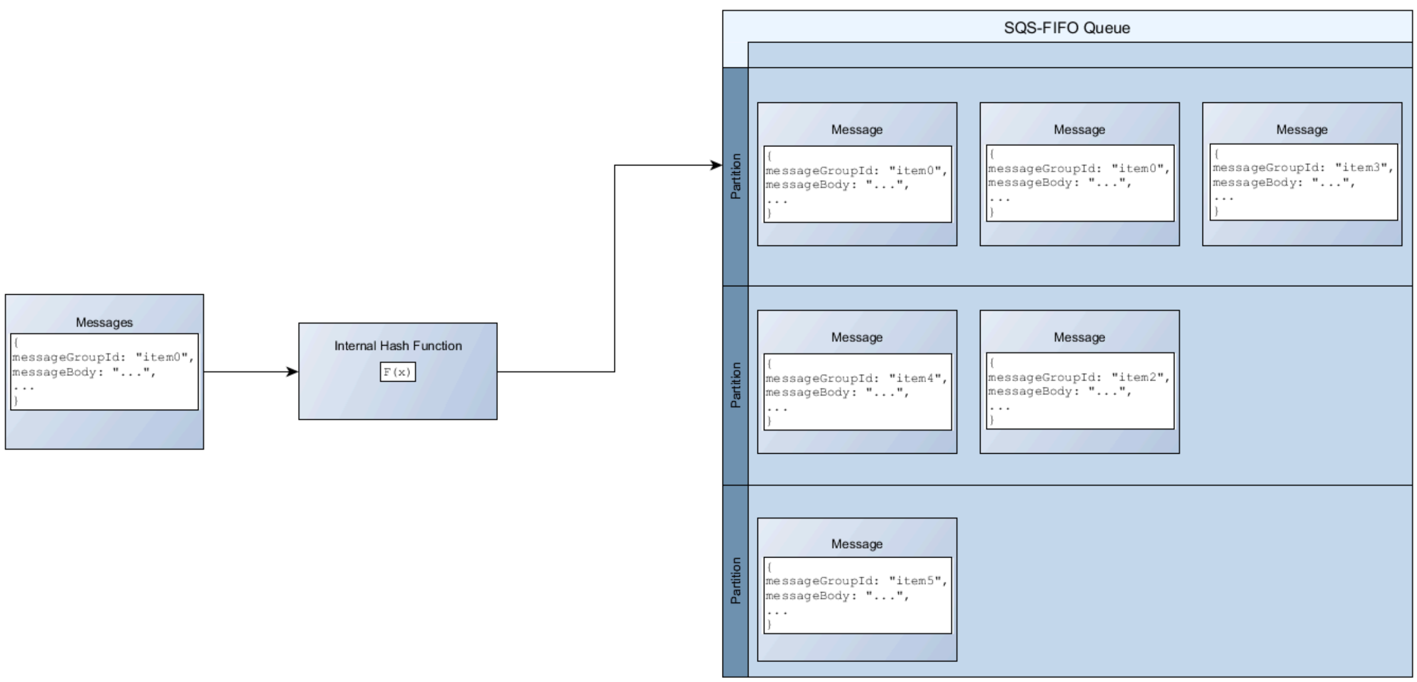
- If the current request rate approaches or exceeds what the existing partitions can support, additional partitions are allocated until the queue reaches the regional quota. For information on quotas, see [Amazon SQS message quotas](#).
- If the current partitions have low utilization, the number of partitions may be reduced.

Partition management occurs automatically in the background and is transparent to your applications. Your queue and messages are available at all times.

Distributing data by message group IDs

To add a message to a FIFO queue, Amazon SQS uses the value of each message's message group ID as input to an internal hash function. The output value from the hash function determines which partition stores the message.

The following diagram shows a queue that spans multiple partitions. The queue's message group ID is based on item number. Amazon SQS uses its hash function to determine where to store a new item; in this case, it's based on the hash value of the string `item0`. Note that the items are stored in the same order in which they are added to the queue. Each item's location is determined by the hash value of its message group ID.



Note

Amazon SQS is optimized for uniform distribution of items across a FIFO queue's partitions, regardless of the number of partitions. AWS recommends that you use message group IDs that can have a large number of distinct values.

Optimizing partition utilization

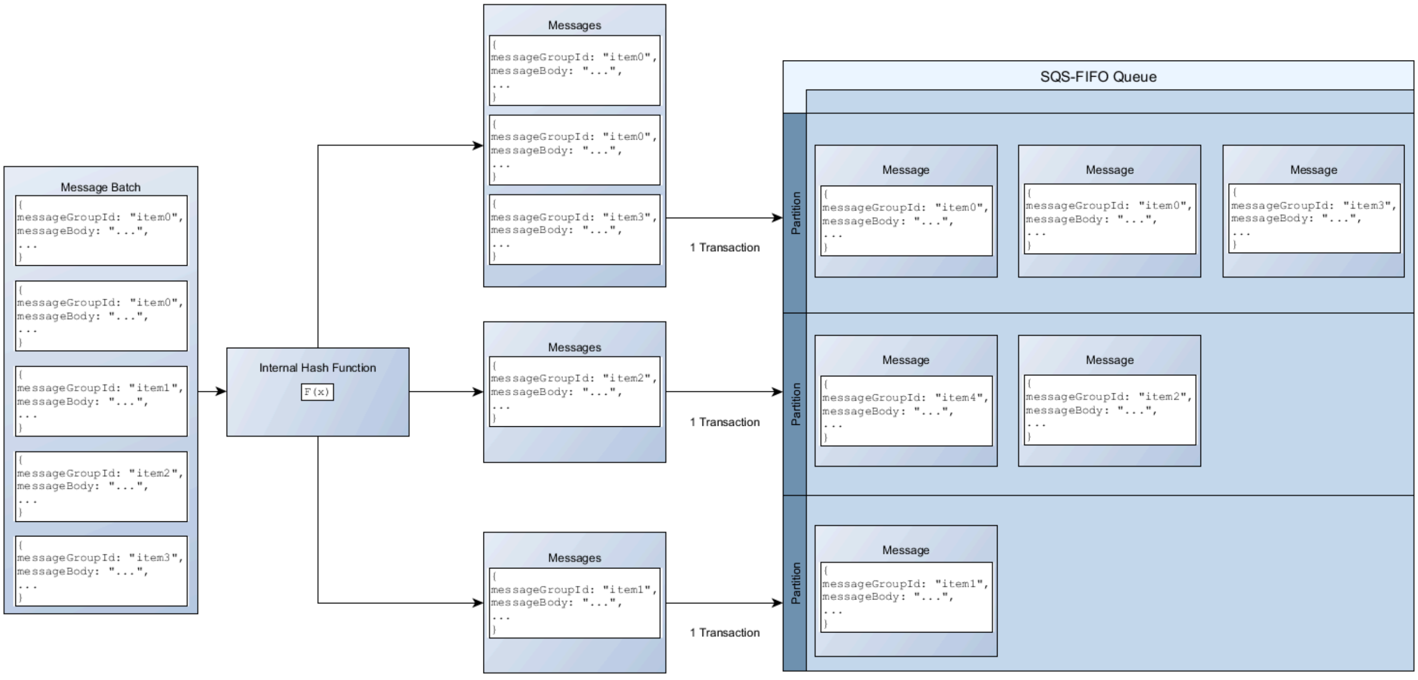
Each partition supports up to 3,000 messages per second with batching, or up to 300 messages per second for send, receive, and delete operations in supported regions. For more information on high throughput message quotas, see Amazon SQS service quotas in the *Amazon Web Services General Reference*.

When using batch APIs, each message is routed based on the process described in Distributing data by message group IDs. Messages that are routed to the same partition are grouped and processed in a single transaction.

To optimize partition utilization for the `SendMessageBatch` API, AWS recommends batching messages with the same message group IDs when possible.

To optimize partition utilization for the `DeleteMessageBatch` and `ChangeMessageVisibilityBatch` APIs, AWS recommends using `ReceiveMessage` requests with the `MaxNumberOfMessages` parameter set to 10, and batching the receipt-handles returned by a single `ReceiveMessage` request.

In the following example, a batch of messages with various message group IDs is sent. The batch is split into three groups, each of which counts against the quota for the partition.



> **Note**
>
> Amazon SQS only guarantees that messages with the same message group ID's internal hash function are grouped within a batch request. Depending on the output of the internal hash function and the number of partitions, messages with different message group IDs might be grouped. Since the hash function or number of partitions can change at any time, messages that are grouped at one point may not be grouped later.

Enabling high throughput for FIFO queues in Amazon SQS

You can enable high throughput for any new or existing FIFO queue. The feature includes three new options when you create and edit FIFO queues:

- **Enable high throughput FIFO** – Makes higher throughput available for messages in the current FIFO queue.
- **Deduplication scope** – Specifies whether deduplication occurs at the queue or message group level.
- **FIFO throughput limit** – Specifies whether the throughput quota on messages in the FIFO queue is set at the queue or message group level.

To enable high throughput for a FIFO queue (console)

1. Start creating or editing a FIFO queue.
2. When specifying options for the queue, choose **Enable high throughput FIFO**.

   Enabling high throughput for FIFO queues sets the related options as follows:

   - **Deduplication scope** is set to **Message group**, the required setting for using high throughput for FIFO queues.
   - **FIFO throughput limit** is set to **Per message group ID**, the required setting for using high throughput for FIFO queues.

   If you change any of the settings required for using high throughput for FIFO queues, normal throughput is in effect for the queue, and deduplication occurs as specified.
3. Continue specifying all options for the queue. When you finish, choose **Create queue** or **Save**.

After creating or editing the FIFO queue, you can send messages to it and receive and delete messages, all at a higher TPS. For high throughput quotas, see Message throughput in Amazon SQS message quotas.

FIFO queue and message identifiers in Amazon SQS

This section describes the identifiers of FIFO queues. These identifiers can help you find and manipulate specific queues and messages.

Topics

- Identifiers for FIFO queues in Amazon SQS
- Additional identifiers for Amazon SQS FIFO queues

Identifiers for FIFO queues in Amazon SQS

For more information about the following identifiers, see the *Amazon Simple Queue Service API Reference*.

Queue name and URL

When you create a new queue, you must specify a queue name unique for your AWS account and region. Amazon SQS assigns each queue you create an identifier called a *queue URL* that includes the queue name and other Amazon SQS components. Whenever you want to perform an action on a queue, you provide its queue URL.

The name of a FIFO queue must end with the `.fifo` suffix. The suffix counts towards the 80-character queue name quota. To determine whether a queue is FIFO, you can check whether the queue name ends with the suffix.

The following is the queue URL for a FIFO queue named `MyQueue` owned by a user with the AWS account number `123456789012`.

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

You can retrieve the URL of a queue programmatically by listing your queues and parsing the string that follows the account number. For more information, see `ListQueues`.

Message ID

Each message receives a system-assigned *message ID* that Amazon SQS returns to you in the `SendMessage` response. This identifier is useful for identifying messages. The maximum length of a message ID is 100 characters.

Receipt handle

Every time you receive a message from a queue, you receive a *receipt handle* for that message. This handle is associated with the action of receiving the message, not with the message itself. To delete the message or to change the message visibility, you must provide the receipt handle (not the message ID). Thus, you must always receive a message before you can delete it (you can't put a message into the queue and then recall it). The maximum length of a receipt handle is 1,024 characters.

> **Important**
>
> If you receive a message more than once, each time you receive it, you get a different receipt handle. You must provide the most recently received receipt handle when you request to delete the message (otherwise, the message might not be deleted).

The following is an example of a receipt handle (broken across three lines).

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

Additional identifiers for Amazon SQS FIFO queues

For more information about the following identifiers, see Exactly-once processing in Amazon SQS and the *Amazon Simple Queue Service API Reference*.

Message deduplication ID

The token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute deduplication interval.

Message group ID

The tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are always processed one by one, in a strict order relative to the message group (however, messages that belong to different message groups might be processed out of order).

Sequence number

The large, non-consecutive number that Amazon SQS assigns to each message.

Amazon SQS quotas

This topic lists quotas within Amazon Simple Queue Service (Amazon SQS).

Topics

- Amazon SQS FIFO queue quotas
- Amazon SQS standard queue quotas
- Amazon SQS message quotas
- Amazon SQS policy quotas

Amazon SQS FIFO queue quotas

Amazon SQS quotas

The following table lists quotas related to FIFO queues.

Quota	Description
Delay queue	The default (minimum) delay for a queue is 0 seconds. The maximum is 15 minutes.
Listed queues	1,000 queues per `ListQueues` request.
Long polling wait time	The maximum long polling wait time is 20 seconds.
Message groups	There is no quota to the number of message groups within a FIFO queue.
Messages per queue (backlog)	The number of messages that an Amazon SQS queue can store is unlimited.
Messages per queue (in flight)	For FIFO queues, there can be a maximum of 20,000 in flight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this quota, Amazon SQS returns no error messages.

Quota	Description
Queue name	The name of a FIFO queue must end with the `.fifo` suffix. The suffix counts towards the 80-character queue name quota. To determine whether a queue is FIFO, you can check whether the queue name ends with the suffix.
Queue tag	We don't recommend adding more than 50 tags to a queue. Tagging supports Unicode characters in UTF-8.
	The tag `Key` is required, but the tag `Value` is optional.
	The tag `Key` and tag `Value` are case-sensitive.
	The tag `Key` and tag `Value` can include Unicode alphanumeric characters in UTF-8 and whitespaces. The following special characters are allowed: `_ . : / = + - @`
	The tag `Key` or `Value` must not include the reserved prefix `aws:` (you can't delete tag keys or values with this prefix).
	The maximum tag `Key` length is 128 Unicode characters in UTF-8. The tag `Key` must not be empty or null.
	The maximum tag `Value` length is 256 Unicode characters in UTF-8. The tag `Value` may be empty or null.
	Tagging actions are limited to 30 TPS per AWS account. If your application requires a higher throughput, submit a request.

Amazon SQS standard queue quotas

The following table lists quotas related to standard queues.

Quota	Description
Delay queue	The default (minimum) delay for a queue is 0 seconds. The maximum is 15 minutes.
Listed queues	1,000 queues per `ListQueues` request.
Long polling wait time	The maximum long polling wait time is 20 seconds.
Messages per queue (backlog)	The number of messages that an Amazon SQS queue can store is unlimited.
Messages per queue (in flight)	For most standard queues (depending on queue traffic and message backlog), there can be a maximum of approximately 120,000 in flight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this quota while using short polling, Amazon SQS returns the `OverLimit` error message. If you use long polling, Amazon SQS returns no error messages. To avoid reaching the quota, you should delete messages from the queue after they're processed. You can also increase the number of queues you use to process your messages. To request a quota increase, submit a support request.
Queue name	A queue name can have up to 80 characters. The following characters are accepted: alphanumeric characters, hyphens (-), and underscores (_). **Note** Queue names are case-sensitive (for example, `Test-queue` and `test-queue` are different queues).
Queue tag	We don't recommend adding more than 50 tags to a queue. Tagging supports Unicode characters in UTF-8.

Quota	Description
	The tag `Key` is required, but the tag `Value` is optional.
	The tag `Key` and tag `Value` are case-sensitive.
	The tag `Key` and tag `Value` can include Unicode alphanumeric characters in UTF-8 and whitespaces. The following special characters are allowed: `_ . : / = + - @`
	The tag `Key` or `Value` must not include the reserved prefix `aws:` (you can't delete tag keys or values with this prefix).
	The maximum tag `Key` length is 128 Unicode characters in UTF-8. The tag `Key` must not be empty or null.
	The maximum tag `Value` length is 256 Unicode characters in UTF-8. The tag `Value` may be empty or null.
	Tagging actions are limited to 30 TPS per AWS account. If your application requires a higher throughput, submit a request.

Amazon SQS message quotas

The following table lists quotas related to messages.

Quota	Description
Batched message ID	A batched message ID can have up to 80 characters. The following characters are accepted: alphanumeric characters, hyphens (-), and underscores (_).
Message attributes	A message can contain up to 10 metadata attributes.

Quota	Description
Message batch	A single message batch request can include a maximum of 10 messages. For more information, see Configuring AmazonSQSBufferedAsyncClient in the Amazon SQS batch actions section.
Message content	A message can include only XML, JSON, and unformatted text. The following Unicode characters are allowed: `#x9` | `#xA` | `#xD` | `#x20` to `#xD7FF` | `#xE000` to `#xFFFD` | `#x10000` to `#x10FFFF` Any characters not included in this list are rejected. For more information, see the W3C specification for characters.
Message group ID	Consume messages from the backlog to avoid building up a large backlog of messages with the same message group ID. `MessageGroupId` is required for FIFO queues. You can't use it for Standard queues. You must associate a non-empty `MessageGroupId` with a message. If you don't provide a `MessageGroupId`, the action fails. The length of `MessageGroupId` is 128 characters. Valid values: alphanumeric characters and punctuation `(!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~)`.
Message retention	By default, a message is retained for 4 days. The minimum is 60 seconds (1 minute). The maximum is 1,209,600 seconds (14 days).
Message throughput	Standard queues support a nearly unlimited number of API calls per second, per API action (`SendMessage`, `ReceiveMessage`, or `DeleteMessage`).

Quota	Description
	FIFO queues • FIFO queues support a quota of 300 transactions per second, per API action (`SendMessage`, `ReceiveMessage`, and `DeleteMessage`). • If you use batching, FIFO queues support up to 3,000 messages per second, per API action (`SendMessage`, `ReceiveMessage`, and `DeleteMessage`). The 3,000 messages per second represent 300 API calls, each with a batch of 10 messages.

Quota	Description
	High throughput for FIFO queues • Without batching (`SendMessage` , `ReceiveMessage` , and `DeleteMessage`), high throughput for FIFO queues process up to 70,000 transactions per second, per API action in US East (N. Virginia), US West (Oregon), and Europe (Ireland) Regions. • For US East (Ohio) and Europe (Frankfurt) Regions, the default throughput is 18,000 transactions per second per API action. • For Asia Pacific (Mumbai), Asia Pacific (Singapore), Asia Pacific (Sydney) and Asia Pacific (Tokyo) Regions, the default throughput is 9,000 transactions per second per API action. • For Europe (London) and South America (São Paulo), the default throughput is 4,500 transactions per second per API action. • For maximum throughput, increase the number of message group IDs you use for messages sent without batching. • You can increase throughput up to 700,000 messages per second by using batching APIs (`SendMessageBatch` and `DeleteMessageBatch`)in US East (N. Virginia), US West (Oregon), and Europe (Ireland) Regions. The 700,000 messages per second represents 70,000 transactions per second, each with a batch of 10 messages. For Europe (Frankfurt) and US East (Ohio) Regions, you can achieve up to 180,000 messages per second by using batching APIs. The 180,000 messages per second represents 18,000 transactions per second, each with a batch of 10 messages.

Quota	Description
	For Asia Pacific (Mumbai), Asia Pacific (Singapore), Asia Pacific (Sydney) and Asia Pacific (Tokyo) Regions, you can achieve up to 90,000 messages per second with batching. To achieve the maximum throughput when using `SendMessageBatch` and `DeleteMessageBatch`, all messages in a batch request must use the same message group ID. • For Europe (London) and South America (São Paulo), regions, you can achieve up to 45,000 messages per second with batching. To achieve the maximum throughput when using `SendMessageBatch` and `DeleteMessageBatch`, all messages in a batch request must use the same message group ID. • In all other AWS Regions, maximum throughput is 2,400 (without batching) or 24,000 (using batching) messages per second, per API action. • To request an increase above the region limit, contact the AWS Support Center for assistance. • For more information, see Partitions and data distribution for high throughput for SQS FIFO queues.
Message timer	The default (minimum) delay for a message is 0 seconds. The maximum is 15 minutes.

Quota	Description
Message size	The minimum message size is 1 byte (1 character). The maximum is 262,144 bytes (256 KiB). To send messages larger than 256 KiB, you can use the Amazon SQS Extended Client Library for Java and the Amazon SQS Extended Client Library for Python. This library allows you to send an Amazon SQS message that contains a reference to a message payload in Amazon S3. The maximum payload size is 2 GB. **Note** This extended library works only for synchronous clients.
Message visibility timeout	The default visibility timeout for a message is 30 seconds. The minimum is 0 seconds. The maximum is 12 hours.
Policy information	The maximum quota is 8,192 bytes, 20 statements, 50 principals, or 10 conditions. For more information, see Amazon SQS policy quotas.

Amazon SQS policy quotas

The following table lists quotas related to policies.

Name	Maximum
Bytes	8,192
Conditions	10
Principals	50

Name	Maximum
Statements	20
Actions per statement	7

Amazon SQS features and capabilities

This topic provides commonly used features in Amazon SQS for managing message queues, optimizing performance, ensuring reliable message delivery, and handling message processing efficiently.

Topics

Using dead-letter queues in Amazon SQS

Amazon SQS supports *dead-letter queues* (DLQs), which source queues can target for messages that are not processed successfully. DLQs are useful for debugging your application because you can isolate unconsumed messages to determine why processing did not succeed. For optimal performance, it is a best practice to keep the source queue and DLQ within the same AWS account and Region. Once messages are in a dead-letter queue, you can:

- Examine logs for exceptions that might have caused messages to be moved to a dead-letter queue.
- Analyze the contents of messages moved to the dead-letter queue to diagnose application issues.

- Determine whether you have given your consumer sufficient time to process messages.
- Move messages out of the dead-letter queue using dead-letter queue redrive.

You must first create a new queue before configuring it as a dead-letter queue. For information about configuring a dead-letter queue using the Amazon SQS console, see Learn how to configure a dead-letter queue using the Amazon SQS console. For help with dead-letter queues, such as how to configure an alarm for any messages moved to a dead-letter queue, see Create alarms for dead-letter queues using Amazon CloudWatch.

Using policies for dead-letter queues

Use a **redrive policy** to specify the `maxReceiveCount`. The `maxReceiveCount` is the number of times a consumer can receive a message from a source queue before it is moved to a dead-letter queue. For example, if the `maxReceiveCount` is set to a low value such as 1, one failure to receive a message would cause the message to move to the dead-letter queue. To ensure that your system is resilient against errors, set the `maxReceiveCount` high enough to allow for sufficient retries.

The **redrive allow policy** specifies which source queues can access the dead-letter queue. You can choose whether to allow all source queues, allow specific source queues, or deny all source queues use of the dead-letter queue. The default allows all source queues to use the dead-letter queue. If you choose to allow specific queues using the `byQueue` option, you can specify up to 10 source queues using the source queue Amazon Resource Name (ARN). If you specify `denyAll`, the queue cannot be used as a dead-letter queue.

Understanding message retention periods for dead-letter queues

For standard queues, the expiration of a message is always based on its original enqueue timestamp. When a message is moved to a dead-letter queue, the enqueue timestamp is unchanged. The `ApproximateAgeOfOldestMessage` metric indicates when the message moved to the dead-letter queue, not when the message was originally sent. For example, assume that a message spends 1 day in the original queue before it's moved to a dead-letter queue. If the dead-letter queue's retention period is 4 days, the message is deleted from the dead-letter queue after 3 days and the `ApproximateAgeOfOldestMessage` is 3 days. Thus, it is a best practice to always set the retention period of a dead-letter queue to be longer than the retention period of the original queue.

For FIFO queues, the enqueue timestamp resets when the message is moved to a dead-letter queue. The `ApproximateAgeOfOldestMessage` metric indicates when the message moved to the dead-letter queue. In the same example above, the message is deleted from the dead-letter queue after 4 days and the `ApproximateAgeOfOldestMessage` is 4 days.

Learn how to configure a dead-letter queue using the Amazon SQS console

A *dead-letter queue* is a queue which source queues can target for messages that are not processed successfully. For more information, see Using dead-letter queues in Amazon SQS .

Amazon SQS does *not* create the dead-letter queue automatically. You must first create the queue before using it as a dead-letter queue. For instructions on creating a queue to use as a dead letter queue, see Create a queue using the Amazon SQS console.

The dead-letter queue of a FIFO queue must also be a FIFO queue. Similarly, the dead-letter queue of a standard queue must also be a standard queue.

When you create or edit a queue, you can configure a dead-letter queue.

To configure a dead-letter queue for an existing queue (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. Select a queue and choose **Edit**.
4. Scroll to the **Dead-letter queue** section and choose **Enabled**.
5. Choose the Amazon Resource Name (ARN) of an existing **Dead Letter Queue** that you want to associate with this source queue.
6. To configure the number of times that a message can be received before being sent to a dead-letter queue, set **Maximum receives** to a value between 1 and 1,000.
7. When you finish configuring the dead-letter queue, choose **Save**.

   After you save the queue, the console displays the **Details** page for your queue. On the **Details** page, the **Dead-letter queue** tab displays the **Maximum Receives** and **Dead Letter Queue** ARN in the **Dead-letter queue**.

Learn how to configure a dead-letter queue redrive in Amazon SQS

You can use dead-letter queue redrive to move unconsumed messages out of an existing dead-letter queue. By default, dead-letter queue redrive moves messages from a dead-letter queue to a source queue. However, you can also configure any other queue as the redrive destination if both queues are the same type. For example, if the dead-letter queue is a FIFO queue, the redrive destination queue must be a FIFO queue as well. Additionally, you can configure the redrive velocity to set the rate at which Amazon SQS moves messages.

> **Note**
>
> When a message is moved from a FIFO queue to a FIFO DLQ, the original message's deduplication ID will be replaced with the original message's ID. This is to make sure that the DLQ deduplication will not prevent storing of two independent messages that happen to share a deduplication ID.

Dead-letter queues redrive messages in the order they are received, starting with the oldest message. However, the destination queue ingests the redriven messages, as well as new messages from other producers, according to the order in which it receives them. For example, if a producer is sending messages to a source FIFO queue when simultaneously receiving redriven messages from a dead letter queue, the redriven messages will interweave with the new messages from the producer.

> **Note**
>
> The redrive task resets the retention period. All redriven messages are considered new messages with a new `messageID` and `enqueueTime` are assigned to redriven messages.

Topics

Configuring a dead-letter queue redrive for an existing standard queue using the Amazon SQS API

You can configure a dead-letter queue redrive using the `SendMessageBatch`, `ReceiveMessage`, and `DeleteMessageBatch` API actions:

API action	Description
`StartMessageMoveTask`	Starts an asynchronous task to move messages from a specified source queue to a specified destination queue.
`ListMessageMoveTasks`	Gets the most recent message movement tasks (up to 10) under a specific source queue.
`CancelMessageMoveTask`	Cancels a specified message movement task. A message movement can only be cancelled when the current status is RUNNING.

Configuring a dead-letter queue redrive for an existing standard queue using the Amazon SQS console

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. Choose the name of queue that you have configured as a dead-letter queue.
4. Choose **Start DLQ redrive**.
5. Under **Redrive configuration**, for **Message destination**, do either of the following:
   - To redrive messages to their source queue, choose **Redrive to source queue(s)**.
   - To redrive messages to another queue, choose **Redrive to custom destination**. Then, enter the Amazon Resource Name (ARN) of an existing destination queue.
6. Under **Velocity control settings**, choose one of the following:
   - **System optimized** - Redrive dead-letter queue messages at the maximum number of messages per second.

- **Custom max velocity** - Redrive dead-letter queue messages with a custom maximum rate of messages per second. The maximum allowed rate is 500 messages per second.
  - It is recommended to start with a small value for Custom max velocity and verify that the source queue doesn't get overwhelmed with messages. From there, gradually ramp-up the Custom max velocity value, continuing to monitor the state of the source queue.

7. When you finish configuring the dead-letter queue redrive, choose **Redrive messages**.

> **Important**
>
> Amazon SQS doesn't support filtering and modifying messages while redriving them from the dead-letter queue.
> A dead-letter queue redrive task can run a maximum of 36 hours. Amazon SQS supports a maximum of 100 active redrive tasks per account.

8. If you want to cancel the message redrive task, on the **Details** page for your queue, choose **Cancel DLQ redrive**. When canceling an in progress message redrive, any messages that have already been successfully moved to their move destination queue will remain in the destination queue.

Configuring queue permissions for dead-letter queue redrive

You can give user access to specific dead-letter queue actions by adding permissions to your policy. The minimum required permissions for a dead-letter queue redrive are as follows:

Minimum Permissions	Required API methods
To start a message redrive	• Add the `sqs:StartMessageMoveTask` , `sqs:ReceiveMessage` , `sqs:DeleteMessage` , and `sqs:GetQueueAttributes` of the dead-letter queue. If either the dead-letter queue or the original source queue are encrypted (also known as an SSE queue), `kms:Decrypt` for any KMS key that has been used to encrypt the messages is also required. •

Minimum Permissions	Required API methods
	Add the `sqs:SendMessage` of the destination queue. If the destination queue is encrypted, `kms:GenerateDataKey` and `kms:Decrypt` are also required.
To cancel an in-progress message redrive	• Add the `sqs:CancelMessageMoveTask` , `sqs:ReceiveMessage` , `sqs:DeleteMessage` , and `sqs:GetQueueAttributes` of the dead-letter queue. If the dead-letter queue is encrypted (also known as an SSE queue), `kms:Decrypt` is also required.
To show a message move status	• Add the `sqs:ListMessageMoveTasks` and `sqs:GetQueueAttributes` of the dead-letter queue.

To configure permissions for an encrypted queue pair (a source queue with a dead-letter queue)

Use the following steps to configure minimum permissions for a dead-letter queue redrive:

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Create a **policy** with the following permissions and attach it to your login IAM user or role:

   - `sqs:StartMessageMoveTask`
   - `sqs:CancelMessageMoveTask`
   - `sqs:ListMessageMoveTasks`
   - `sqs:ListDeadLetterSourceQueues`
   - `sqs:ReceiveMessage`
   - `sqs:DeleteMessage`
   - `sqs:GetQueueAttributes`
   - The `Resource` ARN of the dead-letter queue (for example, "arn:aws:sqs:*<DLQ_region>*:*<DLQ_accountId>*:*<DLQ_name>*").

- `sqs:SendMessage`
- The `Resource` ARN of the destination queue (for example, "arn:aws:sqs:*<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>*").
- `kms:Decrypt` – Allows decryption action.
- `kms:GenerateDataKey`
- The `Resource` ARN(s) of any KMS encryption key that has been used to encrypt the messages in the original source queue (for example, "arn:aws:kms:*<region>*:*<accountId>*:key/*<keyId_used to encrypt the message body>*").
- The Resource ARN of the KMS encryption key that is used for the redrive destination queue (for example, "arn:aws:kms:*<region>*:*<accountId>*:key/*<keyId_used for the destination queue>*").

Your access policy should resemble the following:

```
{
  "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>"
    },
    {
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource":
 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>"
    },
    {
      "Effect": "Allow",
```

```
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:<region>:<accountId>:key/<keyId>"
    }
  ]
}
```

To configure permissions using a non-encrypted queue pair (a source queue with a dead-letter queue)

Use the following steps to configure minimum permissions for a standard **unencrypted** dead-letter queue. Required minimum permissions are to *receive*, *delete* and *get* attributes from the dead-letter queue, and *send* attributes to the source queue.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Policies**.
3. Create a **policy** with the following permissions and attach it to your login IAM user or role:

   - `sqs:StartMessageMoveTask`
   - `sqs:CancelMessageMoveTask`
   - `sqs:ListMessageMoveTasks`
   - `sqs:ListDeadLetterSourceQueues`
   - `sqs:ReceiveMessage`
   - `sqs:DeleteMessage`
   - `sqs:GetQueueAttributes`
   - The `Resource` ARN of the dead-letter queue (for example, "arn:aws:sqs:*<DLQ_region>*:*<DLQ_accountId>*:*<DLQ_name>*") .
   - `sqs:SendMessage`
   - The `Resource` ARN of the destination queue (for example, "arn:aws:sqs:*<DestQueue_region>*:*<DestQueue_accountId>*:*<DestQueue_name>*").

   Your access policy should resemble the following:

```
{
"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>"
    },
    {
"Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource":
 "arn:aws:sqs:<DestQueue_region>:<DestQueue_accountId>:<DestQueue_name>"
    }
  ]
}
```

CloudTrail update and permission requirements for Amazon SQS dead-letter queue redrive

On June 8, 2023, Amazon SQS introduced dead-letter queue (DLQ) redrive for AWS SDK and AWS Command Line Interface (CLI). This capability is an addition to the already supported DLQ redrive for the AWS console. If you've previously used the AWS console to redrive dead-letter queue messages, you may be affected by the following changes:

- CloudTrail event renaming for dead-letter queue redrive
- Updated permissions for dead-letter queue redrive

CloudTrail event renaming

On October 15, 2023, the CloudTrail event names for dead-letter queue redrive will change on the Amazon SQS console. If you've set alarms for these CloudTrail events, you must update them now. The following are the new CloudTrail event names for DLQ redrive:

Previous event name	New event name
`CreateMoveTask`	`StartMessageMoveTask`
`CancelMoveTask`	`CancelMessageMoveTask`

Updated permissions

Included with the SDK and CLI release, Amazon SQS has also updated queue permissions for DLQ redrive to adhere to security best practices. Use the following queue permission types to redrive messages from your DLQs.

1. Action-based permissions (update for the DLQ API actions)
2. Managed Amazon SQS policy permissions
3. Permission policy that uses sqs:* wildcard

> **Important**
>
> To use the DLQ redrive for SDK or CLI, you are required to have a DLQ redrive permission policy that matches one of the above options.

If your queue permissions for DLQ redrive don't match one of the options above, you must update your permissions by August 31, 2023. Between now and August 31, 2023, your account will be able to redrive messages using the permissions you configured using the AWS console only in the regions where you have previously used the DLQ redrive. For example, say you had "Account A" in both us-east-1 and eu-west-1. "Account A" was used to redrive messages on the AWS console in us-east-1 prior to June 8, 2023, but not in eu-west-1. Between June 8, 2023 and August 31, 2023, if "Account A’s" policy permissions don't match one of the options above, it can only be used to redrive messages on the AWS console in us-east-1, and not in eu-west-1.

> **Important**
>
> If your DLQ redrive permissions do not match one of these options after August 31, 2023, your account will no longer be able to redrive DLQ messages using the AWS console. However, if you used the DLQ redrive feature on the AWS Console during August 2023, you have an extension until October 15, 2023 to adopt the new permissions according to one of these options.
>
> For more information, see the section called "Identifying impacted policies".

The following are queue permission examples for each DLQ redrive option. When using server-side encrypted (SSE) queues, the corresponding AWS KMS key permission is required.

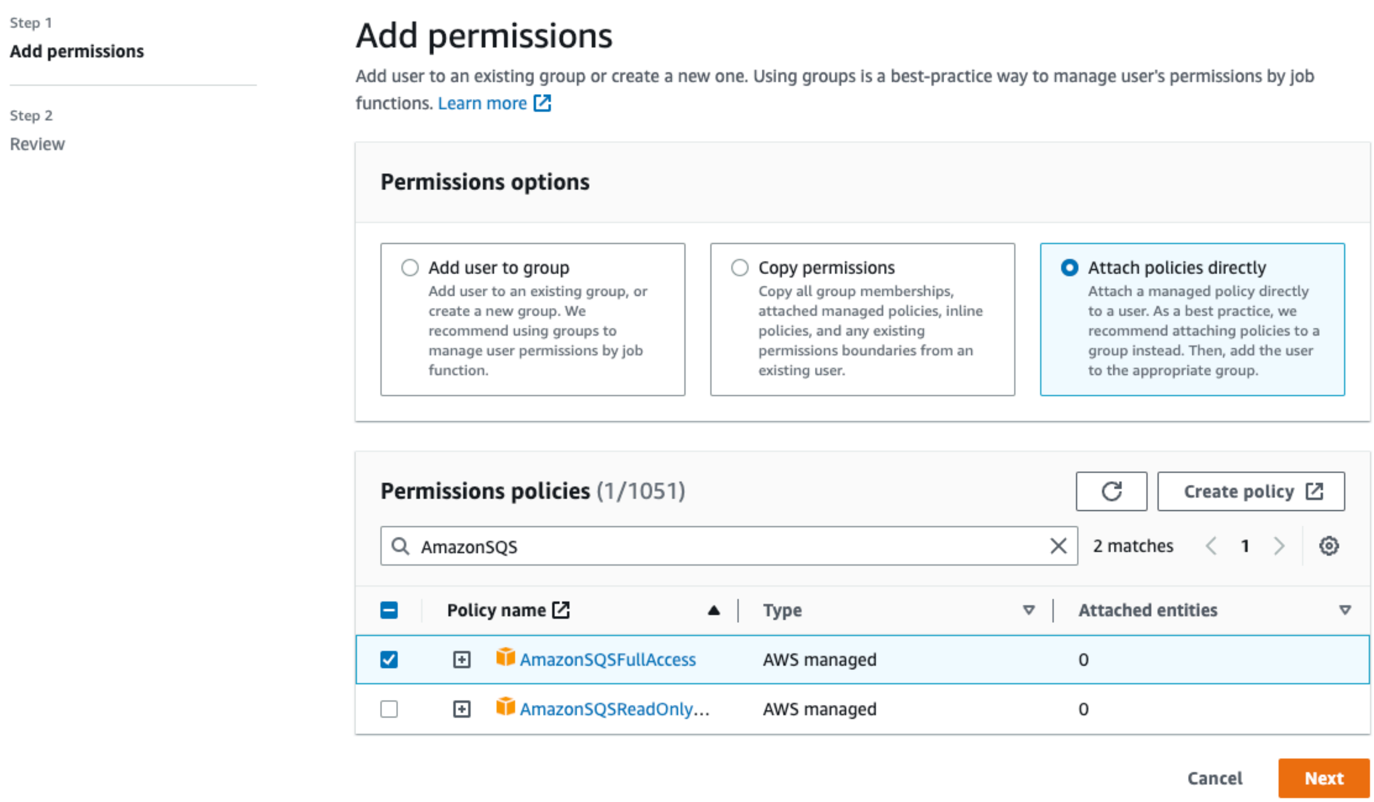
Action-based

```
{
  "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>"
    }
  ]
}
```

Managed policy

The following managed policies contain the required updated permissions:

- **AmazonSQSFullAccess –** Includes the following dead-letter queue redrive tasks: start, cancel, and list.
- **AmazonSQSReadOnlyAccess –** Provides read-only access, and includes the list dead-letter queue redrive task.



Permission Policy that uses sqs* wildcard

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

Identifying impacted policies

If you are using customer managed policies (CMPs), you can use AWS CloudTrail and IAM to identify the policies impacted by the queue permissions update.

> **Note**
>
> If you are using `AmazonSQSFullAccess` and `AmazonSQSReadOnlyAccess`, no further action is required.

1. Sign in to the AWS CloudTrail console.
2. On the **Event history** page, under **Look up attributes**, use the drop down menu to select *Event name*. Then, search for `CreateMoveTask`.
3. Choose an event to open the **Details** page. In the **Event records** section, retrieve the `UserName` or `RoleName` from the `userIdentity` ARN.
4. Sign into IAM console.
   - For users, choose Users. Select the user with the `UserName` identified in the previous step.
   - For roles, choose Roles. Search for the user with the `RoleName` identified in the previous step.
5. On the **Details** page, in the **Permissions** section, review any policies with the `sqs:` prefix in `Action`, or review policies that have Amazon SQS queue defined in `Resource`.

Create alarms for dead-letter queues using Amazon CloudWatch

You can configure an alarm for any messages moved to a dead-letter queue using Amazon CloudWatch and the metric `ApproximateNumberOfMessagesVisible`. For more information, see Creating CloudWatch alarms for Amazon SQS metrics. After you receive an alert that messages have been sent to the dead-letter queue, you can review the messages using polling to receive the message.

Message metadata for Amazon SQS

You can use message attributes to attach custom metadata to Amazon SQS messages for your applications. You can use message system attributes to store metadata for other AWS services, such as AWS X-Ray.

Topics

Amazon SQS message attributes

Amazon SQS lets you include structured metadata (such as timestamps, geospatial data, signatures, and identifiers) with messages using *message attributes*. Each message can have up to 10 attributes. Message attributes are optional and separate from the message body (however, they are sent alongside it). Your consumer can use message attributes to handle a message in a particular way without having to process the message body first. For information about sending messages with attributes using the Amazon SQS console, see Sending a message with attributes using Amazon SQS.

> **Note**
>
> Don't confuse message attributes with *message system attributes*: Whereas you can use message attributes to attach custom metadata to Amazon SQS messages for your applications, you can use message system attributes to store metadata for other AWS services, such as AWS X-Ray.

Topics

Message attribute components

> **Important**
>
> All components of a message attribute are included in the 256 KB message size restriction. The `Name`, `Type`, `Value`, and the message body must not be empty or null.

Each message attribute consists of the following components:

- **Name** – The message attribute name can contain the following characters: `A-Z`, `a-z`, `0-9`, underscore (_), hyphen (-), and period (`.`). The following restrictions apply:
  - Can be up to 256 characters long
  - Can't start with `AWS.` or `Amazon.` (or any casing variations)
  - Is case-sensitive
  - Must be unique among all attribute names for the message
  - Must not start or end with a period
  - Must not have periods in a sequence
- **Type** – The message attribute data type. Supported types include `String`, `Number`, and `Binary`. You can also add custom information for any data type. The data type has the same restrictions as the message body (for more information, see `SendMessage` in the *Amazon Simple Queue Service API Reference*). In addition, the following restrictions apply:
  - Can be up to 256 characters long
  - Is case-sensitive
- **Value** – The message attribute value. For `String` data types, the attribute values has the same restrictions as the message body.

Message attribute data types

Message attribute data types instruct Amazon SQS how to handle the corresponding message attribute values. For example, if the type is `Number`, Amazon SQS validates numerical values.

Amazon SQS supports the logical data types `String`, `Number`, and `Binary` with optional custom data type labels with the format `.custom-data-type`

- **String** – `String` attributes can store Unicode text using any valid XML characters.
- **Number** – `Number` attributes can store positive or negative numerical values. A number can have up to 38 digits of precision, and it can be between 10^-128 and 10^+126.

> **Note**
>
> Amazon SQS removes leading and trailing zeroes.

- **Binary** – Binary attributes can store any binary data such as compressed data, encrypted data, or images.

- **Custom** – To create a custom data type, append a custom-type label to any data type. For example:
  - `Number.byte`, `Number.short`, `Number.int`, and `Number.float` can help distinguish between number types.
  - `Binary.gif` and `Binary.png` can help distinguish between file types.

> **Note**
>
> Amazon SQS doesn't interpret, validate, or use the appended data.
> The custom-type label has the same restrictions as the message body.

Calculating the MD5 message digest for message attributes

If you use the AWS SDK for Java, you can skip this section. The `MessageMD5ChecksumHandler` class of the SDK for Java supports MD5 message digests for Amazon SQS message attributes.

If you use either the Query API or one of the AWS SDKs that doesn't support MD5 message digests for Amazon SQS message attributes, you must use the following guidelines to perform the MD5 message digest calculation.

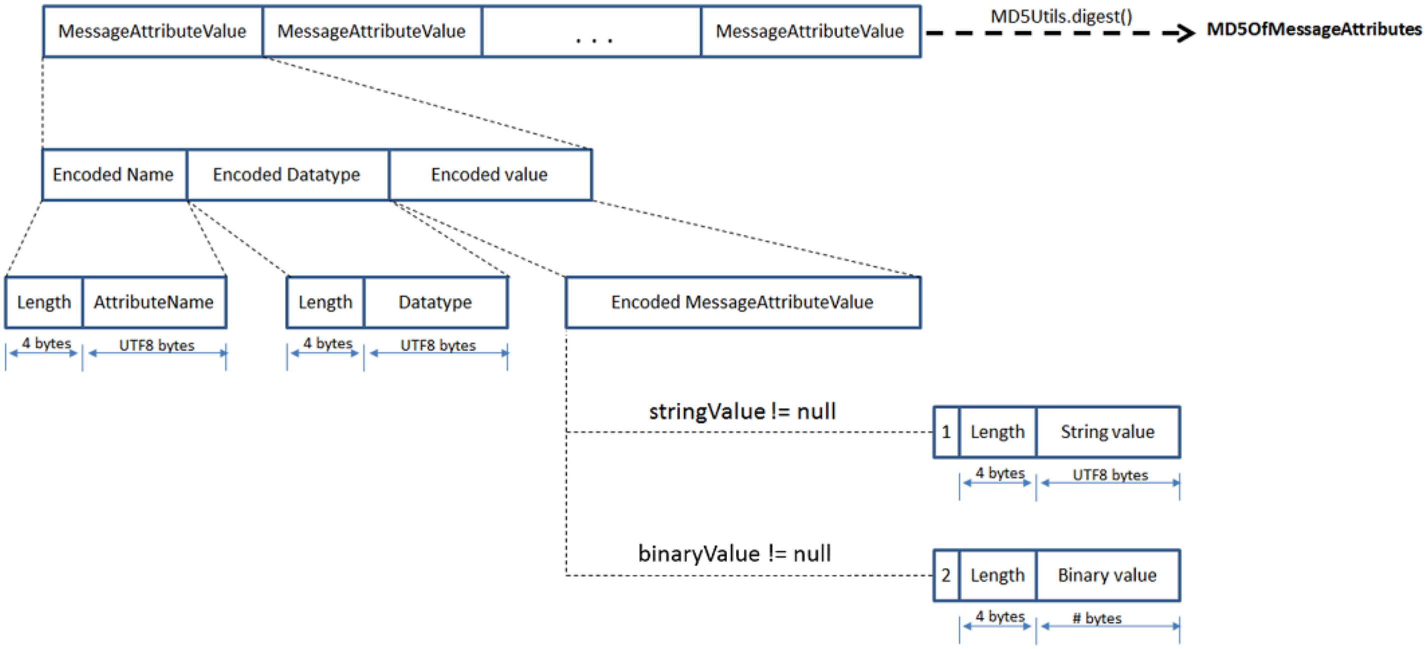
> **Note**
>
> Always include custom data type suffixes in the MD5 message-digest calculation.

Overview

The following is an overview of the MD5 message digest calculation algorithm:

1. Sort all message attributes by name in ascending order.
2. Encode the individual parts of each attribute (`Name`, `Type`, and `Value`) into a buffer.
3. Compute the message digest of the entire buffer.

The following diagram shows the encoding of the MD5 message digest for a single message attribute:



To encode a single Amazon SQS message attribute

1. Encode the name: the length (4 bytes) and the UTF-8 bytes of the name.
2. Encode the data type: the length (4 bytes) and the UTF-8 bytes of the data type.
3. Encode the transport type (`String` or `Binary`) of the value (1 byte).

> **Note**
>
> The logical data types `String` and `Number` use the `String` transport type. The logical data type `Binary` uses the `Binary` transport type.

   a. For the `String` transport type, encode 1.
   b. For the `Binary` transport type, encode 2.

4. Encode the attribute value.

   a. For the `String` transport type, encode the attribute value: the length (4 bytes) and the UTF-8 bytes of the value.
   b. For the `Binary` transport type, encode the attribute value: the length (4 bytes) and the raw bytes of the value.

Amazon SQS message system attributes

Whereas you can use message attributes to attach custom metadata to Amazon SQS messages for your applications, you can use *message system attributes* to store metadata for other AWS services, such as AWS X-Ray. For more information, see the `MessageSystemAttribute` request parameter of the `SendMessage` and `SendMessageBatch` API actions, the `AWSTraceHeader` attribute of the `ReceiveMessage` API action, and the `MessageSystemAttributeValue` data type in the *Amazon Simple Queue Service API Reference*.

Message system attributes are structured exactly like message attributes, with the following exceptions:

- Currently, the only supported message system attribute is `AWSTraceHeader`. Its type must be `String` and its value must be a correctly formatted AWS X-Ray trace header string.
- The size of a message system attribute doesn't count towards the total size of a message.

Resources required to process Amazon SQS messages

To help you estimate the resources you need to process queued messages, Amazon SQS can determine the approximate number of delayed, visible, and not visible messages in a queue. For more information about visibility, see Amazon SQS visibility timeout.

Note

For some metrics, the result is approximate because of the distributed architecture of Amazon SQS. In most cases, the count should be close to the actual number of messages in the queue.

The following table lists the attribute name to use with the `GetQueueAttributes` action:

Task	Attribute name
Get the approximate number of messages available for retrieval from the queue.	`ApproximateNumberOfMessagesVisible`
Get the approximate number of messages in the queue that are delayed and not available	`ApproximateNumberOfMessagesDelayed`

Task	Attribute name
for reading immediately. This can happen when the queue is configured as a delay queue or when a message has been sent with a delay parameter.	
Get the approximate number of messages that are in flight. Messages are considered to be *in flight* if they have been sent to a client but have not yet been deleted or have not yet reached the end of their visibility window.	`ApproximateNumberOfMessages NotVisible`

Amazon SQS list queue pagination

The `listQueues` and `listDeadLetterQueues` API methods support optional pagination controls. By default, these API methods return up to 1000 queues in the response message. You can set the `MaxResults` parameter to return fewer results in each response.

Set parameter `MaxResults` in the [`listQueues`] or [`listDeadLetterQueues`] request to specify the maximum number of results to be returned in the response. If you do not set `MaxResults`, the response includes a maximum of 1,000 results and the `NextToken` value in the response is null.

If you set `MaxResults`, the response includes a value for `NextToken` if there are additional results to display. Use `NextToken` as a parameter in your next request to `listQueues` to receive the next page of results. If there are no additional results to display, the `NextToken` value in the response is null.

Amazon SQS cost allocation tags

To organize and identify your Amazon SQS queues for cost allocation, you can add metadata *tags* that identify a queue's purpose, owner, or environment. This is especially useful when you have many queues. To configure tags using the Amazon SQS console, see the section called “Configuring tags for a queue”

You can use cost allocation tags to organize your AWS bill to reflect your own cost structure. To do this, sign up to get your AWS account bill to include tag keys and values. For more information, see Setting Up a Monthly Cost Allocation Report in the *AWS Billing User Guide*.

Each tag consists of a key-value pair that you define. For example, you can easily identify your *production* and *testing* queues if you tag your queues as follows:

Queue	Key	Value
`MyQueueA`	`QueueType`	`Production`
`MyQueueB`	`QueueType`	`Testing`

> **Note**
>
> When you use queue tags, keep the following guidelines in mind:
>
> - We don't recommend adding more than 50 tags to a queue. Tagging supports Unicode characters in UTF-8.
> - Tags don't have any semantic meaning. Amazon SQS interprets tags as character strings.
> - Tags are case-sensitive.
> - A new tag with a key identical to that of an existing tag overwrites the existing tag.
> - Tagging actions are limited to 30 TPS per AWS account. If your application requires a higher throughput, submit a request.
>
> For a full list of tag restrictions, see Amazon SQS standard queue quotas.

Amazon SQS short and long polling

Amazon SQS offers short and long polling options for receiving messages from a queue. Consider your application's requirements for responsiveness and cost efficiency when choosing between these two polling options:

- **Short polling** (default) – The `ReceiveMessage` request queries a subset of servers (based on a weighted random distribution) to find available messages and sends an immediate response, even if no messages are found.
- **Long polling** – `ReceiveMessage` queries all servers for messages, sending a response once at least one message is available, up to the specified maximum. An empty response is sent only

if the polling wait time expires. This option can reduce the number of empty responses and potentially lower costs.

The following sections explain the details of short polling and long polling.

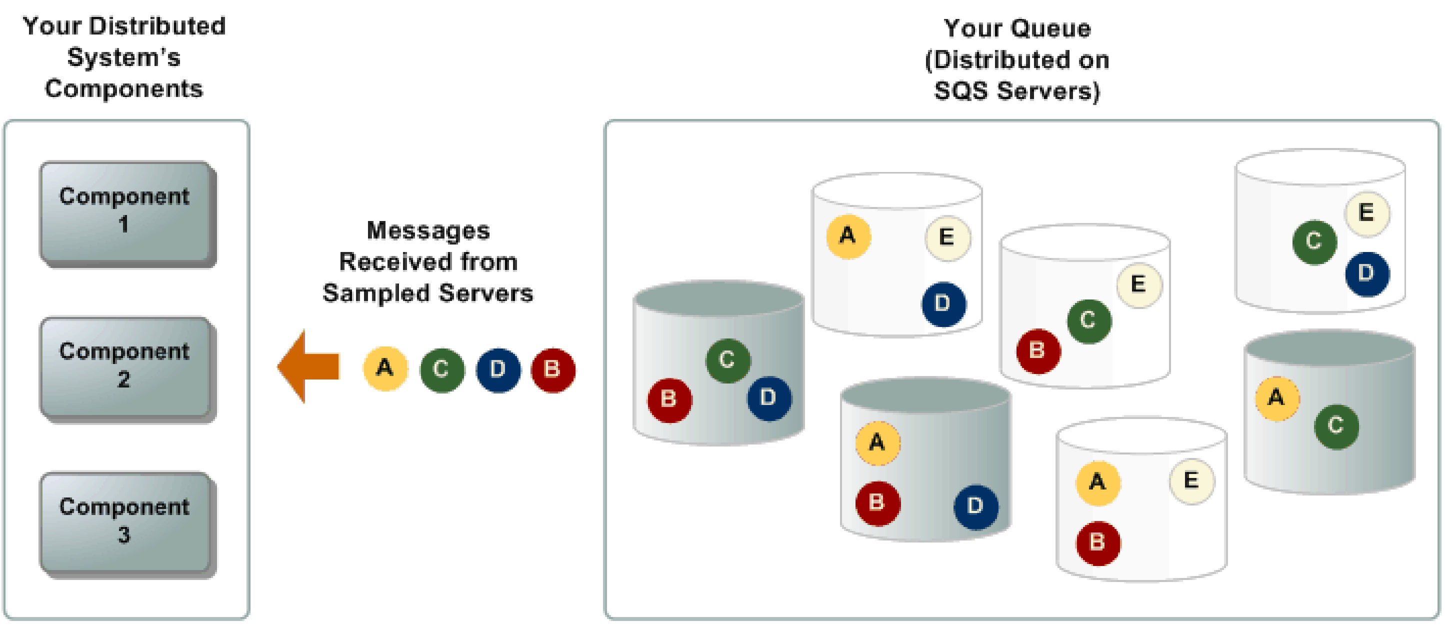
Topics

- Consuming messages using short polling
- Consuming messages using long polling
- Differences between long and short polling

Consuming messages using short polling

When you consume messages from a queue (FIFO or standard) using short polling, Amazon SQS samples a subset of its servers (based on a weighted random distribution) and returns messages from only those servers. Thus, a particular `ReceiveMessage` request might not return all of your messages. However, if you have fewer than 1,000 messages in your queue, a subsequent request will return your messages. If you keep consuming from your queues, Amazon SQS samples all of its servers, and you receive all of your messages.

The following diagram shows the short-polling behavior of messages returned from a standard queue after one of your system components makes a receive request. Amazon SQS samples several of its servers (in gray) and returns messages A, C, D, and B from these servers. Message E isn't returned for this request, but is returned for a subsequent request.



Consuming messages using long polling

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 (when there are no messages available for a `ReceiveMessage` request) and false empty responses (when messages are available but aren't included in a response). For information about enabling long polling for a new or existing queue using the Amazon SQS console, see the Configuring queue parameters using the Amazon SQS console. For best practices, see Setting-up long polling in Amazon SQS.

Long polling offers the following benefits:

- Reduce empty responses by allowing Amazon SQS to wait until a message is available in a queue before sending a response. Unless the connection times out, the response to the `ReceiveMessage` request contains at least one of the available messages, up to the maximum number of messages specified in the `ReceiveMessage` action. In rare cases, you might receive empty responses even when a queue still contains messages, especially if you specify a low value for the `ReceiveMessageWaitTimeSeconds` parameter.
- Reduce false empty responses by querying all—rather than a subset of—Amazon SQS servers.
- Return messages as soon as they become available.

For information about how to confirm that a queue is empty, see Confirming that an Amazon SQS queue is empty.

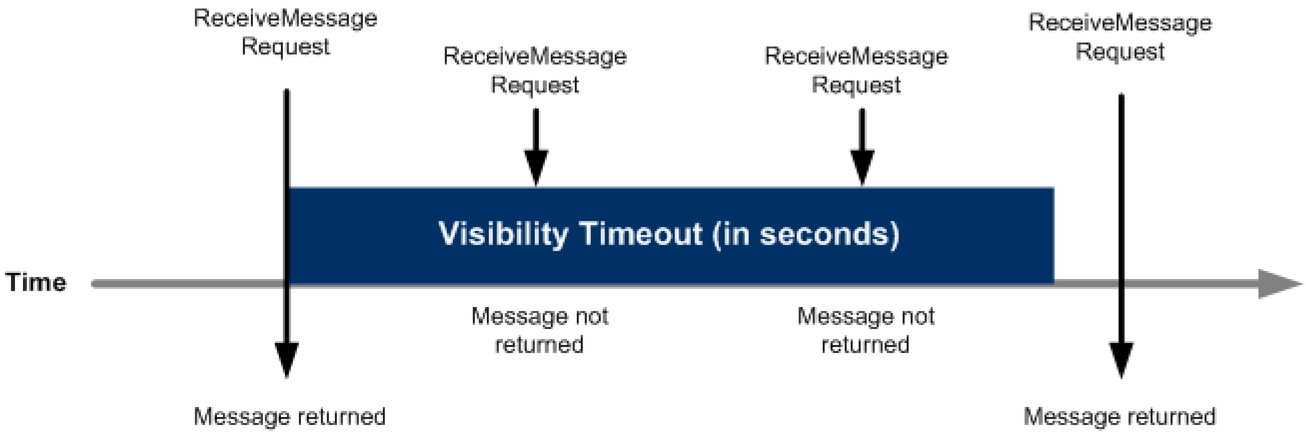
Differences between long and short polling

Short polling occurs when the `WaitTimeSeconds` parameter of a `ReceiveMessage` request is set to `0` in one of two ways:

- The `ReceiveMessage` call sets `WaitTimeSeconds` to `0`.
- The `ReceiveMessage` call doesn't set `WaitTimeSeconds`, but the queue attribute `ReceiveMessageWaitTimeSeconds` is set to `0`.

Amazon SQS visibility timeout

When a consumer receives and processes a message from a queue, the message remains in the queue. Amazon SQS doesn't automatically delete the message. Because Amazon SQS is a distributed system, there's no guarantee that the consumer actually receives the message (for example, due to a connectivity issue, or due to an issue in the consumer application). Thus, the consumer must delete the message from the queue after receiving and processing it.



Immediately after a message is received, it remains in the queue. To prevent other consumers from processing the message again, Amazon SQS sets a *visibility timeout*, a period of time during which Amazon SQS prevents all consumers from receiving and processing the message. The default visibility timeout for a message is 30 seconds. The minimum is 0 seconds. The maximum is 12 hours. For information about configuring visibility timeout for a queue using the console, see Configuring queue parameters using the Amazon SQS console.

> **Note**
>
> For standard queues, the visibility timeout isn't a guarantee against receiving a message twice. For more information, see Amazon SQS at-least-once delivery.
> FIFO queues allow the producer or consumer to attempt multiple retries:
>
> - If the producer detects a failed `SendMessage` action, it can retry sending as many times as necessary, using the same message deduplication ID. Assuming that the producer receives at least one acknowledgement before the deduplication interval expires, multiple retries neither affect the ordering of messages nor introduce duplicates.
> - If the consumer detects a failed `ReceiveMessage` action, it can retry as many times as necessary, using the same receive request attempt ID. Assuming that the consumer receives at least one acknowledgement before the visibility timeout expires, multiple retries don't affect the ordering of messages.
> - When you receive a message with a message group ID, no more messages for the same message group ID are returned unless you delete the message or it becomes visible.

Topics

In flight messages

An Amazon SQS message has three basic states:

1. Sent to a queue by a producer.
2. Received from the queue by a consumer.
3. Deleted from the queue.

A message is considered to be *stored* after it is sent to a queue by a producer, but not yet received from the queue by a consumer (that is, between states 1 and 2). There is no quota to the number of stored messages. A message is considered to be *in flight* after it is received from a queue by a consumer, but not yet deleted from the queue (that is, between states 2 and 3). There is a quota to the number of in flight messages.

> **Important**
>
> Quotas that apply to in flight messages are unrelated to the *unlimited* number of stored messages.

For most standard queues (depending on queue traffic and message backlog), there can be a maximum of approximately 120,000 in flight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this quota while using short polling, Amazon SQS returns the `OverLimit` error message. If you use long polling, Amazon SQS returns no error messages. To avoid reaching the quota, you should delete messages from the queue after they're processed. You can also increase the number of queues you use to process your messages. To request a quota increase, submit a support request.

For FIFO queues, there can be a maximum of 20,000 in flight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this quota, Amazon SQS returns no error messages.

> **Important**
>
> When working with FIFO queues, `DeleteMessage` operations will fail if the request is received outside of the visibility timeout window. If the visibility timeout is 0 seconds, the message must be deleted within the same millisecond it was sent, or it is considered abandoned. This can cause Amazon SQS to include duplicate messages in the same response to a `ReceiveMessage` operation if the `MaxNumberOfMessages` parameter is greater than 1. For additional details see How the Amazon SQS FIFO API Works.

Setting the visibility timeout

The visibility timeout begins when Amazon SQS returns a message. During this time, the consumer processes and deletes the message. However, if the consumer fails before deleting the message and your system doesn't call the `DeleteMessage` action for that message before the visibility timeout expires, the message becomes visible to other consumers and the message is received again. If a message must be received only once, your consumer should delete it within the duration of the visibility timeout.

Every Amazon SQS queue has the default visibility timeout setting of 30 seconds. You can change this setting for the entire queue. Typically, you should set the visibility timeout to the maximum time that it takes your application to process and delete a message from the queue. When receiving messages, you can also set a special visibility timeout for the returned messages without changing the overall queue timeout. For more information, see the best practices in the Processing messages in a timely manner in Amazon SQS section.

If you don't know how long it takes to process a message, create a *heartbeat* for your consumer process: Specify the initial visibility timeout (for example, 2 minutes) and then—as long as your consumer still works on the message—keep extending the visibility timeout by 2 minutes every minute.

Important

The maximum visibility timeout is 12 hours from the time that Amazon SQS receives the `ReceiveMessage` request. Extending the visibility timeout does not reset the 12 hour maximum.

Additionally, you may be unable to set the timeout on an individual message to the full 12 hours (e.g. 43,200 seconds) since the `ReceiveMessage` request initiates the timer. For example, if you receive a message and immediately set the 12 hour maximum by sending a `ChangeMessageVisibility` call with `VisibilityTimeout` equal to 43,200 seconds, it will likely fail. However, using a value of 43,195 seconds will work unless there is a significant delay between requesting the message via `ReceiveMessage` and updating the visibility timeout. If your consumer needs longer than 12 hours, consider using Step Functions.

Changing the visibility timeout for a message

When you receive a message from a queue and begin to process it, the visibility timeout for the queue may be insufficient (for example, you might need to process and delete a message). You can shorten or extend a message's visibility by specifying a new timeout value using the [`ChangeMessageVisibility`] action.

For example, if the default timeout for a queue is 60 seconds, 15 seconds have elapsed since you received the message, and you send a `ChangeMessageVisibility` call with `VisibilityTimeout` set to 10 seconds, the 10 seconds begin to count from the time that you make the `ChangeMessageVisibility` call. Thus, any attempt to change the visibility timeout or to delete that message 10 seconds after you initially change the visibility timeout (a total of 25 seconds) might result in an error.

Note

The new timeout period takes effect from the time you call the `ChangeMessageVisibility` action. In addition, the new timeout period applies only to the particular receipt of the message. `ChangeMessageVisibility` doesn't affect the timeout of later receipts of the message or later queues.

Terminating the visibility timeout for a message

When you receive a message from a queue, you might find that you actually don't want to process and delete that message. Amazon SQS allows you to terminate the visibility timeout for a specific message. This makes the message immediately visible to other components in the system and available for processing.

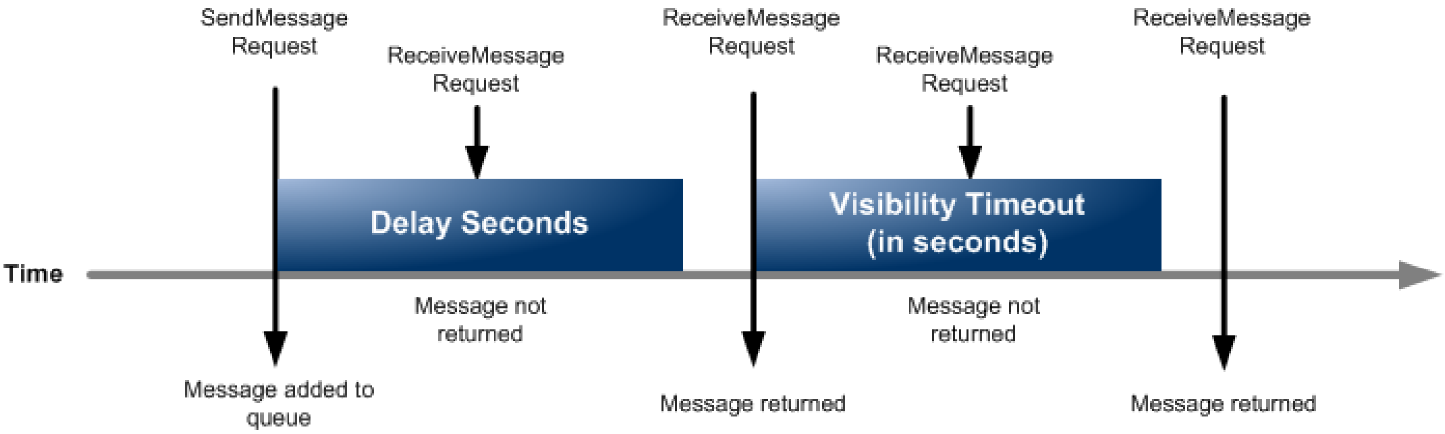
To terminate a message's visibility timeout after calling `ReceiveMessage`, call `ChangeMessageVisibility` with `VisibilityTimeout` set to 0 seconds.

Amazon SQS delay queues

Delay queues let you postpone the delivery of new messages to consumers for a number of seconds, for example, when your consumer application needs additional time to process messages. If you create a delay queue, any messages that you send to the queue remain invisible to consumers for the duration of the delay period. The default (minimum) delay for a queue is 0 seconds. The maximum is 15 minutes. For information about configuring delay queues using the console see Configuring queue parameters using the Amazon SQS console.

> **Note**
>
> For standard queues, the per-queue delay setting is *not retroactive*—changing the setting doesn't affect the delay of messages already in the queue.
> For FIFO queues, the per-queue delay setting is *retroactive*—changing the setting affects the delay of messages already in the queue.

Delay queues are similar to visibility timeouts because both features make messages unavailable to consumers for a specific period of time. The difference between the two is that, for delay queues, a message is hidden *when it is first added to queue*, whereas for visibility timeouts a message is hidden *only after it is consumed from the queue*. The following diagram illustrates the relationship between delay queues and visibility timeouts.



To set delay seconds on *individual messages*, rather than on an entire queue, use [message timers](#) to allow Amazon SQS to use the message timer's `DelaySeconds` value instead of the delay queue's `DelaySeconds` value.

Amazon SQS temporary queues

Temporary queues help you save development time and deployment costs when using common message patterns such as *request-response*. You can use the [Temporary Queue Client](#) to create high-throughput, cost-effective, application-managed temporary queues.

The client maps multiple *temporary queues*—application-managed queues created on demand for a particular process—onto a single Amazon SQS queue automatically. This allows your application to make fewer API calls and have a higher throughput when the traffic to each temporary queue is low. When a temporary queue is no longer in use, the client cleans up the temporary queue automatically, even if some processes that use the client aren't shut down cleanly.

The following are the benefits of temporary queues:

- They serve as lightweight communication channels for specific threads or processes.
- They can be created and deleted without incurring additional cost.
- They are API-compatible with static (normal) Amazon SQS queues. This means that existing code that sends and receives messages can send messages to and receive messages from virtual queues.

Topics

- [Virtual queues](#)

Virtual queues

Virtual queues are local data structures that the Temporary Queue Client creates. Virtual queues let you combine multiple low-traffic destinations into a single Amazon SQS queue. For best practices, see Avoid reusing the same message group ID with virtual queues.

> **Note**
>
> - Creating a virtual queue creates only temporary data structures for consumers to receive messages in. Because a virtual queue makes no API calls to Amazon SQS, virtual queues incur no cost.
> - TPS quotas apply to all virtual queues across a single host queue. For more information, see Amazon SQS message quotas.

The `AmazonSQSVirtualQueuesClient` wrapper class adds support for attributes related to virtual queues. To create a virtual queue, you must call the `CreateQueue` API action using the `HostQueueURL` attribute. This attribute specifies the existing queue that hosts the virtual queues.

The URL of a virtual queue is in the following format.

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

When a producer calls the `SendMessage` or `SendMessageBatch` API action on a virtual queue URL, the Temporary Queue Client does the following:

1. Extracts the virtual queue name.
2. Attaches the virtual queue name as an additional message attribute.
3. Sends the message to the host queue.

While the producer sends messages, a background thread polls the host queue and sends received messages to virtual queues according to the corresponding message attributes.

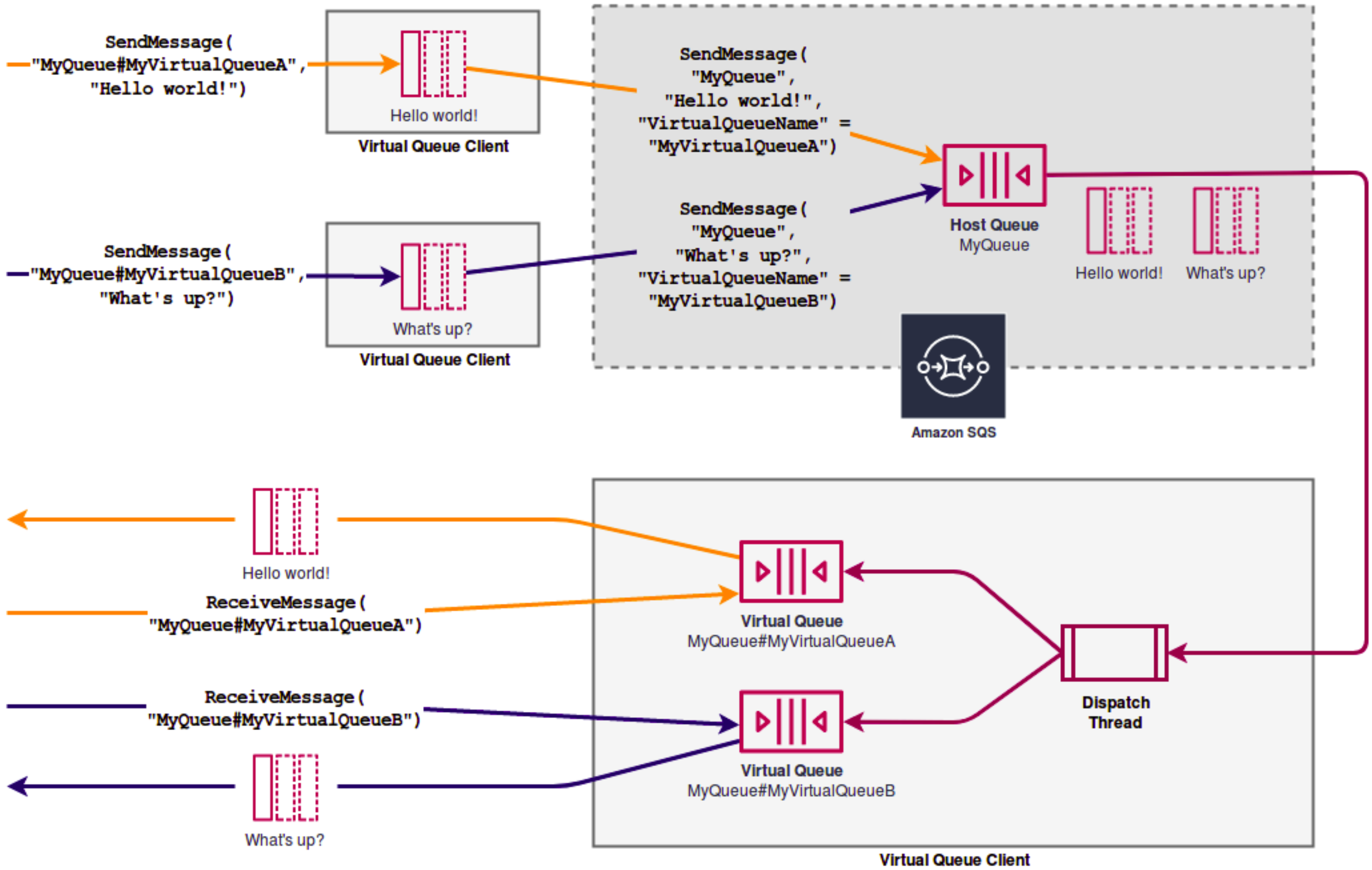
While the consumer calls the `ReceiveMessage` API action on a virtual queue URL, the Temporary Queue Client blocks the call locally until the background thread sends a message into the virtual queue. (This process is similar to message prefetching in the Buffered Asynchronous Client: a single API action can provide messages to up to 10 virtual queues.) Deleting a virtual queue removes any client-side resources without calling Amazon SQS itself.

The `AmazonSQSTemporaryQueuesClient` class turns all queues it creates into temporary queues automatically. It also creates host queues with the same queue attributes automatically, on demand. These queues' names share a common, configurable prefix (by default, `__RequesterClientQueues__`) that identifies them as temporary queues. This allows the client to act as a drop-in replacement that optimizes existing code which creates and deletes queues. The client also includes the `AmazonSQSRequester` and `AmazonSQSResponder` interfaces that allow two-way communication between queues.

Request-response messaging pattern (virtual queues)

The most common use case for temporary queues is the *request-response* messaging pattern, where a requester creates a *temporary queue* for receiving each response message. To avoid creating an Amazon SQS queue for each response message, the Temporary Queue Client lets you create and delete multiple temporary queues without making any Amazon SQS API calls. For more information, see Implementing request-response systems.

The following diagram shows a common configuration using this pattern.



Example scenario: Processing a login request

The following example scenario shows how you can use the `AmazonSQSRequester` and `AmazonSQSResponder` interfaces to process a user's login request.

On the client side

```
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();
    }
}
```

Sending a login request does the following:

1. Creates a temporary queue.
2. Attaches the temporary queue's URL to the message as an attribute.
3. Sends the message.
4. Receives a response from the temporary queue.
5. Deletes the temporary queue.
6. Returns the response.

On the server side

The following example assumes that, upon construction, a thread is created to poll the queue and call the `handleLoginRequest()` method for every message. In addition, `doLogin()` is an assumed method.

```
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));
    }
}
```

Cleaning up queues

To make sure that Amazon SQS reclaims any in-memory resources used by virtual queues, when your application no longer needs the Temporary Queue Client, it should call the `shutdown()` method. You can also use the `shutdown()` method of the `AmazonSQSRequester` interface.

The Temporary Queue Client also provides a way to eliminate orphaned host queues. For each queue that receives an API call over a period of time (by default, five minutes), the client uses the `TagQueue` API action to tag a queue that remains in use.

> **Note**
>
> Any API action taken on a queue marks it as non-idle, including a `ReceiveMessage` action that returns no messages.

The background thread uses the `ListQueues` and `ListTags` API actions to check all queues with the configured prefix, deleting any queues that haven't been tagged for at least five minutes. In this way, if one client doesn't shut down cleanly, the other active clients clean up after it. In order to reduce the duplication of work, all clients with the same prefix communicate through a shared, internal work queue named after the prefix.

Amazon SQS message timers

Message timers let you specify an initial invisibility period for a message added to a queue. For example, if you send a message with a 45-second timer, the message isn't visible to consumers for its first 45 seconds in the queue. The default (minimum) delay for a message is 0 seconds. The maximum is 15 minutes. For information about sending messages with timers using the console, see Send a message.

> **Note**
>
> FIFO queues don't support timers on individual messages.

To set a delay period on *an entire queue*, rather than on individual messages, use delay queues. A message timer setting for an individual message overrides any `DelaySeconds` value on an Amazon SQS delay queue.

Accessing Amazon EventBridge Pipes through the Amazon SQS console

Amazon EventBridge Pipes connect sources to targets. Pipes are intended for point-to-point integrations between supported sources and targets, with support for advanced transformations and enrichment. EventBridge Pipes provide a highly scalable way to connect your Amazon SQS queue to AWS services such as Step Functions, Amazon SQS, and API Gateway, as well as third-party software as a service (SaaS) applications like Salesforce.

To set up a pipe, you choose the source, add optional filtering, define optional enrichment, and choose the target for the event data.

On the details page for an Amazon SQS queue, you can view the pipes that use that queue as their source. From there, you can also:

- Launch the EventBridge console to view pipe details.
- Launch the EventBridge console to create a new pipe with the queue as its source.

For more information on configuring an Amazon SQS queue as a pipe source, see Amazon SQS queue as a source in the *Amazon EventBridge User Guide*. For more information about EventBridge Pipes in general, see EventBridge Pipes.

To access EventBridge pipes for a given Amazon SQS queue

1. Open the Queues page of the Amazon SQS console.
2. Select a queue.
3. On the queue detail page, choose the **EventBridge Pipes** tab.

   The **EventBridge Pipes** tab includes a list of any pipes currently configured to use the selected queue as a source, including:

   - pipe name
   - current status
   - pipe target
   - when the pipe was last modified
4. View more pipe details or create a new pipe, if desired:

   - **To access more details about a pipe:**

     Choose the pipe name.

     This launches the **Pipe details** page of the EventBridge console.
   - **To create a new pipe:**

     Choose **Connect Amazon SQS queue to pipe**.

     This launches the **Create pipe** page of the EventBridge console, with the Amazon SQS queue specified as the pipe source. For more information, see Creating an EventBridge pipe in the *Amazon EventBridge User Guide*.

     > **Important**
     >
     > A message on an Amazon SQS queue is read by a single pipe and then deleted from the queue after being processed, *whether or not the message matches the filter you can configured for that pipe*. Proceed with caution when configuring multiple pipes to use the same queue as their source.

Managing large Amazon SQS messages with Extended Client Library and Amazon Simple Storage Service

You can use the Amazon SQS Extended Client Library for Java and the Amazon SQS Extended Client Library for Python to send large messages. This is especially useful for consuming large message payloads, from 256 KB and up to 2 GB. Both libraries save the message payload to an Amazon Simple Storage Service bucket, and send the reference of the stored Amazon S3 object to the Amazon SQS queue.

> **Note**
>
> The Amazon SQS Extended Client Libraries are compatible with both Standard and FIFO queues.

Topics

Managing large Amazon SQS messages using Java and Amazon S3

You can use the Amazon SQS Extended Client Library for Java and Amazon Simple Storage Service (Amazon S3) to manage large Amazon Simple Queue Service (Amazon SQS) messages. This is especially useful for consuming large message payloads, from 256 KB and up to 2 GB. The library saves the message payload to an Amazon S3 bucket and sends a message containing a reference of the stored Amazon S3 object to an Amazon SQS queue.

You can use the Amazon SQS Extended Client Library for Java to do the following:

- Specify whether messages are always stored in Amazon S3 or only when the size of a message exceeds 256 KB
- Send a message that references a single message object stored in an S3 bucket
- Retrieve the message object from an Amazon S3 bucket
- Delete the message object from an Amazon S3 bucket

Prerequisites

The following example uses the AWS Java SDK. To install and set up the SDK, see Set up the AWS SDK for Java in the *AWS SDK for Java Developer Guide*.

Before you run the example code, configure your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the *AWS SDK for Java Developer Guide*.

The SDK for Java and Amazon SQS Extended Client Library for Java require the J2SE Development Kit 8.0 or later.

> **Note**
>
> You can use the Amazon SQS Extended Client Library for Java to manage Amazon SQS messages using Amazon S3 *only* with the AWS SDK for Java. You can't do this with the AWS CLI, the Amazon SQS console, the Amazon SQS HTTP API, or any of the other AWS SDKs.

AWS SDK for Java 2.x Example: Using Amazon S3 to manage large Amazon SQS messages

The following *AWS SDK for Java 2.x* example creates an Amazon S3 bucket with a random name and adds a lifecycle rule to permanently delete objects after 14 days. It also creates a queue named `MyQueue` and sends a random message that is stored in an S3 bucket and is more than 256 KB to the queue. Finally, the code retrieves the message, returns information about it, and then deletes the message, the queue, and the bucket.

```
/*
 * 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 com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;
import org.joda.time.DateTime;
import org.joda.time.format.DateTimeFormat;

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

public class SQSExtendedClientExample {

// Create an Amazon S3 bucket with a random name.
private final static String S3_BUCKET_NAME = 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 AmazonS3 s3 = AmazonS3ClientBuilder.defaultClient();

    /*
     * 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 BucketLifecycleConfiguration.Rule expirationRule =
            new BucketLifecycleConfiguration.Rule();
    expirationRule.withExpirationInDays(14).withStatus("Enabled");
    final BucketLifecycleConfiguration lifecycleConfig =
            new BucketLifecycleConfiguration().withRules(expirationRule);

    // Create the bucket and allow message objects to be stored in the bucket.
```

```
    s3.createBucket(S3_BUCKET_NAME);
    s3.setBucketLifecycleConfiguration(S3_BUCKET_NAME, lifecycleConfig);
    System.out.println("Bucket created and configured.");

    /*
     * Set the Amazon SQS extended client configuration with large payload
     * support enabled.
     */
    final ExtendedClientConfiguration extendedClientConfig =
            new ExtendedClientConfiguration()
                    .withLargePayloadSupportEnabled(s3, S3_BUCKET_NAME);

    final AmazonSQS sqsExtended =
            new AmazonSQSExtendedClient(AmazonSQSClientBuilder
                    .defaultClient(), extendedClientConfig);

    /*
     * Create a long string of characters for the message object which will
     * be stored in the bucket.
     */
    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().toString();
    final CreateQueueRequest createQueueRequest =
            new CreateQueueRequest(QueueName);
    final String myQueueUrl = sqsExtended
            .createQueue(createQueueRequest).getQueueUrl();
    System.out.println("Queue created.");

    // Send the message.
    final SendMessageRequest myMessageRequest =
            new SendMessageRequest(myQueueUrl, myLongString);
    sqsExtended.sendMessage(myMessageRequest);
    System.out.println("Sent the message.");

    // Receive the message.
    final ReceiveMessageRequest receiveMessageRequest =
            new ReceiveMessageRequest(myQueueUrl);
    List<Message> messages = sqsExtended
            .receiveMessage(receiveMessageRequest).getMessages();
```

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

    // Delete the message, the queue, and the bucket.
    final String messageReceiptHandle = messages.get(0).getReceiptHandle();
    sqsExtended.deleteMessage(new DeleteMessageRequest(myQueueUrl,
            messageReceiptHandle));
    System.out.println("Deleted the message.");

    sqsExtended.deleteQueue(new DeleteQueueRequest(myQueueUrl));
    System.out.println("Deleted the queue.");

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

private static void deleteBucketAndAllContents(AmazonS3 client) {

    ObjectListing objectListing = client.listObjects(S3_BUCKET_NAME);

    while (true) {
        for (S3ObjectSummary objectSummary : objectListing
                .getObjectSummaries()) {
            client.deleteObject(S3_BUCKET_NAME, objectSummary.getKey());
        }

        if (objectListing.isTruncated()) {
            objectListing = client.listNextBatchOfObjects(objectListing);
        } else {
            break;
        }
    }

    final VersionListing list = client.listVersions(
            new ListVersionsRequest().withBucketName(S3_BUCKET_NAME));

    for (S3VersionSummary s : list.getVersionSummaries()) {
```

```
        client.deleteVersion(S3_BUCKET_NAME, s.getKey(), s.getVersionId());
    }

    client.deleteBucket(S3_BUCKET_NAME);
}
}
```

AWS SDK for Java 2.x Example: Using Amazon S3 to manage large Amazon SQS messages

The following *AWS SDK for Java 2.x* example creates an Amazon S3 bucket with a random name and adds a lifecycle rule to permanently delete objects after 14 days. It also creates a queue named `MyQueue` and sends a random message that is stored in an S3 bucket and is more than 256 KB to the queue. Finally, the code retrieves the message, returns information about it, and then deletes the message, the queue, and the bucket.

```
/*
 * 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 S3_BUCKET_NAME = 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(S3_BUCKET_NAME).build());

 s3.putBucketLifecycleConfiguration(PutBucketLifecycleConfigurationRequest.builder()
                .bucket(S3_BUCKET_NAME)
                .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, S3_BUCKET_NAME);

        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(S3_BUCKET_NAME).build());

        listObjectsResponse.contents().forEach(object -> {

 client.deleteObject(DeleteObjectRequest.builder().bucket(S3_BUCKET_NAME).key(object.key()).bui
        });
```

```
        ListObjectVersionsResponse listVersionsResponse =
 client.listObjectVersions(ListObjectVersionsRequest.builder().bucket(S3_BUCKET_NAME).build());

        listVersionsResponse.versions().forEach(version -> {

 client.deleteObject(DeleteObjectRequest.builder().bucket(S3_BUCKET_NAME).key(version.key()).ve
        });


 client.deleteBucket(DeleteBucketRequest.builder().bucket(S3_BUCKET_NAME).build());
    }
}
```

You can use Apache Maven to configure and build Amazon SQS Extended Client for your Java project, or to build the SDK itself. Specify individual modules from the SDK that you use in your application.

```
<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>
```

Managing large Amazon SQS messages using Python and Amazon S3

You can use the Amazon Simple Queue Service Extended Client Library for Python and Amazon Simple Storage Service to manage large Amazon SQS messages. This is especially useful for consuming large message payloads, from 256 KB and up to 2 GB. The library saves the message payload to an Amazon S3 bucket and sends a message containing a reference of the stored Amazon S3 object to an Amazon Amazon SQS queue.

You can use the Extended Client Library for Python to do the following:

- Specify whether payloads are always stored in Amazon S3, or only stored in S3 when a payload size exceeds 256 KB
- Send a message that references a single message object stored in an Amazon S3 bucket
- Retrieve the corresponding payload object from an Amazon S3 bucket
- Delete the corresponding payload object from an Amazon S3 bucket

Prerequisites

The following are the prerequisites for using the Amazon SQS Extended Client Library for Python:

- An AWS account with the necessary credentials. To create an AWS account, navigate to the AWS home page , and then choose **Create an AWS Account** . Follow the instructions. For information about credentials, see Credentials.
- An AWS SDK: The example on this page uses AWS Python SDK Boto3. To install and set up the SDK, see the *AWS SDK for Python* documentation in the *AWS SDK for Python Developer Guide*
- Python 3.x (or later) and `pip`.
- The Amazon SQS Extended Client Library for Python, available from PyPI

> **Note**
>
> You can use the Amazon SQS Extended Client Library for Python to manage Amazon SQS messages using Amazon S3 only with the AWS SDK for Python. You can't do this with the

AWS CLI, the Amazon SQS console, the Amazon SQS HTTP API, or any of the other AWS SDKs.

Configuring message storage

The Amazon SQS Extended Client makes uses the following message attributes to configure the Amazon S3 message storage options:

- `large_payload_support`: The Amazon S3 bucket name to store large messages.
- `always_through_s3`: If `True`, then all messages are stored in Amazon S3. If `False`, messages smaller than 256 KB will not be serialized to the s3 bucket. The default is `False`.
- `use_legacy_attribute`: If `True`, all published messages use the Legacy reserved message attribute (`SQSLargePayloadSize`) instead of the current reserved message attribute (`ExtendedPayloadSize`).

Managing large Amazon SQS messages with Extended Client Library for Python

The following example creates an Amazon S3 bucket with a random name. It then creates an Amazon SQS queue named `MyQueue` and sends a message that is stored in an S3 bucket and is more than 256 KB to the queue. Finally, the code retrieves the message, returns information about it, and then deletes the message, the queue, and the bucket.

```
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 = "S3_BUCKET_NAME"
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
```

Configuring Amazon SQS queues using the Amazon SQS console

Use the Amazon SQS console to configure and manage Amazon Simple Queue Service (Amazon SQS) queues and features. You can also use the console to configure features such as server-side encryption, associate a dead-letter queue with your queue, or set a trigger to invoke an AWS Lambda function.

Topics

Attribute-based access control for Amazon SQS

What is ABAC?

Attribute-based access control (ABAC) is an authorization process that defines permissions based on tags that are attached to users and AWS resources. ABAC provides granular and flexible access control based on attributes and values, reduces security risk related to reconfigured role-based policies, and centralizes auditing and access policy management. For more details about ABAC, see What is ABAC for AWS in the *IAM User Guide*.

Amazon SQS supports ABAC by allowing you to control access to your Amazon SQS queues based on the tags and aliases that are associated with an Amazon SQS queue. The tag and alias condition keys that enable ABAC in Amazon SQS authorize IAM principals to use Amazon SQS queues without editing policies or managing grants. To learn more about ABAC condition keys, see Condition keys for Amazon SQS in the *Service Authorization Reference*.

With ABAC, you can use tags to configure IAM access permissions and policies for your Amazon SQS queues, which helps you to scale your permissions management. You can create a single permissions policy in IAM using tags that you add to each business role—without having to update the policy each time you add a new resource. You can also attach tags to IAM principals to create an ABAC policy. You can design ABAC policies to allow Amazon SQS operations when the tag on the IAM user role that's making the call matches the Amazon SQS queue tag. To learn more about tagging in AWS, see AWS Tagging Strategies and Amazon SQS cost allocation tags.

Note

ABAC for Amazon SQS is currently available in all AWS Commercial Regions where Amazon SQS is available, with the following exceptions:

- Asia Pacific (Hyderabad)
- Asia Pacific (Melbourne)
- Europe (Spain)
- Europe (Zurich)

Why should I use ABAC in Amazon SQS?

Here are some benefits of using ABAC in Amazon SQS:

- ABAC for Amazon SQS requires fewer permissions policies. You don't have to create different policies for different job functions. You can use resource and request tags that apply to more than one queue, which reduces operational overhead.
- Use ABAC to scale teams quickly. Permissions for new resources are automatically granted based on tags when resources are appropriately tagged during their creation.
- Use permissions on the IAM principal to restrict resource access. You can create tags for the IAM principal and use them to restrict access to specific actions that match the tags on the IAM principal. This helps you to automate the process of granting request permissions.
- Track who's accessing your resources. You can determine the identity of a session by looking at user attributes in AWS CloudTrail.

Topics

Tagging for access control in Amazon SQS

The following is an example of how to use tags for access control. The IAM policy restricts an IAM user to all Amazon SQS actions for all queues that include a resource tag with the key environment and the value production. For more information, see Attribute-based access control with tags and AWS Organizations.

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

Creating IAM users and Amazon SQS queues

The following examples explain how to create an ABAC policy to control access to Amazon SQS using the AWS Management Console and AWS CloudFormation.

Using the AWS Management Console

Create an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. Choose **User** from the left navigation pane.
3. Choose **Add Users** and enter a name in the **User name** text box.
4. Select the **Access key - Programmatic access** box and choose **Next:Permissions**.
5. Choose **Next:Tags**.
6. Add the tag key as `environment` and the tag value as `beta`.
7. Choose **Next:Review** and then choose **Create user**.
8. Copy and store the access key ID and secret access key in a secure location.

Add IAM user permissions

1. Select the IAM user that you created.
2. Choose **Add inline policy**.
3. On the JSON tab, paste the following policy:

```
{
  "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"
        }
      }
    }
  ]
}
```

4. Choose **Review policy**.
5. Choose **Create policy**.

Using AWS CloudFormation

Use the following sample AWS CloudFormation template to create an IAM user with an inline policy attached and an Amazon SQS queue:

```
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"
```

Testing attribute-based access control in Amazon SQS

The following examples show you how to test attribute-based access control in Amazon SQS.

Create a queue with the tag key set to environment and the tag value set to prod

Run this AWS CLI command to test creating the queue with the tag key set to environment and the tag value set to prod. If you don't have AWS CLI, you can download and configure it for your machine.

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

You receive an `AccessDenied` error from the Amazon SQS endpoint:

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

This is because the tag value on the IAM user does not match the tag passed in the `CreateQueue` API call. Remember that we applied a tag to the IAM user with the key set to `environment` and the value set to `beta`.

Create a queue with the tag key set to environment and the tag value set to beta

Run the this CLI command to test creating a queue with the tag key set to `environment` and the tag value set to `beta`.

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

You receive a message confirming the successful creation of the queue, similar to the one below.

```
{
"QueueUrl": "<queueUrl>“
}
```

Sending a message to a queue

Run this CLI command to test sending a message to a queue.

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

The response shows a successful message delivery to the Amazon SQS queue. The IAM user permission allows you to send a message to a queue that has a `beta` tag. The response includes `MD5OfMessageBody` and `MessageId` containing the message.

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

Configuring queue parameters using the Amazon SQS console

When you create or edit a queue, you can configure the following parameters:

- **Visibility timeout** – The length of time that a message received from a queue (by one consumer) won't be visible to the other message consumers. For more information, see Visibility timeout.

> **Note**
>
> Using the console to configure the visibility timeout configures the timeout value for all of the messages in the queue. To configure the timeout for single or multiple messages, you must use one of the AWS SDKs.

- **Message retention period** – The amount of time that Amazon SQS retains messages that remain in the queue. By default, the queue retains messages for four days. You can configure a queue to retain messages for up to 14 days. For more information, see Message retention period.

- **Delivery delay** – The amount of time that Amazon SQS will delay before delivering a message that is added to the queue. For more information, see Delivery delay.
- **Maximum message size** – The maximum message size for this queue. For more information, see Maximum message size.
- **Receive message wait time** – The maximum amount of time that Amazon SQS waits for messages to become available after the queue gets a receive request. For more information, see Amazon SQS short and long polling.
- **Enable content-based deduplication** – Amazon SQS can automatically create deduplication IDs based on the body of the message. For more information, see Amazon SQS FIFO queues.
- **Enable high throughput FIFO** – Use to enable high throughput for messages in the queue. Choosing this option changes the related options (Deduplication scope and FIFO throughput limit) to the required settings for enabling high throughput for FIFO queues. For more information, see High throughput for FIFO queues in Amazon SQS and Amazon SQS message quotas.
- **Redrive allow policy**: defines which source queues can use this queue as the dead-letter queue. For more information, see Using dead-letter queues in Amazon SQS .

To configure queue parameters for an existing queue (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**. Choose a queue and choose **Edit**.
3. Scroll to the **Configuration** section.
4. For **Visibility timeout** , enter the duration and units. The range is 0 seconds to 12 hours. The default value is 30 seconds.
5. For **Message retention period**, enter the duration and units. The range is 1 minute to 14 days. The default value is 4 days.
6. For a standard queue, enter a value for **Receive message wait time**. The range is 0 to 20 seconds. The default value is 0 seconds, which sets short polling. Any non-zero value sets long polling.
7. For **Delivery delay**, enter the duration and units. The range is 0 seconds to 15 minutes. The default value is 0 seconds.
8. For **Maximum message size**, enter a value. The range is 1 KB to 256 KB. The default value is 256 KB.

9. For a FIFO queue, choose **Enable content-based deduplication** to enable content-based deduplication. The default setting is disabled.

10. (Optional) For a FIFO queue to enable higher throughput for sending and receiving messages in the queue, choose **Enable high throughput FIFO**.

    Choosing this option changes the related options (**Deduplication scope** and **FIFO throughput limit**) to the required settings for enabling high throughput for FIFO queues. If you change any of the settings required for using high throughput FIFO, normal throughput is in effect for the queue, and deduplication occurs as specified. For more information, see High throughput for FIFO queues in Amazon SQS and Amazon SQS message quotas.

11. For **Redrive allow policy**, choose **Enabled**. Select from the following: **Allow all** (default), **By queue** or **Deny all**. When choosing **By queue**, specify a list of up to 10 source queues by the Amazon Resource Name (ARN).

12. When you finish configuring the queue parameters, choose **Save**.

Configuring an access policy in Amazon SQS

When you edit a queue, you can configure its access policy.

The access policy defines the accounts, users, and roles that can access the queue. The access policy also defines the actions (such as `SendMessage`, `ReceiveMessage`, or `DeleteMessage`) that the users can access. The default policy allows only the queue owner to send and receive messages.

To configure the access policy for an existing queue (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. Choose a queue and choose **Edit**.
4. Scroll to the **Access policy** section.
5. Edit the access policy statements in the input box. For more on access policy statements, see Identity and access management in Amazon SQS.
6. When you finish configuring the access policy, choose **Save**.

Configuring server-side encryption for a queue using SQS-managed encryption keys

In addition to the default Amazon SQS managed server-side encryption (SSE) option, Amazon SQS managed SSE (SSE-SQS) lets you create custom managed server-side encryption that uses SQS-managed encryption keys to protect sensitive data sent over message queues. With SSE-SQS, you don't need to create and manage encryption keys, or modify your code to encrypt your data. SSE-SQS lets you transmit data securely and helps you meet strict encryption compliance and regulatory requirements at no additional cost.

SSE-SQS protects data at rest using 256-bit Advanced Encryption Standard (AES-256) encryption. SSE encrypts messages as soon as Amazon SQS receives them. Amazon SQS stores messages in encrypted form and decrypts them only when sending them to an authorized consumer.

> **Note**
>
> - The default SSE option is only effective when you create a queue without specifying encryption attributes.
> - Amazon SQS allows you to turn off all queue encryption. Therefore, turning off KMS-SSE, will not automatically enable SQS-SSE. If you wish to enable SQS-SSE after turning off KMS-SSE, you must add an attribute change in the request.

To configure SSE-SQS encryption for a queue (console)

> **Note**
>
> Any new queue created using the HTTP (non-TLS) endpoint will not enable SSE-SQS encryption by default. It is a security best practice to create Amazon SQS queues using HTTPS or Signature Version 4 endpoints.

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. Choose a queue, and then choose **Edit**.
4. Expand **Encryption**.

5. For **Server-side encryption**, choose **Enabled** (default).

> **Note**
>
> With SSE enabled, anonymous `SendMessage` and `ReceiveMessage` requests to the encrypted queue will be rejected. Amazon SQS security best practises recommend against using anonymous requests. If you wish to send anonymous requests to an Amazon SQS queue, make sure to disable SSE.

6. Select **Amazon SQS key (SSE-SQS)**. There is no additional fee for using this option.
7. Choose **Save**.

Configuring server-side encryption for a queue using the Amazon SQS console

To protect the data in a queue's messages, Amazon SQS has server-side encryption (SSE) enabled by default for all newly created queues. Amazon SQS integrates with the Amazon Web Services Key Management Service (Amazon Web Services KMS) to manage KMS keys for server-side encryption (SSE). For information about using SSE, see Encryption at rest in Amazon SQS.

The KMS key that you assign to your queue must have a key policy that includes permissions for all principals that are authorized to use the queue. For information, see Key Management.

If you aren't the owner of the KMS key, or if you log in with an account that doesn't have `kms:ListAliases` and `kms:DescribeKey` permissions, you won't be able to view information about the KMS key on the Amazon SQS console. Ask the owner of the KMS key to grant you these permissions. For more information, see Key Management.

When you create or edit a queue, you can configure SSE-KMS.

To configure SSE-KMS for an existing queue (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. Choose a queue, and then choose **Edit**.
4. Expand **Encryption**.

5. For **Server-side encryption**, choose **Enabled** (default).

> **Note**
>
> With SSE enabled, anonymous `SendMessage` and `ReceiveMessage` requests to the encrypted queue will be rejected. Amazon SQS security best practises recommend against using anonymous requests. If you wish to send anonymous requests to an Amazon SQS queue, make sure to disable SSE.

6. Select **AWS Key Management Service key (SSE-KMS)**.

   The console displays the **Description**, the **Account**, and the **KMS key ARN** of the KMS key.
7. Specify the KMS key ID for the queue. For more information, see Key terms.
   a. Choose the **Choose a KMS key alias** option.
   b. The default key is the Amazon Web Services managed KMS key for Amazon SQS. To use this key, choose it from the **KMS key** list.
   c. To use a custom KMS key from your Amazon Web Services account, choose it from the **KMS key** list. For instructions on creating custom KMS keys, see Creating Keys in the *Amazon Web Services Key Management Service Developer Guide*.
   d. To use a custom KMS key that is not in the list, or a custom KMS key from another Amazon Web Services account, choose **Enter the KMS key alias** and enter the KMS key Amazon Resource Name (ARN).
8. (Optional) For **Data key reuse period**, specify a value between 1 minute and 24 hours. The default is 5 minutes. For more information, see Understanding the data key reuse period.
9. When you finish configuring SSE-KMS, choose **Save**.

Configuring cost allocation tags for a queue using the Amazon SQS console

To help organize and identify your Amazon SQS queues, you can add cost allocation tags to them. For more information, see Amazon SQS cost allocation tags.

On the **Details** page for a queue, the **Tagging** tab displays the tags for the queue.

When you create or edit a queue, you can configure tags for it.

To configure tags for an existing queue (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. Choose a queue and choose **Edit**.
4. Scroll to the **Tags** section.
5. Add, modify, or remove the queue tags:
   a. To add a tag, choose **Add new tag**, enter a **Key** and **Value**, and then choose **Add new tag**.
   b. To update a tag, change its **Key** and **Value**.
   c. To remove a tag, choose **Remove** next to its key-value pair.
6. When you finish configuring the tags, choose **Save**.

Subscribing a queue to an Amazon SNS topic using the Amazon SQS console

You can subscribe one or more Amazon SQS queues to an Amazon Simple Notification Service (Amazon SNS) topic. When you publish a message to a topic, Amazon SNS sends the message to each of the subscribed queues. Amazon SQS manages the subscription and any necessary permissions. For more information about Amazon SNS, see What is Amazon SNS? in the *Amazon Simple Notification Service Developer Guide*.

When you subscribe an Amazon SQS queue to an SNS topic, Amazon SNS uses HTTPS to forward messages to Amazon SQS. For information about using Amazon SNS with encrypted Amazon SQS queues, see Configure KMS permissions for AWS services.

> **Important**
>
> Amazon SQS supports a maximum of 20 statements per access policy. Subscribing to an Amazon SNS topic adds one such statement. Exceeding this amount will result in a failed topic subscription delivery.

To subscribe a queue to an SNS topic (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.

2. In the navigation pane, choose **Queues**.
3. From the list of queues, choose the queue to subscribe to the SNS topic.
4. From **Actions**, choose **Subscribe to Amazon SNS topic**.
5. From the **Specify an Amazon SNS topic available for this queue** menu, choose the SNS topic for your queue.

   If the SNS topic isn't listed in the menu, choose **Enter Amazon SNS topic ARN** and then enter the topic's Amazon Resource Name (ARN).
6. Choose **Save**.
7. To verify the result of the subscription, publish to the topic and then view the message that the topic sends to the queue. For more information, see Amazon SNS message publishing in the *Amazon Simple Notification Service Developer Guide*.

If your Amazon SQS queue and SNS topic are in different AWS accounts, the topic owner must first confirm the subscription. For more information, see Confirm the subscription in the *Amazon Simple Notification Service Developer Guide*.

For information on subscribing to a cross-region SNS topic, see Sending Amazon SNS messages to an Amazon SQS queue or AWS Lambda function in a different Region in the *Amazon Simple Notification Service Developer Guide*

Configuring an Amazon SQS queue to trigger an AWS Lambda function

You can use an AWS Lambda function to process messages in an Amazon SQS queue. Lambda polls the queue and invokes your Lambda function synchronously with an event that contains queue messages. To allow your function time to process each batch of records, set the source queue's visibility timeout to at least six times the timeout that you configure on your function. The extra time allows for Lambda to retry if your function is throttled while processing a previous batch.

You can specify another queue to act as a *dead-letter queue* for messages that your Lambda function can't process.

A Lambda function can process items from multiple queues (using one Lambda event source for each queue). You can use the same queue with multiple Lambda functions.

If you associate an encrypted queue with a Lambda function but Lambda doesn't poll for messages, add the `kms:Decrypt` permission to your Lambda execution role.

Note the following restrictions:

- Your queue and the Lambda function must be in the same AWS Region.
- An encrypted queue that uses the default key (AWS managed KMS key for Amazon SQS) cannot invoke a Lambda function in a different AWS account.

For information about implementing the Lambda function, see Using AWS Lambda with Amazon SQS in the *AWS Lambda Developer Guide*.

Prerequisites

To configure Lambda function triggers, you must meet the following requirements:

- If you use a user, your Amazon SQS role must include the following permissions:
  - `lambda:CreateEventSourceMapping`
  - `lambda:ListEventSourceMappings`
  - `lambda:ListFunctions`
- The Lambda execution role must include the following permissions:
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `sqs:ReceiveMessage`
- If you associate an encrypted queue with a Lambda function, add the `kms:Decrypt` permission to the Lambda execution role.

For more information, see Overview of managing access in Amazon SQS.

To configure a queue to trigger a Lambda function (console)

1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. On the **Queues** page, choose the queue to configure.
4. On the queue's page, choose the **Lambda triggers** tab.

5. On the **Lambda triggers** page, choose a Lambda trigger.

   If the list doesn't include the Lambda trigger that you need, choose **Configure Lambda function trigger**. Enter the Amazon Resource Name (ARN) of the Lambda function or choose an existing resource. Then choose **Save**.
6. Choose **Save**. The console saves the configuration and displays the **Details** page for the queue.

   On the **Details** page, the **Lambda triggers** tab displays the Lambda function and its status. It takes approximately 1 minute for the Lambda function to become associated with your queue.
7. To verify the results of the configuration, send a message to your queue and then view the triggered Lambda function in the Lambda console.

Automating notifications from AWS services to Amazon SQS using Amazon EventBridge

Amazon EventBridge lets you automate AWS services and respond to system events such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge nearly in real time. You can write simple rules to indicate which events are of interest to you and what automated actions to take when an event matches a rule.

EventBridge lets you set a variety of *targets*—such as Amazon SQS standard and FIFO queues—which receive events in JSON format. For more information, see Amazon EventBridge targets in the *Amazon EventBridge User Guide*.

Sending a message with attributes using Amazon SQS

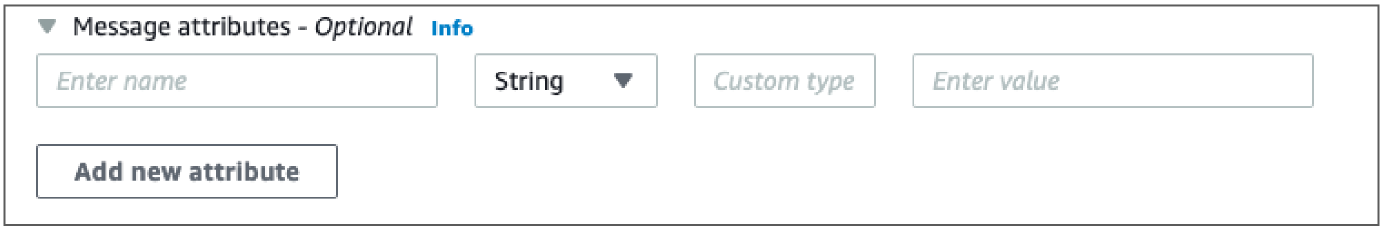
For standard and FIFO queues, you can include structured metadata (such as timestamps, geospatial data, signatures, and identifiers) with messages. For more information, see Amazon SQS message attributes.

To send a message with attributes to a queue using the Amazon SQS console

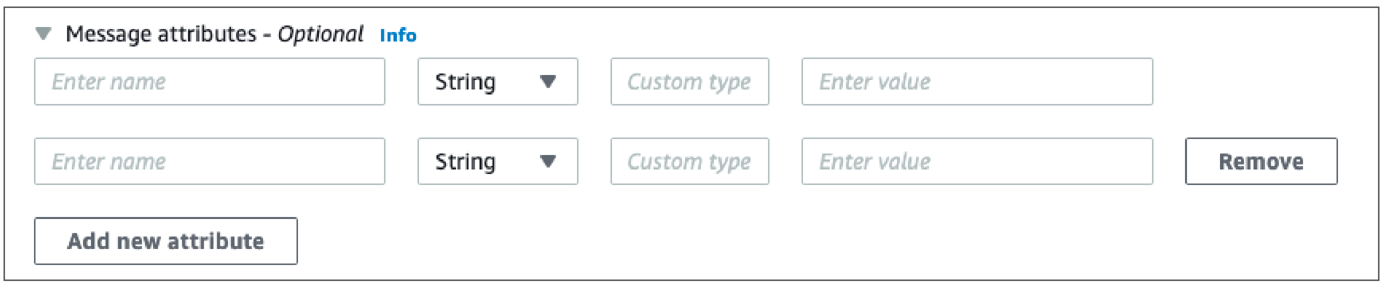
1. Open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. In the navigation pane, choose **Queues**.
3. On the **Queues** page, choose a queue.
4. Choose **Send and receive messages**.

5. Enter the message attribute parameters.

   a. In the name text box, enter a unique name of up to 256 characters.
   b. For the attribute type, choose **String**, **Number**, or **Binary**.
   c. (Optional) Enter a custom data type. For example, you could add **`byte`**, **`int`**, or **`float`** as custom data types for **Number**.
   d. In the value text box, enter the message attribute value.



6. To add another message attribute., choose **Add new attribute**.



7. You can modify the attribute values any time before sending the message.
8. To delete an attribute, choose **Remove**. To delete the first attribute, close **Message attributes**.
9. When you finish adding attributes to the message, choose **Send message**. Your message is sent and the console displays a success message. To view information about the message attributes of the sent message, choose **View details**. Choose **Done** to close the **Message details** dialog box.

Amazon SQS best practices

Amazon SQS manages and processes message queues, facilitating smooth communication between different parts of an application and ensuring scalable and reliable message handling. This guide covers essential operational best practices for Amazon SQS, including using long polling to minimize empty responses, implementing dead-letter queues for processing errors, and optimizing queue permissions for security. The goal is to provide a concrete and actionable approach to operating and troubleshooting Amazon SQS messaging.

Topics

Amazon SQS error handling and problematic messages

This topic provides detailed instructions on managing and mitigating errors in Amazon SQS, including techniques for handling request errors, capturing problematic messages, and configuring dead-letter queue retention to ensure message reliability.

Topics

Handling request errors in Amazon SQS

To handle request errors, use one of the following strategies:

- If you use an AWS SDK, you already have automatic *retry and backoff* logic at your disposal. For more information, see Error Retries and Exponential Backoff in AWS in the *Amazon Web Services General Reference*.
- If you don't use the AWS SDK features for retry and backoff, allow a pause (for example, 200 ms) before retrying the ReceiveMessage action after receiving no messages, a timeout, or an

error message from Amazon SQS. For subsequent use of `ReceiveMessage` that gives the same results, allow a longer pause (for example, 400 ms).

Capturing problematic messages in Amazon SQS

To capture all messages that can't be processed, and to collect accurate CloudWatch metrics, configure a dead-letter queue.

- The redrive policy redirects messages to a dead-letter queue after the source queue fails to process a message a specified number of times.
- Using a dead-letter queue decreases the number of messages and reduces the possibility of exposing you to *poison pill* messages (messages that are received but can't be processed).
- Including a poison pill message in a queue can distort the `ApproximateAgeOfOldestMessage` CloudWatch metric by giving an incorrect age of the poison pill message. Configuring a dead-letter queue helps avoid false alarms when using this metric.

Setting-up dead-letter queue retention in Amazon SQS

For standard queues, the expiration of a message is always based on its original enqueue timestamp. When a message is moved to a dead-letter queue, the enqueue timestamp is unchanged. The `ApproximateAgeOfOldestMessage` metric indicates when the message moved to the dead-letter queue, *not* when the message was originally sent. For example, assume that a message spends 1 day in the original queue before it's moved to a dead-letter queue. If the dead-letter queue's retention period is 4 days, the message is deleted from the dead-letter queue after 3 days and the `ApproximateAgeOfOldestMessage` is 3 days. Thus, it is a best practice to always set the retention period of a dead-letter queue to be longer than the retention period of the original queue.

For FIFO queues, the enqueue timestamp resets when the message is moved to a dead-letter queue. The `ApproximateAgeOfOldestMessage` metric indicates when the message moved to the dead-letter queue. In the same example above, the message is deleted from the dead-letter queue after 4 days and the `ApproximateAgeOfOldestMessage` is 4 days.

Amazon SQS message deduplication and grouping

This topic provides best practices for ensuring consistent message processing in Amazon SQS, including the use of message deduplication ID and message group ID properties to prevent duplicate messages and manage message groups effectively.

Topics

Avoiding inconsistent message processing in Amazon SQS

Because Amazon SQS is a distributed system, it is possible for a consumer to not receive a message even when Amazon SQS marks the message as delivered while returning successfully from a `ReceiveMessage` API method call. In this case, Amazon SQS records the message as delivered at least once, although the consumer has never received it. Because no additional attempts to deliver messages are made under these conditions, we don't recommend setting the number of maximum receives to 1 for a dead-letter queue.

Using the Amazon SQS message deduplication ID

Message deduplication ID is the token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute deduplication interval.

> **Note**
>
> Amazon SQS continues to keep track of the message deduplication ID even after the message is received and deleted.

Providing the message deduplication ID in Amazon SQS

The producer should provide message deduplication ID values for each message in the following scenarios:

- Messages sent with identical message bodies that Amazon SQS must treat as unique.
- Messages sent with identical content but different message attributes that Amazon SQS must treat as unique.
- Messages sent with different content (for example, retry counts included in the message body) that Amazon SQS must treat as duplicates.

Enabling deduplication for a single-producer/consumer system in Amazon SQS

If you have a single producer and a single consumer and the messages are unique because an application-specific message ID is included in the body of the message, follow these best practices:

- Enable content-based deduplication for the queue (each of your messages has a unique body). The producer can omit the message deduplication ID.
- When content-based deduplication is enabled for an Amazon SQS FIFO queue, and a message is sent with a deduplication ID, the `SendMessage` deduplication ID overrides the generated content-based deduplication ID.
- Although the consumer isn't required to provide a receive request attempt ID for each request, it's a best practice because it allows fail-retry sequences to execute faster.
- You can retry send or receive requests because they don't interfere with the ordering of messages in FIFO queues.

Designing for outage recovery scenarios in Amazon SQS

The deduplication process in FIFO queues is time-sensitive. When designing your application, make sure that both the producer and the consumer can recover in case of a client or network outage.

- The producer must be aware of the deduplication interval of the queue. Amazon SQS has a deduplication interval of 5 minutes. Retrying `SendMessage` requests after the deduplication interval expires can introduce duplicate messages into the queue. For example, a mobile device in a car sends messages whose order is important. If the car loses cellular connectivity for a period of time before receiving an acknowledgement, retrying the request after regaining cellular connectivity can create a duplicate.

- The consumer must have a visibility timeout that minimizes the risk of being unable to process messages before the visibility timeout expires. You can extend the visibility timeout while the messages are being processed by calling the `ChangeMessageVisibility` action. However, if the visibility timeout expires, another consumer can immediately begin to process the messages, causing a message to be processed multiple times. To avoid this scenario, configure a dead-letter queue.

Working with visibility timeouts in Amazon SQS

For optimal performance, set the visibility timeout to be larger than the AWS SDK read timeout. This applies to using the `ReceiveMessage` API action with either short polling or long polling.

Using the Amazon SQS message group ID

`MessageGroupId` is the tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are always processed one by one, in a strict order relative to the message group (however, messages that belong to different message groups might be processed out of order).

Interleaving multiple ordered message groups in Amazon SQS

To interleave multiple ordered message groups within a single FIFO queue, use message group ID values (for example, session data for multiple users). In this scenario, multiple consumers can process the queue, but the session data of each user is processed in a FIFO manner.

> **Note**
>
> When messages that belong to a particular message group ID are invisible, no other consumer can process messages with the same message group ID.

Avoiding processing duplicates in a multiple-producer/consumer system in Amazon SQS

To avoid processing duplicate messages in a system with multiple producers and consumers where throughput and latency are more important than ordering, the producer should generate a unique message group ID for each message.

> **Note**
>
> In this scenario, duplicates are eliminated. However, the ordering of message can't be guaranteed.
>
> Any scenario with multiple producers and consumers increases the risk of inadvertently delivering a duplicate message if a worker doesn't process the message within the visibility timeout and the message becomes available to another worker.

Avoid having a large backlog of messages with the same message group ID in Amazon SQS

For FIFO queues, there can be a maximum of 20,000 in flight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this quota, Amazon SQS returns no error messages. A FIFO queue looks through the first 20k messages to determine available message groups. This means that if you have a backlog of messages in a single message group, you can't consume messages from other message groups that were sent to the queue at a later time until you successfully consume the messages from the backlog.

> **Note**
>
> A backlog of messages that have the same message group ID might build up because of a consumer that can't successfully process a message. Message processing issues can occur because of an issue with the content of a message or because of a technical issue with the consumer.
>
> To move away messages that can't be processed repeatedly, and to unblock the processing of other messages that have the same message group ID, consider setting up a dead-letter queue policy.

Avoid reusing the same message group ID with virtual queues in Amazon SQS

To prevent messages with the same message group ID sent to different virtual queues with the same host queue from blocking each other, avoid reusing the same message group ID with virtual queues.

Using the Amazon SQS receive request attempt ID

The receive request attempt ID is the token used for deduplication of `ReceiveMessage` calls.

During a long-lasting network outage that causes connectivity issues between your SDK and Amazon SQS, it's a best practice to provide the receive request attempt ID and to retry with the same receive request attempt ID if the SDK operation fails.

Amazon SQS message processing and timing

This topic provides a comprehensive guidance on optimizing the speed and efficiency of message handling in Amazon SQS, including strategies for timely message processing, selecting the best polling mode, and configuring long polling for improved performance.

Topics

Processing messages in a timely manner in Amazon SQS

Setting the visibility timeout depends on how long it takes your application to process and delete a message. For example, if your application requires 10 seconds to process a message and you set the visibility timeout to 15 minutes, you must wait for a relatively long time to attempt to process the message again if the previous processing attempt fails. Alternatively, if your application requires 10 seconds to process a message but you set the visibility timeout to only 2 seconds, a duplicate message is received by another consumer while the original consumer is still working on the message.

To make sure that there is sufficient time to process messages, use one of the following strategies:

- If you know (or can reasonably estimate) how long it takes to process a message, extend the message's *visibility timeout* to the maximum time it takes to process and delete the message. For more information, see Configuring the Visibility Timeout.
- If you don't know how long it takes to process a message, create a *heartbeat* for your consumer process: Specify the initial visibility timeout (for example, 2 minutes) and then—as long as your

consumer still works on the message—keep extending the visibility timeout by 2 minutes every minute.

> **Important**
>
> The maximum visibility timeout is 12 hours from the time that Amazon SQS receives the `ReceiveMessage` request. Extending the visibility timeout does not reset the 12 hour maximum.
> Additionally, you may be unable to set the timeout on an individual message to the full 12 hours (e.g. 43,200 seconds) since the `ReceiveMessage` request initiates the timer. For example, if you receive a message and immediately set the 12 hour maximum by sending a `ChangeMessageVisibility` call with `VisibilityTimeout` equal to 43,200 seconds, it will likely fail. However, using a value of 43,195 seconds will work unless there is a significant delay between requesting the message via `ReceiveMessage` and updating the visibility timeout. If your consumer needs longer than 12 hours, consider using Step Functions.

Setting-up long polling in Amazon SQS

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 (when there are no messages available for a `ReceiveMessage` request) and false empty responses (when messages are available but aren't included in a response). For more information, see Amazon SQS short and long polling.

For optimal message processing, use the following strategies:

- In most cases, you can set the `ReceiveMessage` wait time to 20 seconds. If 20 seconds is too long for your application, set a shorter `ReceiveMessage` wait time (1 second minimum). If you don't use an AWS SDK to access Amazon SQS, or if you configure an AWS SDK to have a shorter wait time, you might have to modify your Amazon SQS client to either allow longer requests or use a shorter wait time for long polling.
- If you implement long polling for multiple queues, use one thread for each queue instead of a single thread for all queues. Using a single thread for each queue allows your application to process the messages in each of the queues as they become available, while using a single thread for polling multiple queues might cause your application to become unable to process messages

available in other queues while the application waits (up to 20 seconds) for the queue which doesn't have any available messages.

> **Important**
>
> To avoid HTTP errors, make sure that the HTTP response timeout for `ReceiveMessage` requests is longer than the `WaitTimeSeconds` parameter. For more information, see ReceiveMessage.

Using the appropriate polling mode in Amazon SQS

- Long polling lets you consume messages from your Amazon SQS queue as soon as they become available.
  - To reduce the cost of using Amazon SQS and to decrease the number of empty receives to an empty queue (responses to the `ReceiveMessage` action which return no messages), enable long polling. For more information, see Amazon SQS Long Polling.
  - To increase efficiency when polling for multiple threads with multiple receives, decrease the number of threads.
  - Long polling is preferable over short polling in most cases.
- Short polling returns responses immediately, even if the polled Amazon SQS queue is empty.
  - To satisfy the requirements of an application that expects immediate responses to the `ReceiveMessage` request, use short polling.
  - Short polling is billed the same as long polling.

Amazon SQS Java SDK examples

You can use the AWS SDK for Java to build Java applications that interact with Amazon Simple Queue Service (Amazon SQS) and other AWS services. To install and set up the SDK, see Getting started in the *AWS SDK for Java 2.x Developer Guide*.

For examples of basic Amazon SQS queue operations, such as how to create a queue or send a message, see Working with Amazon SQS Message Queues in the *AWS SDK for Java 2.x Developer Guide*.

The examples in this topic demonstrate additional Amazon SQS features, such as server-side encryption (SSE), cost-allocation tags, and message attributes.

Topics

Using server-side encryption with Amazon SQS queues

You can use the AWS SDK for Java to add server-side encryption (SSE) to an Amazon SQS queue. Each queue uses an AWS Key Management Service (AWS KMS) KMS key to generate the data encryption keys. This example uses the AWS managed KMS key for Amazon SQS. For more information about using SSE and the role of the KMS key, see Encryption at rest in Amazon SQS.

Adding SSE to an existing queue

To enable server-side encryption for an existing queue, use the `SetQueueAttributes` method to set the `KmsMasterKeyId` attribute.

The following code example sets the AWS KMS key as the AWS managed KMS key for Amazon SQS. The example also sets the AWS KMS key reuse period to 140 seconds.

Before you run the example code, make sure that you have set your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the *AWS SDK for Java 2.x Developer Guide*.

```
// Create an SqsClient for the specified Region.
```

```
SqsClient sqsClient = SqsClient.builder().region(Region.US_WEST_1).build();

// Get the URL of your queue.
String myQueueName = "my queue";
GetQueueUrlResponse getQueueUrlResponse =

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

// 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");

// Create the SetQueueAttributesRequest.
SetQueueAttributesRequest set_attrs_request = SetQueueAttributesRequest.builder()
          .queueUrl(queueUrl)
          .attributes(attributes)
          .build();

sqsClient.setQueueAttributes(set_attrs_request);
```

Disabling SSE for a queue

To disable server-side encryption for an existing queue, set the `KmsMasterKeyId` attribute to an empty string using the `SetQueueAttributes` method.

Important

`null` isn't a valid value for `KmsMasterKeyId`.

Creating a queue with SSE

To enable SSE when you create the queue, add the `KmsMasterKeyId` attribute to the CreateQueue API method.

The following example creates a new queue with SSE enabled. The queue uses the AWS managed KMS key for Amazon SQS. The example also sets the AWS KMS key reuse period to 160 seconds.

Before you run the example code, make sure that you have set your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the *AWS SDK for Java 2.x Developer Guide*.

```
// 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);
```

Retrieving SSE attributes

For information about retrieving queue attributes, see Examples in the *Amazon Simple Queue Service API Reference*.

To retrieve the KMS key ID or the data key reuse period for a particular queue, run the `GetQueueAttributes` method and retrieve the `KmsMasterKeyId` and `KmsDataKeyReusePeriodSeconds` values.

Configuring tags for an Amazon SQS queue

Use cost-allocation tags to help organize and identify your Amazon SQS queues. The following examples show how to configure tags using the AWS SDK for Java. For more information, see Amazon SQS cost allocation tags.

Before you run the example code, make sure that you have set your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the *AWS SDK for Java 2.x Developer Guide*.

Listing tags

To list the tags for a queue, use the `ListQueueTags` method.

```
// 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() ));
```

Adding or updating tags

To add or update tag values for a queue, use the `TagQueue` method.

```
 // 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);
```

Removing tags

To remove one or more tags from the queue, use the `UntagQueue` method. The following example removes the `Accounting ID` tag.

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

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

Sending message attributes to an Amazon SQS queue

You can include structured metadata (such as timestamps, geospatial data, signatures, and identifiers) with messages using *message attributes*. For more information, see Amazon SQS message attributes.

Before you run the example code, make sure that you have set your AWS credentials. For more information, see Set up AWS Credentials and Region for Development in the *AWS SDK for Java 2.x Developer Guide*.

Defining attributes

To define an attribute for a message, add the following code, which uses the `MessageAttributeValue` data type. For more information, see Message attribute components and Message attribute data types.

The AWS SDK for Java automatically calculates the message body and message attribute checksums and compares them with the data that Amazon SQS returns. For more information, see the *AWS SDK for Java 2.x Developer Guide* and Calculating the MD5 message digest for message attributes for other programming languages.

String

This example defines a `String` attribute named `Name` with the value `Jane`.

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

Number

This example defines a `Number` attribute named `AccurateWeight` with the value `230.000000000000000001`.

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

Binary

This example defines a Binary attribute named ByteArray with the value of an uninitialized 10-byte array.

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

String (custom)

This example defines the custom attribute String.EmployeeId named EmployeeId with the value ABC123456.

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

Number (custom)

This example defines the custom attribute Number.AccountId named AccountId with the value 000123456.

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

Note

Because the base data type is Number, the [ReceiveMessage](#) method returns 123456.

Binary (custom)

This example defines the custom attribute Binary.JPEG named ApplicationIcon with the value of an uninitialized 10-byte array.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
```

```
messageAttributes.put("ApplicationIcon", new MessageAttributeValue()
.withDataType("Binary.JPEG")
.withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

Sending a message with attributes

This example adds the attributes to the `SendMessageRequest` before sending the message.

```
// 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

If you send a message to a First-In-First-Out (FIFO) queue, make sure that the `sendMessage` method executes *after* you provide the message group ID.

If you use the `SendMessageBatch` method instead of `SendMessage`, you must specify message attributes for each message in the batch.

Using APIs with Amazon SQS

This section provides information about constructing Amazon SQS endpoints, making query API requests using the GET and POST methods, and using batch API actions. For detailed information about Amazon SQS actions—including parameters, errors, examples, and data types, see the *Amazon Simple Queue Service API Reference*.

To access Amazon SQS using a variety of programming languages, you can also use AWS SDKs, which contain the following automatic functionality:

- Cryptographically signing your service requests
- Retrying requests
- Handling error responses

For more information, see the section called “Working with AWS SDKs”.

For command line tool information, see the Amazon SQS sections in the *AWS CLI Command Reference* and the *AWS Tools for PowerShell Cmdlet Reference*.

Amazon SQS APIs with AWS JSON protocol

Amazon SQS uses AWS JSON protocol as the transport mechanism for all Amazon SQS APIs on the specified AWS SDK versions. AWS JSON protocol provides a higher throughput, lower latency, and faster application-to-application communication. AWS JSON protocol is more efficient in serialization/deserialization of requests and responses when compared to AWS query protocol. If you still prefer to use the AWS query protocol with SQS APIs, see What languages are supported for AWS JSON protocol used in Amazon SQS APIs? for the AWS SDK versions that support Amazon SQS AWS query protocol.

Amazon SQS uses AWS JSON protocol to communicate between AWS SDK clients (for example, Java, Python, Golang, JavaScript) and the Amazon SQS server. An HTTP request of an Amazon SQS API operation accepts JSON formatted input. The Amazon SQS operation is executed, and the execution response is sent back to the SDK client in JSON format. Compared to AWS query, AWS JSON is simpler, faster, and more efficient to transport data between client and server.

- AWS JSON protocol acts as a mediator between the Amazon SQS client and server.
- The server doesn’t understand the programming language in which the Amazon SQS operation is created, but it understands the AWS JSON protocol.

- The AWS JSON protocol uses the serialization (convert object to JSON format) and de-serialization (convert JSON format to object) between Amazon SQS client and server.

For more information about AWS JSON protocol with Amazon SQS, see Amazon SQS AWS JSON protocol FAQs.

AWS JSON protocol is available on the specified AWS SDK version. To review SDK version and release dates across language variants, see the AWS SDKs and Tools version support matrix in the *AWS SDKs and Tools Reference Guide*

Topics

Making query API requests using AWS JSON protocol in Amazon SQS

In this section you learn how to construct an Amazon SQS endpoint, make POST requests, and interpret responses.

> **Note**
>
> AWS JSON protocol is supported for most language variants. For a full list of supported language variants, see What languages are supported for AWS JSON protocol used in Amazon SQS APIs?.

Topics

- Amazon SQS AWS JSON protocol FAQs

Constructing an endpoint

To work with Amazon SQS queues, you must construct an endpoint. For information about Amazon SQS endpoints, see the following pages in the *Amazon Web Services General Reference*:

- Regional endpoints
- Amazon Simple Queue Service endpoints and quotas

Every Amazon SQS endpoint is independent. For example, if two queues are named *MyQueue* and one has the endpoint `sqs.us-east-2.amazonaws.com` while the other has the endpoint `sqs.eu-west-2.amazonaws.com`, the two queues don't share any data with each other.

The following is an example of an endpoint that makes a request to create a queue.

```
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

Queue names and queue URLs are case sensitive.
The structure of *AUTHPARAMS* depends on the signature of the API request. For more information, see Signing AWS API Requests in the *Amazon Web Services General Reference*.

Making a POST request

An Amazon SQS POST request sends query parameters as a form in the body of an HTTP request.

The following is an example of an HTTP header with `X-Amz-Target` set to `AmazonSQS.<operationName>`, and an HTTP header with `Content-Type` set to `application/x-amz-json-1.0`.

```
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",
}
```

This HTTP POST request sends a message to an Amazon SQS queue.

> **Note**
>
> Both HTTP headers `X-Amz-Target` and `Content-Type` are required.
> Your HTTP client might add other items to the HTTP request, according to the client's HTTP version.

Interpreting Amazon SQS JSON API responses

In response to an action request, Amazon SQS returns a JSON data structure that contains the results of the request. For more information, see the individual actions in the *Amazon Simple Queue Service API Reference* and Amazon SQS AWS JSON protocol FAQs.

Topics

- Successful JSON response structure
- JSON error response structure

Successful JSON response structure

If the request is successful, the main response element is `x-amzn-RequestId`, which contains the Universal Unique Identifier (UUID) of the request, as well as other appended response field(s). For example, the following `CreateQueue` response contains the `QueueUrl` field, which, in turn, contains the URL of the created queue.

```
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 error response structure

If a request is unsuccessful, Amazon SQS returns the main response, including the HTTP header and the body.

In the HTTP header, `x-amzn-RequestId` contains the UUID of the request. `x-amzn-query-error` contains two pieces of information: the type of error, and whether the error was a producer or consumer error.

In the response body, `"__type"` indicates other error details, and `Message` indicates the error condition in a readable format.

The following is an example error response in JSON format:

```
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 protocol FAQs

Frequently asked questions about using AWS JSON protocol with Amazon SQS.

What is AWS JSON protocol, and how does it differ from existing Amazon SQS API requests and responses?

JSON is one of the most widely used and accepted wiring methods for communication between heterogeneous systems. Amazon SQS uses JSON as a medium to communicate between an AWS SDK client (for example, Java, Python, Golang, JavaScript) and Amazon SQS server. An HTTP request of an Amazon SQS API operation accepts input in the form of JSON. The Amazon SQS operation is executed and the response of execution is shared back to the SDK client in the form of JSON. Compared to AWS query, JSON is more efficient at transporting data between client and server.

- Amazon SQS AWS JSON protocol acts as a mediator between Amazon SQS client and server.
- The server doesn't understand the programming language in which the Amazon SQS operation is created, but it understands the AWS JSON protocol.
- The Amazon SQS AWS JSON protocol uses the serialization (convert object to JSON format) and deserialization (convert JSON format to object) between the Amazon SQS client and server.

How do I get started with AWS JSON protocols for Amazon SQS?

To get started with the latest AWS SDK version to achieve faster messaging for Amazon SQS, upgrade your AWS SDK to the specified version or any subsequent version. To learn more about SDK clients, see the Guide column in the table below.

The following is a list of SDK versions across language variants for AWS JSON protocol for use with Amazon SQS APIs:

Language	SDK client repository	Required SDK client version	Guide
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

Language	SDK client repository	Required SDK client version	Guide
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
JavaScript v2.x	aws/aws-sdk-js	v2.1492.0	JavaScript on AWS
JavaScript v3.x	aws/aws-sdk-js-v3	v3.447.0	JavaScript on 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	AWSCommand Line Interface
Ruby	aws/aws-sdk-ruby	1.67.0	AWS SDK for Ruby

What are the risks of enabling JSON protocol for my Amazon SQS workloads?

If you are using a custom implementation of AWS SDK or a combination of custom clients and AWS SDK to interact with Amazon SQS that generates AWS Query based (aka XML-based) responses, it may be incompatible with AWS JSON protocol. If you encounter any issues, contact AWS Support.

What if I am already on the latest AWS SDK version, but my open sourced solution does not support JSON?

You must change your SDK version to the version previous to what you are using. See How do I get started with AWS JSON protocols for Amazon SQS? for more information. AWS SDK versions listed in How do I get started with AWS JSON protocols for Amazon SQS? uses JSON wire protocol for Amazon SQS APIs. If you change your AWS SDK to the previous version, your Amazon SQS APIs will use the AWS query.

What languages are supported for AWS JSON protocol used in Amazon SQS APIs?

Amazon SQS supports all language variants where AWS SDKs are generally available (GA). Currently, we don't support Kotlin, Rust, or Swift. To learn more about other language variants, see Tools to Build on AWS.

What regions are supported for AWS JSON protocol used in Amazon SQS APIs

Amazon SQS supports AWS JSON protocol in all AWS regions where Amazon SQS is available.

What latency improvements can I expect when upgrading to the specified AWS SDK versions for Amazon SQS using the AWS JSON protocol?

AWS JSON protocol is more efficient at serialization and deserialization of requests and responses when compared to AWS query protocol. Based on AWS performance tests for a 5 KB message payload, JSON protocol for Amazon SQS reduces end-to-end message processing latency by up to 23%, and reduces application client side CPU and memory usage.

Will AWS query protocol be deprecated?

AWS query protocol will continue to be supported. You can continue using AWS query protocol as long as your AWS SDK version is set any previous version other that what is listed in How do I get started with AWS JSON protocols for Amazon SQS.

Where can I find more information about AWS JSON protocol?

You can find more information about JSON protocol at AWS JSON 1.0 protocol in the *Smithy* documentation. For more about Amazon SQS API requests using AWS JSON protocol, see Making query API requests using AWS JSON protocol in Amazon SQS.

Making query API requests using AWS query protocol in Amazon SQS

In this section you learn how to construct an Amazon SQS endpoint, make GET and POST requests, and interpret responses.

Topics

Constructing an endpoint

In order to work with Amazon SQS queues, you must construct an endpoint. For information about Amazon SQS endpoints, see the following pages in the *Amazon Web Services General Reference*:

- Regional endpoints
- Amazon Simple Queue Service endpoints and quotas

Every Amazon SQS endpoint is independent. For example, if two queues are named *MyQueue* and one has the endpoint `sqs.us-east-2.amazonaws.com` while the other has the endpoint `sqs.eu-west-2.amazonaws.com`, the two queues don't share any data with each other.

The following is an example of an endpoint which makes a request to create a queue.

```
https://sqs.eu-west-2.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=MyQueue
&Version=2012-11-05
&AUTHPARAMS
```

> **Note**
>
> Queue names and queue URLs are case sensitive.

The structure of *AUTHPARAMS* depends on the signature of the API request. For more information, see Signing AWS API Requests in the *Amazon Web Services General Reference*.

Making a GET request

An Amazon SQS GET request is structured as a URL which consists of the following:

- **Endpoint** – The resource that the request is acting on (the queue name and URL), for example: `https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue`
- **Action** – The action that you want to perform on the endpoint. A question mark (?) separates the endpoint from the action, for example: `?Action=SendMessage&MessageBody=Your%20Message%20Text`
- **Parameters** – Any request parameters. Each parameter is separated by an ampersand (&), for example: `&Version=2012-11-05&AUTHPARAMS`

The following is an example of a GET request that sends a message to an Amazon SQS queue.

```
https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue
?Action=SendMessage&MessageBody=Your%20message%20text
&Version=2012-11-05
&AUTHPARAMS
```

Note

Queue names and queue URLs are case sensitive.
Because GET requests are URLs, you must URL-encode all parameter values. Because spaces aren't allowed in URLs, each space is URL-encoded as `%20`. The rest of the example isn't URL-encoded to make it easier to read.

Making a POST request

An Amazon SQS POST request sends query parameters as a form in the body of an HTTP request.

The following is an example of an HTTP header with `Content-Type` set to `application/x-www-form-urlencoded`.

```
POST /123456789012/MyQueue HTTP/1.1
Host: sqs.us-east-2.amazonaws.com
Content-Type: application/x-www-form-urlencoded
```

The header is followed by a `form-urlencoded` GET request that sends a message to an Amazon SQS queue. Each parameter is separated by an ampersand (&).

```
Action=SendMessage
&MessageBody=Your+Message+Text
&Expires=2020-10-15T12%3A00%3A00Z
&Version=2012-11-05
&AUTHPARAMS
```

Note

Only the `Content-Type` HTTP header is required. The *AUTHPARAMS* is the same as for the GET request.

Your HTTP client might add other items to the HTTP request, according to the client's HTTP version.

Interpreting Amazon SQS XML API responses

In response to an action request, Amazon SQS returns an XML data structure that contains the results of the request. For more information, see the individual actions in the *Amazon Simple Queue Service API Reference*.

Topics

- Successful XML response structure
- XML error response structure

Successful XML response structure

If the request is successful, the main response element is named after the action, with `Response` appended (for example, *`ActionName`*`Response`).

This element contains the following child elements:

- **_ActionName_Result** – Contains an action-specific element. For example, the `CreateQueueResult` element contains the `QueueUrl` element which, in turn, contains the URL of the created queue.
- **ResponseMetadata** – Contains the `RequestId` which, in turn, contains the Universal Unique Identifier (UUID) of the request.

The following is an example successful response in XML format:

```
<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 error response structure

If a request is unsuccessful, Amazon SQS always returns the main response element `ErrorResponse`. This element contains an `Error` element and a `RequestId` element.

The `Error` element contains the following child elements:

- **Type** – Specifies whether the error was a producer or consumer error.
- **Code** – Specifies the type of error.
- **Message** – Specifies the error condition in a readable format.
- **Detail** – (Optional) Specifies additional details about the error.

The `RequestId` element contains the UUID of the request.

The following is an example error response in XML format:

```
<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>
```

Authenticating requests for Amazon SQS

Authentication is the process of identifying and verifying the party that sends a request. During the first stage of authentication, AWS verifies the identity of the producer and whether the producer is registered to use AWS (for more information, see Step 1: Create an AWS account and IAM user). Next, AWS abides by the following procedure:

1. The producer (sender) obtains the necessary credential.
2. The producer sends a request and the credential to the consumer (receiver).
3. The consumer uses the credential to verify whether the producer sent the request.
4. One of the following happens:
    - If authentication succeeds, the consumer processes the request.
    - If authentication fails, the consumer rejects the request and returns an error.

Topics

Basic authentication process with HMAC-SHA

When you access Amazon SQS using the Query API, you must provide the following items to authenticate your request:

- The **AWS Access Key ID** that identifies your AWS account, which AWS uses to look up your Secret Access Key.

- The **HMAC-SHA request signature**, calculated using your Secret Access Key (a shared secret known only to you and AWS—for more information, see RFC2104). The AWS SDK handles the signing process; however, if you submit a query request over HTTP or HTTPS, you must include a signature in every query request.

  1. Derive a Signature Version 4 Signing Key. For more information, see Deriving the Signing Key with Java.

     Note

     Amazon SQS supports Signature Version 4, which provides improved SHA256-based security and performance over previous versions. When you create new applications that use Amazon SQS, use Signature Version 4.

  2. Base64-encode the request signature. The following sample Java code does this:

```
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);
    }
}
```

- The **timestamp (or expiration)** of the request. The timestamp that you use in the request must be a `dateTime` object, with the complete date, including hours, minutes, and seconds. For example: `2007-01-31T23:59:59Z` Although this isn't required, we recommend providing the object using the Coordinated Universal Time (Greenwich Mean Time) time zone.

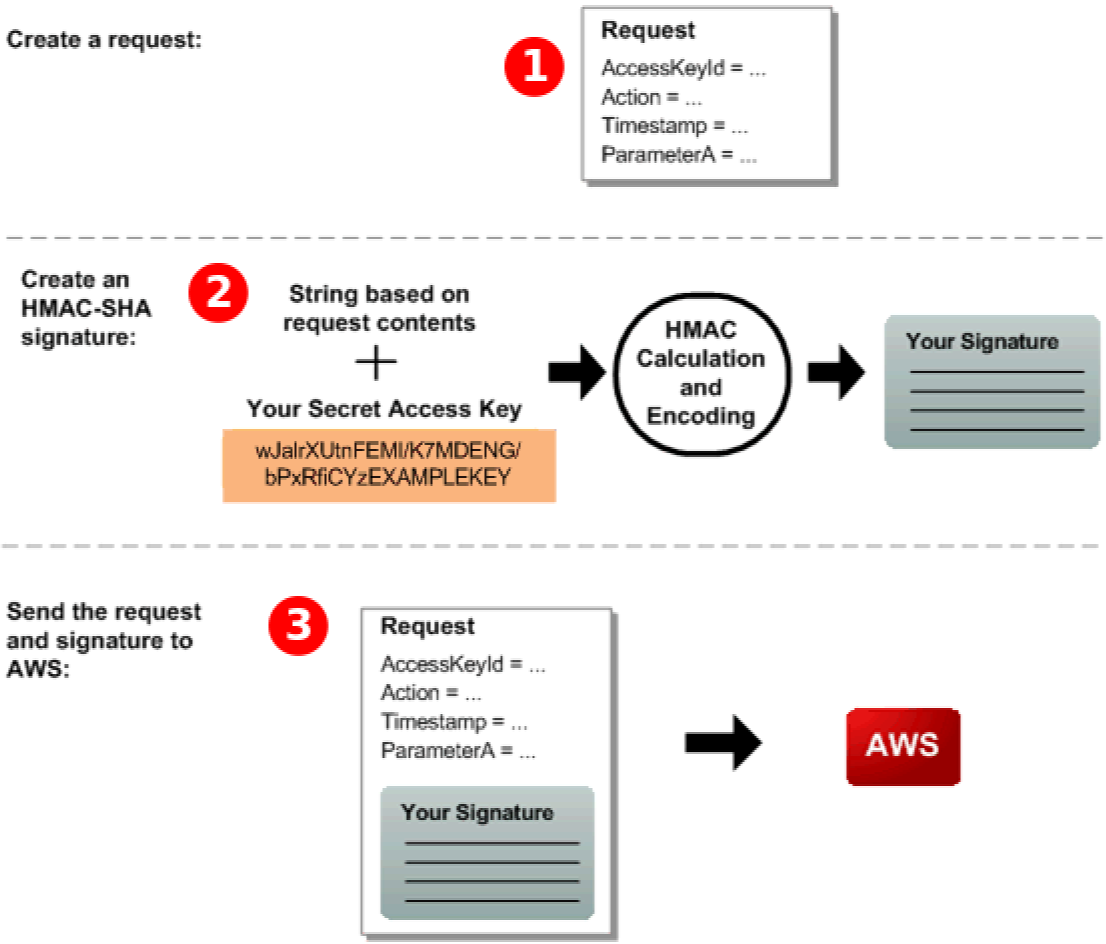
  Note

  Make sure that your server time is set correctly. If you specify a timestamp (rather than an expiration), the request automatically expires 15 minutes after the specified time

(AWS doesn't process requests with timestamps more than 15 minutes earlier than the current time on AWS servers).
If you use .NET, you must not send overly specific timestamps (because of different interpretations of how extra time precision should be dropped). In this case, you should manually construct `dateTime` objects with precision of no more than one millisecond.

Part 1: The request from the user

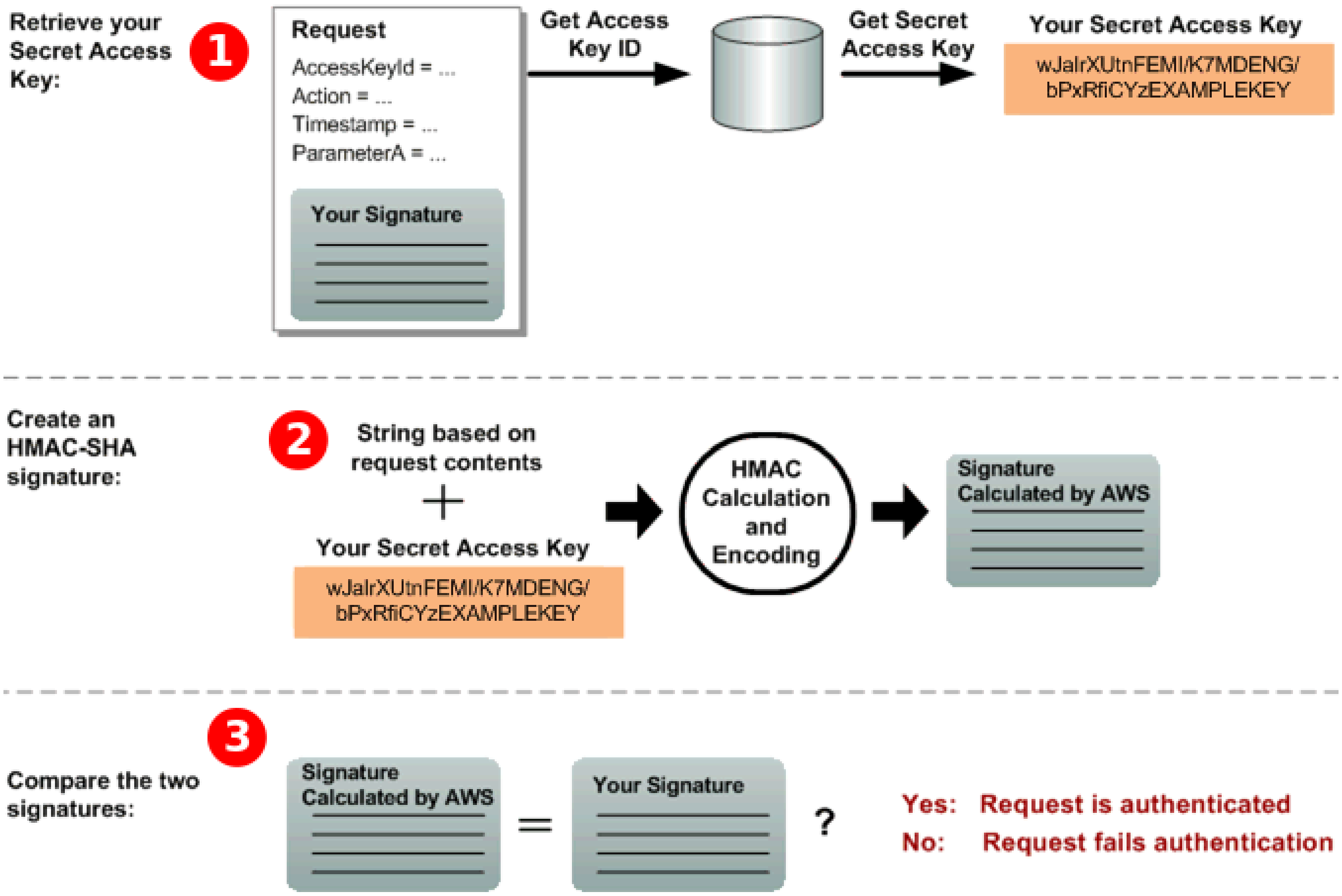
The following is the process you must follow to authenticate AWS requests using an HMAC-SHA request signature.



1. Construct a request to AWS.
2. Calculate a keyed-hash message authentication code (HMAC-SHA) signature using your Secret Access Key.
3. Include the signature and your Access Key ID in the request, and then send the request to AWS.

Part 2: The response from AWS

AWS begins the following process in response.



1. AWS uses the Access Key ID to look up your Secret Access Key.
2. AWS generates a signature from the request data and the Secret Access Key, using the same algorithm that you used to calculate the signature you sent in the request.
3. One of the following happens:

   - If the signature that AWS generates matches the one you send in the request, AWS considers the request to be authentic.
   - If the comparison fails, the request is discarded, and AWS returns an error.

Amazon SQS batch actions

Amazon SQS provides batch actions to help you reduce costs and manipulate up to 10 messages with a single action. These batch actions include:

- `SendMessageBatch`
- `DeleteMessageBatch`
- `ChangeMessageVisibilityBatch`

Using batch actions, you can perform multiple operations in a single API call, which helps optimize performance and reduce costs. You can take advantage of batch functionality using the query API or any AWS SDK that supports Amazon SQS batch actions.

Important Details

- **Message Size Limit:** The total size of all messages sent in a single `SendMessageBatch` call cannot exceed 262,144 bytes (256 KiB).
- **Permissions:** You cannot set permissions explicitly for `SendMessageBatch`, `DeleteMessageBatch`, or `ChangeMessageVisibilityBatch`. Instead, setting permissions for `SendMessageBatch`, `DeleteMessageBatch`, or `ChangeMessageVisibilityBatch` sets permissions for the corresponding batch versions of the actions.
- **Console Support:** The Amazon SQS console does not support batch actions. You must use the query API or an AWS SDK to perform batch operations.

Batching message actions

To further optimize costs and efficiency, consider the following best practices for batching message actions:

- **Batch API Actions:** Use the Amazon SQS batch API actions actions to send, receive, and delete messages, and to change the message visibility timeout for multiple messages with a single action. This reduces the number of API calls and associated costs.
- **Client-Side Buffering and Long Polling:** Combine client-side buffering with request batching by using long polling together with the buffered asynchronous client included with the AWS SDK for Java. This approach helps to minimize the number of requests and optimizes the handling of large volumes of messages.

> **Note**
>
> The Amazon SQS Buffered Asynchronous Client doesn't currently support FIFO queues.

Enabling client-side buffering and request batching with Amazon SQS

The AWS SDK for Java includes `AmazonSQSBufferedAsyncClient` which accesses Amazon SQS. This client allows for simple request batching using client-side buffering—calls made from the client are first buffered and then sent as a batch request to Amazon SQS.

Client-side buffering allows up to 10 requests to be buffered and sent as a batch request, decreasing your cost of using Amazon SQS and reducing the number of sent requests. `AmazonSQSBufferedAsyncClient` buffers both synchronous and asynchronous calls. Batched requests and support for long polling can also help increase throughput. For more information, see Increasing throughput using horizontal scaling and action batching with Amazon SQS.

Because `AmazonSQSBufferedAsyncClient` implements the same interface as `AmazonSQSAsyncClient`, migrating from `AmazonSQSAsyncClient` to `AmazonSQSBufferedAsyncClient` typically requires only minimal changes to your existing code.

> **Note**
>
> The Amazon SQS Buffered Asynchronous Client doesn't currently support FIFO queues.

Topics

- Using AmazonSQSBufferedAsyncClient
- Configuring AmazonSQSBufferedAsyncClient

Using AmazonSQSBufferedAsyncClient

Before you begin, complete the steps in Setting up Amazon SQS.

> **Important**
>
> The AWS SDK for Java 2.x isn't currently compatible with the `AmazonSQSBufferedAsyncClient`.

You can create a new `AmazonSQSBufferedAsyncClient` based on `AmazonSQSAsyncClient`, for example:

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync);
```

After you create the new `AmazonSQSBufferedAsyncClient`, you can use it to send multiple requests to Amazon SQS (just as you can with `AmazonSQSAsyncClient`), for example:

```
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);
```

Configuring AmazonSQSBufferedAsyncClient

`AmazonSQSBufferedAsyncClient` is preconfigured with settings that work for most use cases. You can further configure `AmazonSQSBufferedAsyncClient`, for example:

1. Create an instance of the `QueueBufferConfig` class with the required configuration parameters.
2. Provide the instance to the `AmazonSQSBufferedAsyncClient` constructor.

```
// 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 configuration parameters

Parameter	Default value	Description
`longPoll`	`true`	When `longPoll` is set to `true`, `AmazonSQSBufferedAsyncClient` attempts to use long polling when it consumes messages.
`longPollWaitTimeoutSeconds`	20 s	The maximum amount of time (in seconds) which a `ReceiveMessage` call blocks off on the server, waiting for messages to appear in the queue before returning with an empty receive result. **Note** When long polling is disabled, this setting has no effect.
`maxBatchOpenMs`	200 ms	The maximum amount of time (in milliseconds) that an outgoing call waits for other calls with which it batches messages of the same type.

Parameter	Default value	Description
		The higher the setting, the fewer batches are required to perform the same amount of work (however, the first call in a batch has to spend a longer time waiting). When you set this parameter to `0`, submitted requests don't wait for other requests, effectively disabling batching.
`maxBatchSize`	10 requests per batch	The maximum number of messages that are batched together in a single request. The higher the setting, the fewer batches are required to carry out the same number of requests. **Note** 10 requests per batch is the maximum allowed value for Amazon SQS.

Parameter	Default value	Description
maxBatchSizeBytes	256 KiB	The maximum size of a message batch, in bytes, that the client attempts to send to Amazon SQS. **Note** 256 KiB is the maximum allowed value for Amazon SQS.

Parameter	Default value	Description
`maxDoneReceiveBatches`	10 batches	The maximum number of receive batches that `AmazonSQSBufferedAsyncClient` prefetches and stores client-side. The higher the setting, the more receive requests can be satisfied without having to make a call to Amazon SQS (however, the more messages are prefetched, the longer they remain in the buffer, causing their own visibility timeout to expire). **Note** `0` indicates that all message pre-fetching is disabled and messages are consumed only on demand.

Parameter	Default value	Description
`maxInflightOutboundBatches`	5 batches	The maximum number of active outbound batches that can be processed at the same time. The higher the setting, the faster outbound batches can be sent (subject to quotas such as CPU or bandwidth) and the more threads are consumed by `AmazonSQSBufferedAsyncClient` .

Parameter	Default value	Description
`maxInflightReceiveBatches`	10 batches	The maximum number of active receive batches that can be processed at the same time. The higher the setting, the more messages can be received (subject to quotas such as CPU or bandwidth), and the more threads are consumed by `AmazonSQSBufferedAsyncClient` . **Note** `0` indicates that all message pre-fetching is disabled and messages are consumed only on demand.

Parameter	Default value	Description
`visibilityTimeoutSeconds`	-1	When this parameter is set to a positive, non-zero value, the visibility timeout set here overrides the visibility timeout set on the queue from which messages are consumed. **Note** `-1` indicates that the default setting is selected for the queue. You can't set visibility timeout to `0`.

Increasing throughput using horizontal scaling and action batching with Amazon SQS

Amazon SQS queues can deliver very high throughput. For information on throughput quotas, see Amazon SQS message quotas.

To achieve high throughput, you must scale message producers and consumers horizontally (add more producers and consumers).

Topics

- Horizontal scaling
- Action batching
- Working Java example for single-operation and batch requests

Horizontal scaling

Because you access Amazon SQS through an HTTP request-response protocol, the *request latency* (the interval between initiating a request and receiving a response) limits the throughput that you can achieve from a single thread using a single connection. For example, if the latency from an Amazon EC2-based client to Amazon SQS in the same region averages 20 ms, the maximum throughput from a single thread over a single connection averages 50 TPS.

Horizontal scaling involves increasing the number of message producers (which make `SendMessage` requests) and consumers (which make `ReceiveMessage` and `DeleteMessage` requests) in order to increase your overall queue throughput. You can scale horizontally in three ways:

- Increase the number of threads per client
- Add more clients
- Increase the number of threads per client and add more clients

When you add more clients, you achieve essentially linear gains in queue throughput. For example, if you double the number of clients, you also double the throughput.

> **Note**
>
> As you scale horizontally, make sure that your Amazon SQS client has enough connections or threads to support the number of concurrent message producers and consumers that send requests and receive responses. For example, by default, instances of the AWS SDK for Java `AmazonSQSClient` class maintain at most 50 connections to Amazon SQS. To create additional concurrent producers and consumers, you must adjust the maximum number of allowable producer and consumer threads on an `AmazonSQSClientBuilder` object, for example:
>
> ```
> final AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()
>         .withClientConfiguration(new ClientConfiguration()
>                 .withMaxConnections(producerCount + consumerCount))
>         .build();
> ```
>
> For `AmazonSQSAsyncClient`, you also must make sure that enough threads are available. This example only works for Java v. 1.x.

Action batching

Batching performs more work during each round trip to the service (for example, when you send multiple messages with a single `SendMessageBatch` request). The Amazon SQS batch actions are `SendMessageBatch`, `DeleteMessageBatch`, and `ChangeMessageVisibilityBatch`. To take advantage of batching without changing your producers or consumers, you can use the Amazon SQS Buffered Asynchronous Client.

> **Note**
>
> Because `ReceiveMessage` can process 10 messages at a time, there is no `ReceiveMessageBatch` action.

Batching distributes the latency of the batch action over the multiple messages in a batch request, rather than accept the entire latency for a single message (for example, a `SendMessage` request). Because each round trip carries more work, batch requests make more efficient use of threads and connections, improving throughput.

You can combine batching with horizontal scaling to provide throughput with fewer threads, connections, and requests than individual message requests. You can use batched Amazon SQS actions to send, receive, or delete up to 10 messages at a time. Because Amazon SQS charges by the request, batching can substantially reduce your costs.

Batching can introduce some complexity for your application (for example, you application must accumulate messages before sending them, or it sometimes must wait longer for a response). However, batching can be still effective in the following cases:

- Your application generates many messages in a short time, so the delay is never very long.
- A message consumer fetches messages from a queue at its discretion, unlike typical message producers that need to send messages in response to events they don't control.

> **Important**
>
> A batch request might succeed even though individual messages in the batch failed. After a batch request, always check for individual message failures and retry the action if necessary.

Working Java example for single-operation and batch requests

Prerequisites

Add the `aws-java-sdk-sqs.jar`, `aws-java-sdk-ec2.jar`, and `commons-logging.jar` packages to your Java build class path. The following example shows these dependencies in a Maven project `pom.xml` file.

```
<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

The following Java code example implements a simple producer-consumer pattern. The main thread spawns a number of producer and consumer threads that process 1 KB messages for a specified time. This example includes producers and consumers that make single-operation requests and those that make batch requests.

```
/*
 * 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.
            }
        }
    }
}
```

Monitoring volume metrics from the example run

Amazon SQS automatically generates volume metrics for sent, received, and deleted messages. You can access those metrics and others through the **Monitoring** tab for your queue or on the CloudWatch console.

> **Note**
>
> The metrics can take up to 15 minutes after the queue starts to become available.

Using Amazon SQS with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

SDK documentation	Code examples
AWS SDK for C++	AWS SDK for C++ code examples
AWS CLI	AWS CLI code examples
AWS SDK for Go	AWS SDK for Go code examples
AWS SDK for Java	AWS SDK for Java code examples
AWS SDK for JavaScript	AWS SDK for JavaScript code examples

SDK documentation	Code examples
AWS SDK for Kotlin	AWS SDK for Kotlin code examples
AWS SDK for .NET	AWS SDK for .NET code examples
AWS SDK for PHP	AWS SDK for PHP code examples
AWS Tools for PowerShell	Tools for PowerShell code examples
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) code examples
AWS SDK for Ruby	AWS SDK for Ruby code examples
AWS SDK for Rust	AWS SDK for Rust code examples
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP code examples
AWS SDK for Swift	AWS SDK for Swift code examples

Example availability

Can't find what you need? Request a code example by using the **Provide feedback** link at the bottom of this page.

Using JMS with Amazon SQS

The Amazon SQS Java Messaging Library is a Java Message Service (JMS) interface for Amazon SQS that lets you take advantage of Amazon SQS in applications that already use JMS. The interface lets you use Amazon SQS as the JMS provider with minimal code changes. Together with the AWS SDK for Java, the Amazon SQS Java Messaging Library lets you create JMS connections and sessions, as well as producers and consumers that send and receive messages to and from Amazon SQS queues.

The library supports sending and receiving messages to a queue (the JMS point-to-point model) according to the JMS 1.1 specification. The library supports sending text, byte, or object messages synchronously to Amazon SQS queues. The library also supports receiving objects synchronously or asynchronously.

For information about features of the Amazon SQS Java Messaging Library that support the JMS 1.1 specification, see Amazon SQS supported JMS 1.1 implementations and the Amazon SQS FAQs.

Topics

- Prerequisites for working with JMS and Amazon SQS
- Using the Amazon SQS Java Messaging Library
- Using the Java Message Service with other Amazon SQS clients
- Working Java examples for using JMS with Amazon SQS standard queues
- Amazon SQS supported JMS 1.1 implementations

Prerequisites for working with JMS and Amazon SQS

Before you begin, you must have the following prerequisites:

- **SDK for Java**

  There are two ways to include the SDK for Java in your project:

  - Download and install the SDK for Java.
  - Use Maven to get the Amazon SQS Java Messaging Library.

> **Note**
>
> The SDK for Java is included as a dependency.
> The SDK for Java and Amazon SQS Extended Client Library for Java require the J2SE Development Kit 8.0 or later.

For information about downloading the SDK for Java, see SDK for Java.

- **Amazon SQS Java Messaging Library**

  If you don't use Maven, you must add the `amazon-sqs-java-messaging-lib.jar` package to the Java class path. For information about downloading the library, see Amazon SQS Java Messaging Library.

  > **Note**
  >
  > The Amazon SQS Java Messaging Library includes support for Maven and the Spring Framework.
  > For code samples that use Maven, the Spring Framework, and the Amazon SQS Java Messaging Library, see Working Java examples for using JMS with Amazon SQS standard queues.
  >
  > ```
  > <dependency>
  >   <groupId>com.amazonaws</groupId>
  >   <artifactId>amazon-sqs-java-messaging-lib</artifactId>
  >   <version>1.0.4</version>
  >   <type>jar</type>
  > </dependency>
  > ```

- **Amazon SQS Queue**

  Create a queue using the AWS Management Console for Amazon SQS, the `CreateQueue` API, or the wrapped Amazon SQS client included in the Amazon SQS Java Messaging Library.

  - For information about creating a queue with Amazon SQS using either the AWS Management Console or the `CreateQueue` API, see Creating a Queue.
  - For information about using the Amazon SQS Java Messaging Library, see Using the Amazon SQS Java Messaging Library.

Using the Amazon SQS Java Messaging Library

To get started using the Java Message Service (JMS) with Amazon SQS, use the code examples in this section. The following sections show how to create a JMS connection and a session, and how to send and receive a message.

The wrapped Amazon SQS client object included in the Amazon SQS Java Messaging Library checks if an Amazon SQS queue exists. If the queue doesn't exist, the client creates it.

Creating a JMS connection

Before you begin, see the prerequisites in [Prerequisites for working with JMS and Amazon SQS](#).

1. Create a connection factory and call the `createConnection` method against the factory.

   ```
   // 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();
   ```

   The `SQSConnection` class extends `javax.jms.Connection`. Together with the JMS standard connection methods, `SQSConnection` offers additional methods, such as `getAmazonSQSClient` and `getWrappedAmazonSQSClient`. Both methods let you perform administrative operations not included in the JMS specification, such as creating new queues. However, the `getWrappedAmazonSQSClient` method also provides a wrapped version of the Amazon SQS client used by the current connection. The wrapper transforms every exception from the client into an `JMSException`, allowing it to be more easily used by existing code that expects `JMSException` occurrences.

2. You can use the client objects returned from `getAmazonSQSClient` and `getWrappedAmazonSQSClient` to perform administrative operations not included in the JMS specification (for example, you can create an Amazon SQS queue).

   If you have existing code that expects JMS exceptions, then you should use `getWrappedAmazonSQSClient`:

- If you use `getWrappedAmazonSQSClient`, the returned client object transforms all exceptions into JMS exceptions.
- If you use `getAmazonSQSClient`, the exceptions are all Amazon SQS exceptions.

Creating an Amazon SQS queue

The wrapped client object checks if an Amazon SQS queue exists.

If a queue doesn't exist, the client creates it. If the queue does exist, the function doesn't return anything. For more information, see the "Create the queue if needed" section in the TextMessageSender.java example.

To create a standard queue

```
// 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");
}
```

To create a FIFO queue

```
// 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

The name of a FIFO queue must end with the `.fifo` suffix.

For more information about the `ContentBasedDeduplication` attribute, see Exactly-once processing in Amazon SQS.

Sending messages synchronously

1. When the connection and the underlying Amazon SQS queue are ready, create a nontransacted JMS session with `AUTO_ACKNOWLEDGE` mode.

```
// Create the nontransacted session with AUTO_ACKNOWLEDGE mode
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

2. To send a text message to the queue, create a JMS queue identity and a message producer.

```
// 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. Create a text message and send it to the queue.

   - To send a message to a standard queue, you don't need to set any additional parameters.

```
// Create the text message
TextMessage message = session.createTextMessage("Hello World!");

// Send the message
producer.send(message);
System.out.println("JMS Message " + message.getJMSMessageID());
```

   - To send a message to a FIFO queue, you must set the message group ID. You can also set a message deduplication ID. For more information, see Amazon SQS FIFO queue key terms.

```
// 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"));
```

Receiving messages synchronously

1. To receive messages, create a consumer for the same queue and invoke the `start` method.

   You can call the `start` method on the connection at any time. However, the consumer doesn't begin to receive messages until you call it.

   ```
   // Create a consumer for the 'MyQueue'
   MessageConsumer consumer = session.createConsumer(queue);
   // Start receiving incoming messages
   connection.start();
   ```

2. Call the `receive` method on the consumer with a timeout set to 1 second, and then print the contents of the received message.

   - After receiving a message from a standard queue, you can access the contents of the message.

     ```
     // 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());
     }
     ```

   - After receiving a message from a FIFO queue, you can access the contents of the message and other, FIFO-specific message attributes, such as the message group ID, message deduplication ID, and sequence number. For more information, see Amazon SQS FIFO queue key terms.

```
// 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. Close the connection and the session.

```
// Close the connection (and the session).
connection.close();
```

The output looks similar to the following:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

Note

You can use the Spring Framework to initialize these objects.
For additional information, see `SpringExampleConfiguration.xml`, `SpringExample.java`, and the other helper classes in `ExampleConfiguration.java` and `ExampleCommon.java` in the Working Java examples for using JMS with Amazon SQS standard queues section.

For complete examples of sending and receiving objects, see TextMessageSender.java and SyncMessageReceiver.java.

Receiving messages asynchronously

In the example in Using the Amazon SQS Java Messaging Library, a message is sent to MyQueue and received synchronously.

The following example shows how to receive the messages asynchronously through a listener.

1. Implement the MessageListener interface.

```
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();
        }
    }
}
```

   The onMessage method of the MessageListener interface is called when you receive a message. In this listener implementation, the text stored in the message is printed.

2. Instead of explicitly calling the receive method on the consumer, set the message listener of the consumer to an instance of the MyListener implementation. The main thread waits for one second.

```
// 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);
```

The rest of the steps are identical to the ones in the Using the Amazon SQS Java Messaging Library example. For a complete example of an asynchronous consumer, see `AsyncMessageReceiver.java` in Working Java examples for using JMS with Amazon SQS standard queues.

The output for this example looks similar to the following:

```
JMS Message ID:8example-588b-44e5-bbcf-d816example2
Received: Hello World!
```

Using client acknowledge mode

The example in Using the Amazon SQS Java Messaging Library uses `AUTO_ACKNOWLEDGE` mode where every received message is acknowledged automatically (and therefore deleted from the underlying Amazon SQS queue).

1. To explicitly acknowledge the messages after they're processed, you must create the session with `CLIENT_ACKNOWLEDGE` mode.

```
// Create the non-transacted session with CLIENT_ACKNOWLEDGE mode.
Session session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

2. When the message is received, display it and then explicitly acknowledge it.

```
// 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**
>
> In this mode, when a message is acknowledged, all messages received before this message are implicitly acknowledged as well. For example, if 10 messages are received, and only the 10th message is acknowledged (in the order the messages are received), then all of the previous nine messages are also acknowledged.

The rest of the steps are identical to the ones in the Using the Amazon SQS Java Messaging Library example. For a complete example of a synchronous consumer with client acknowledge mode, see `SyncMessageReceiverClientAcknowledge.java` in Working Java examples for using JMS with Amazon SQS standard queues.

The output for this example looks similar to the following:

```
JMS Message ID:4example-aa0e-403f-b6df-5e02example5
Received: Hello World!
Acknowledged: ID:4example-aa0e-403f-b6df-5e02example5
```

Using unordered acknowledge mode

When using `CLIENT_ACKNOWLEDGE` mode, all messages received before an explicitly-acknowledged message are acknowledged automatically. For more information, see Using client acknowledge mode.

The Amazon SQS Java Messaging Library provides another acknowledgement mode. When using `UNORDERED_ACKNOWLEDGE` mode, all received messages must be individually and explicitly acknowledged by the client, regardless of their reception order. To do this, create a session with `UNORDERED_ACKNOWLEDGE` mode.

```
// Create the non-transacted session with UNORDERED_ACKNOWLEDGE mode.
Session session = connection.createSession(false, SQSSession.UNORDERED_ACKNOWLEDGE);
```

The remaining steps are identical to the ones in the Using client acknowledge mode example. For a complete example of a synchronous consumer with `UNORDERED_ACKNOWLEDGE` mode, see `SyncMessageReceiverUnorderedAcknowledge.java`.

In this example, the output looks similar to the following:

```
JMS Message ID:dexample-73ad-4adb-bc6c-4357example7
Received: Hello World!
Acknowledged: ID:dexample-73ad-4adb-bc6c-4357example7
```

Using the Java Message Service with other Amazon SQS clients

Using the Amazon SQS Java Message Service (JMS) Client with the AWS SDK limits Amazon SQS message size to 256 KB. However, you can create a JMS provider using any Amazon SQS client. For example, you can use the JMS Client with the Amazon SQS Extended Client Library for Java to send an Amazon SQS message that contains a reference to a message payload (up to 2 GB) in Amazon S3. For more information, see Managing large Amazon SQS messages using Java and Amazon S3.

The following Java code example creates the JMS provider for the Extended Client Library.

See the prerequisites in Prerequisites for working with JMS and Amazon SQS before testing this example.

```
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);
```

The following Java code example creates the connection factory:

```
// 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();
```

Working Java examples for using JMS with Amazon SQS standard queues

The following code examples show how to use the Java Message Service (JMS) with Amazon SQS standard queues. For more information about working with FIFO queues, see To create a FIFO queue, Sending messages synchronously, and Receiving messages synchronously. (Receiving messages synchronously is the same for standard and FIFO queues. However, messages in FIFO queues contain more attributes.)

See the prerequisites in Prerequisites for working with JMS and Amazon SQS before testing the following examples.

ExampleConfiguration.java

The following Java SDK v 1.x code example sets the default queue name, the region, and the credentials to be used with the other Java examples.

```
/*
 * 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

The following Java code example creates a text message producer.

```
/*
 * 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

The following Java code example creates a synchronous message consumer.

```
/*
 * 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

The following Java code example creates an asynchronous message consumer.

```
/*
 * 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

The following Java code example creates a synchronous consumer with client acknowledge mode.

```
/*
 * 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

The following Java code example creates a synchronous consumer with unordered acknowledge mode.

```
/*
 * 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

The following XML code example is a bean configuration file for SpringExample.java.

```
<!--
    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

The following Java code example uses the bean configuration file to initialize your objects.

```
/*
 * 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

The following Java code example checks if an Amazon SQS queue exists and then creates one if it doesn't. It also includes example logging code.

```
/*
 * 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 supported JMS 1.1 implementations

The Amazon SQS Java Messaging Library supports the following JMS 1.1 implementations. For more information about the supported features and capabilities of the Amazon SQS Java Messaging Library, see the Amazon SQS FAQ.

Supported common interfaces

- `Connection`
- `ConnectionFactory`
- `Destination`
- `Session`
- `MessageConsumer`
- `MessageProducer`

Supported message types

- `ByteMessage`
- `ObjectMessage`
- `TextMessage`

Supported message acknowledgment modes

- AUTO_ACKNOWLEDGE
- CLIENT_ACKNOWLEDGE
- DUPS_OK_ACKNOWLEDGE
- UNORDERED_ACKNOWLEDGE

> **Note**
>
> The UNORDERED_ACKNOWLEDGE mode isn't part of the JMS 1.1 specification. This mode helps Amazon SQS allow a JMS client to explicitly acknowledge a message.

JMS-defined headers and reserved properties

For sending messages

When you send messages, you can set the following headers and properties for each message:

- JMSXGroupID (required for FIFO queues, not allowed for standard queues)
- JMS_SQS_DeduplicationId (optional for FIFO queues, not allowed for standard queues)

After you send messages, Amazon SQS sets the following headers and properties for each message:

- JMSMessageID
- JMS_SQS_SequenceNumber (only for FIFO queues)

For receiving messages

When you receive messages, Amazon SQS sets the following headers and properties for each message:

- JMSDestination
- JMSMessageID
- JMSRedelivered

- `JMSXDeliveryCount`
- `JMSXGroupID` (only for FIFO queues)
- `JMS_SQS_DeduplicationId` (only for FIFO queues)
- `JMS_SQS_SequenceNumber` (only for FIFO queues)

Amazon SQS tutorials

This section provides tutorials that you can use to explore Amazon SQS features and functionality.

Topics

- Creating an Amazon SQS queue using AWS CloudFormation
- Tutorial: Sending a message to an Amazon SQS queue from Amazon Virtual Private Cloud

Creating an Amazon SQS queue using AWS CloudFormation

You can use the AWS CloudFormation console and a JSON (or YAML) template to create an Amazon SQS queue. For more information, see Working with AWS CloudFormation Templates and the `AWS::SQS::Queue` Resource in the *AWS CloudFormation User Guide*.

To use AWS CloudFormation to create an Amazon SQS queue.

1. Copy the following JSON code to a file named `MyQueue.json`. To create a standard queue, omit the `FifoQueue` and `ContentBasedDeduplication` properties. For more information on content-based deduplication, see Exactly-once processing in Amazon SQS.

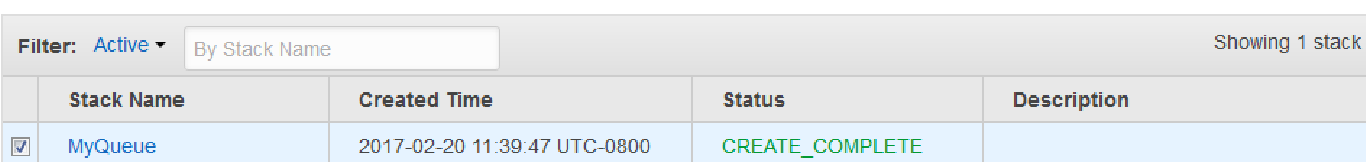
   > **Note**
   >
   > The name of a FIFO queue must end with the `.fifo` suffix.

```
{
   "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. Sign in to the [AWS CloudFormation console](), and then choose **Create Stack**.
3. On the **Specify Template** panel, choose **Upload a template file**, choose your `MyQueue.json` file, and then choose **Next**.
4. On the **Specify Details** page, type `MyQueue` for **Stack Name**, and then choose **Next**.
5. On the **Options** page, choose **Next**.
6. On the **Review** page, choose **Create**.

   AWS CloudFormation begins to create the `MyQueue` stack and displays the **CREATE_IN_PROGRESS** status. When the process is complete, AWS CloudFormation displays the **CREATE_COMPLETE** status.



7. (Optional) To display the name, URL, and ARN of the queue, choose the name of the stack and then on the next page expand the **Outputs** section.

Tutorial: Sending a message to an Amazon SQS queue from Amazon Virtual Private Cloud

In this tutorial, you learn how to send messages to an Amazon SQS queue over a secure, private network. This network consists of a VPC that contains an Amazon EC2 instance. The instance connects to Amazon SQS through an *interface VPC endpoint*, allowing you to connect to the Amazon EC2 instance and send messages to the Amazon SQS queue even though the network is disconnected from the public internet. For more information, see Amazon Virtual Private Cloud endpoints for Amazon SQS.

> **Important**
>
> - You can use Amazon Virtual Private Cloud only with HTTPS Amazon SQS endpoints.
> - When you configure Amazon SQS to send messages from Amazon VPC, you must enable private DNS and specify endpoints in the format `sqs.us-east-2.amazonaws.com`.
> - Private DNS doesn't support legacy endpoints such as `queue.amazonaws.com` or `us-east-2.queue.amazonaws.com`.

Topics

Step 1: Create an Amazon EC2 key pair

A *key pair* lets you connect to an Amazon EC2 instance. It consists of a public key that encrypts your login information and a private key that decrypts it.

1. Sign in to the [Amazon EC2 console](https://console.aws.amazon.com/ec2/).
2. On the navigation menu, under **Network & Security**, choose **Key Pairs**.
3. Choose **Create Key Pair**.
4. In the **Create Key Pair** dialog box, for **Key pair name**, enter `SQS-VPCE-Tutorial-Key-Pair`, and then choose **Create**.
5. Your browser downloads the private key file `SQS-VPCE-Tutorial-Key-Pair.pem` automatically.

   > **Important**
   >
   > Save this file in a safe place. EC2 does not generate a `.pem` file for the same key pair a second time.

6. To allow an SSH client to connect to your EC2 instance, set the permissions for your private key file so that only your user can have read permissions for it, for example:

   ```
   chmod 400 SQS-VPCE-Tutorial-Key-Pair.pem
   ```

Step 2: Create AWS resources

To set up the necessary infrastructure, you must use an AWS CloudFormation *template*, which is a blueprint for creating a *stack* comprised of AWS resources, such as Amazon EC2 instances and Amazon SQS queues.

The stack for this tutorial includes the following resources:

- A VPC and the associated networking resources, including a subnet, a security group, an internet gateway, and a route table
- An Amazon EC2 instance launched into the VPC subnet
- An Amazon SQS queue

1. Download the AWS CloudFormation template named `SQS-VPCE-Tutorial-CloudFormation.yaml` from GitHub.
2. Sign in to the AWS CloudFormation console.
3. Choose **Create Stack**.

4. On the **Select Template** page, choose **Upload a template to Amazon S3**, select the `SQS-VPCE-SQS-Tutorial-CloudFormation.yaml` file, and then choose **Next**.
5. On the **Specify Details** page, do the following:

   a. For **Stack name**, enter `SQS-VPCE-Tutorial-Stack`.

   b. For **KeyName**, choose **SQS-VPCE-Tutorial-Key-Pair**.

   c. Choose **Next**.
6. On the **Options** page, choose **Next**.
7. On the **Review** page, in the **Capabilities** section, choose **I acknowledge that AWS CloudFormation might create IAM resources with custom names.**, and then choose **Create**.

AWS CloudFormation begins to create the stack and displays the **CREATE_IN_PROGRESS** status. When the process is complete, AWS CloudFormation displays the **CREATE_COMPLETE** status.

Step 3: Confirm that your EC2 instance isn't publicly accessible

Your AWS CloudFormation template launches an EC2 instance named `SQS-VPCE-Tutorial-EC2-Instance` into your VPC. This EC2 instance doesn't allow outbound traffic and isn't able to send messages to Amazon SQS. To verify this, you must connect to the instance, try to connect to a public endpoint, and then try to message Amazon SQS.

1. Sign in to the Amazon EC2 console.
2. On the navigation menu, under **Instances**, choose **Instances**.
3. Select **SQS-VPCE-Tutorial-EC2Instance**.
4. Copy the hostname under **Public DNS (IPv4)**, for example, **ec2-203-0-113-0.us-west-2.compute.amazonaws.com**.
5. From the directory that contains the key pair that you created earlier, connect to the instance using the following command, for example:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-east-2.compute.amazonaws.com
```

6. Try to connect to any public endpoint, for example:

```
ping amazon.com
```

The connection attempt fails, as expected.

7. Sign in to the [Amazon SQS console](#).
8. From the list of queues, select the queue created by your AWS CloudFormation template, for example, **VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK**.
9. On the **Details** table, copy the URL, for example, **https://sqs.us-east-2.amazonaws.com/123456789012/**.
10. From your EC2 instance, try to publish a message to the queue using the following command, for example:

```
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."
```

The sending attempt fails, as expected.

> **Important**
>
> Later, when you create a VPC endpoint for Amazon SQS, your sending attempt will succeed.

Step 4: Create an Amazon VPC endpoint for Amazon SQS

To connect your VPC to Amazon SQS, you must define an interface VPC endpoint. After you add the endpoint, you can use the Amazon SQS API from the EC2 instance in your VPC. This allows you to send messages to a queue within the AWS network without crossing the public internet.

> **Note**
>
> The EC2 instance still doesn't have access to other AWS services and endpoints on the internet.

1. Sign in to the [Amazon VPC console](#).
2. On the navigation menu, choose **Endpoints**.
3. Choose **Create Endpoint**.

4. On the **Create Endpoint** page, for **Service Name**, choose the service name for Amazon SQS.

> **Note**
>
> The service names vary based on the current AWS Region. For example, if you are in US East (Ohio), the service name is **com.amazonaws.*us-east-2*.sqs**.

5. For **VPC**, choose **SQS-VPCE-Tutorial-VPC**.
6. For **Subnets**, choose the subnet whose **Subnet ID** contains **SQS-VPCE-Tutorial-Subnet**.
7. For **Security group**, choose **Select security groups**, and then choose the security group whose **Group Name** contains **SQS VPCE Tutorial Security Group**.
8. Choose **Create endpoint**.

   The interface VPC endpoint is created and its ID is displayed, for example, **vpce-0ab1cdef2ghi3j456k**.
9. Choose **Close**.

   The Amazon VPC console opens the **Endpoints** page.

Amazon VPC begins to create the endpoint and displays the **pending** status. When the process is complete, Amazon VPC displays the **available** status.

Step 5: Send a message to your Amazon SQS queue

Now that your VPC includes an endpoint for Amazon SQS, you can connect to your EC2 instance and send messages to your queue.

1. Reconnect to your EC2 instance, for example:

```
ssh -i SQS-VPCE-Tutorial-Key-Pair.pem ec2-user@ec2-203-0-113-0.us-
east-2.compute.amazonaws.com
```

2. Try to publish a message to the queue again using the following command, for example:

```
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."
```

The sending attempt succeeds and the MD5 digest of the message body and the message ID are displayed, for example:

```
{
 "MD5OfMessageBody": "a1bcd2ef3g45hi678j90klmn12p34qr5",
 "MessageId": "12345a67-8901-2345-bc67-d890123e45fg"
}
```

For information about receiving and deleting the message from the queue created by your AWS CloudFormation template (for example, **VPCE-SQS-Tutorial-Stack-CFQueue-1ABCDEFGH2IJK**), see Receiving and deleting a message in Amazon SQS .

For information about deleting your resources, see the following:

- Deleting a VPC Endpoint in the *Amazon VPC User Guide*
- Deleting an Amazon SQS queue
- Terminate Your Instance in the *Amazon EC2 User Guide*
- Deleting Your VPC in the *Amazon VPC User Guide*
- Deleting a Stack on the AWS CloudFormation Console in the *AWS CloudFormation User Guide*
- Deleting Your Key Pair in the *Amazon EC2 User Guide*

Code examples for Amazon SQS using AWS SDKs

The following code examples show how to use Amazon SQS with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios and cross-service examples.

Scenarios are code examples that show you how to accomplish a specific task by calling multiple functions within the same service.

Cross-service examples are sample applications that work across multiple AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Get started

Hello Amazon SQS

The following code examples show how to get started using Amazon SQS.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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();
        }
    }
}
```

- For API details, see ListQueues in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Code for the CMakeLists.txt CMake file.

```
# 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})
```

Code for the hello_sqs.cpp source file.

```
#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;
}
```

- For API details, see ListQueues in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/gov2/sqs#code-examples).

```
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() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 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(context.TODO())
  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)
    }
  }
}
```

- For API details, see ListQueues in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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);
        }
    }
}
```

- For API details, see ListQueues in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Initialize an Amazon SQS client and list queues.

```
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"));
};
```

- For API details, see ListQueues in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient
            .listQueuesPaginated { }
            .transform { it.queueUrls?.forEach { queue -> emit(queue) } }
            .collect { queue ->
                println("The Queue URL is $queue")
            }
    }
}
```

- For API details, see ListQueues in *AWS SDK for Kotlin API reference*.

Code examples

- Actions for Amazon SQS using AWS SDKs
  - Use AddPermission with an AWS SDK or CLI
  - Use ChangeMessageVisibility with an AWS SDK or CLI
  - Use ChangeMessageVisibilityBatch with an AWS SDK or CLI
  - Use CreateQueue with an AWS SDK or CLI
  - Use DeleteMessage with an AWS SDK or CLI
  - Use DeleteMessageBatch with an AWS SDK or CLI
  - Use DeleteQueue with an AWS SDK or CLI
  - Use GetQueueAttributes with an AWS SDK or CLI
  - Use GetQueueUrl with an AWS SDK or CLI
  - Use ListDeadLetterSourceQueues with an AWS SDK or CLI
  - Use ListQueues with an AWS SDK or CLI
  - Use PurgeQueue with an AWS SDK or CLI
  - Use ReceiveMessage with an AWS SDK or CLI

Actions for Amazon SQS using AWS SDKs

The following code examples demonstrate how to perform individual Amazon SQS actions with AWS SDKs. These excerpts call the Amazon SQS API and are code excerpts from larger programs that must be run in context. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Simple Queue Service (Amazon SQS) API Reference.

Examples

Use `AddPermission` with an AWS SDK or CLI

The following code examples show how to use `AddPermission`.

CLI

AWS CLI

To add a permission to a queue

This example enables the specified AWS account to send messages to the specified queue.

Command:

```
aws sqs add-permission --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue --aws-account-ids 12345EXAMPLE --actions SendMessage
```

Output:

```
None.
```

- For API details, see AddPermission in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example allows the specified AWS account to send messages from the specified queue.

```
Add-SQSPermission -Action SendMessage -AWSAccountId 80398EXAMPLE
 -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- For API details, see AddPermission in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ChangeMessageVisibility` with an AWS SDK or CLI

The following code examples show how to use `ChangeMessageVisibility`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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();
}
```

- For API details, see ChangeMessageVisibility in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To change a message's timeout visibility

This example changes the specified message's timeout visibility to 10 hours (10 hours * 60 minutes * 60 seconds).

Command:

```
aws sqs change-message-visibility --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBTpyI...t6HyQg== --
visibility-timeout 36000
```

Output:

```
None.
```

- For API details, see ChangeMessageVisibility in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Receive an Amazon SQS message and change its timeout visibility.

```
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;
};
```

- For API details, see ChangeMessageVisibility in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Receive an Amazon SQS message and change its timeout visibility.

```
// 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");
    }
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ChangeMessageVisibility in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example changes the visibility timeout for the message with the specified receipt handle in the specified queue to 10 hours (10 hours * 60 minutes * 60 seconds = 36000 seconds).

```
Edit-SQSMessageVisibility -QueueUrl https://sqs.us-
east-1.amazonaws.com/8039EXAMPLE/MyQueue -ReceiptHandle AQEBgGDh...J/Iqww== -
VisibilityTimeout 36000
```

- For API details, see ChangeMessageVisibility in *AWS Tools for PowerShell Cmdlet Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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_message_result_before = sqs.receive_message({
    queue_url: queue_url,
    max_number_of_messages: 10 # Receive up to 10 messages, if there are that
 many.
  })

  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 '#{receive_queue_name}', as it
 does not exist."
end
```

- For API details, see ChangeMessageVisibility in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ChangeMessageVisibilityBatch` with an AWS SDK or CLI

The following code examples show how to use `ChangeMessageVisibilityBatch`.

CLI

AWS CLI

To change multiple messages' timeout visibilities as a batch

This example changes the 2 specified messages' timeout visibilities to 10 hours (10 hours * 60 minutes * 60 seconds).

Command:

```
aws sqs change-message-visibility-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://change-message-visibility-batch.json
```

Input file (change-message-visibility-batch.json):

```
[
  {
    "Id": "FirstMessage",
        "ReceiptHandle": "AQEBhz2q...Jf3kaw==",
        "VisibilityTimeout": 36000
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBkTUH...HifSnw==",
        "VisibilityTimeout": 36000
  }
]
```

Output:

```
{
  "Successful": [
    {
      "Id": "SecondMessage"
    },
    {
      "Id": "FirstMessage"
    }
  ]
}
```

- For API details, see ChangeMessageVisibilityBatch in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example changes the visibility timeout for 2 messages with the specified receipt handles in the specified queue. The first message's visibility timeout is changed to 10 hours (10 hours * 60 minutes * 60 seconds = 36000 seconds). The second message's visibility timeout is changed to 5 hours (5 hours * 60 minutes * 60 seconds = 18000 seconds).

```
$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
```

Output:

```
Failed    Successful

------    ----------

{}        {Request2, Request1}
```

- For API details, see ChangeMessageVisibilityBatch in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateQueue` with an AWS SDK or CLI

The following code examples show how to use `CreateQueue`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Publish messages to queues
- Send and receive batches of messages

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a queue with a specific name.

```
    /// <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;
    }
```

Create an Amazon SQS queue and send a message to it.

```
    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;
        }
    }
```

- For API details, see CreateQueue in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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();
}
```

- For API details, see CreateQueue in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To create a queue

This example creates a queue with the specified name, sets the message retention period to 3 days (3 days * 24 hours * 60 minutes * 60 seconds), and sets the queue's dead letter queue to the specified queue with a maximum receive count of 1,000 messages.

Command:

```
aws sqs create-queue --queue-name MyQueue --attributes file://create-queue.json
```

Input file (create-queue.json):

```
{
  "RedrivePolicy": "{\"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-1:80398EXAMPLE:MyDeadLetterQueue\",\"maxReceiveCount\":\"1000\"}",
  "MessageRetentionPeriod": "259200"
}
```

Output:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- For API details, see CreateQueue in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

```
// 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(queueName string, isFifoQueue bool) (string,
 error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(context.TODO(), &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
}
```

- For API details, see [CreateQueue](#) in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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);
        }
    }
}
```

- For API details, see CreateQueue in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

Create an Amazon SQS standard queue.

```
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;
};
```

Create an Amazon SQS queue with long polling.

```
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;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreateQueue in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an Amazon SQS standard queue.

```
// 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);
  }
});
```

Create an Amazon SQS queue that waits for a message to arrive.

```
// 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);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see CreateQueue in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun createQueue(queueNameVal: String): String {
    println("Create Queue")
    val createQueueRequest =
        CreateQueueRequest {
            queueName = queueNameVal
        }

    SqsClient { 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()
    }
}
```

- For API details, see CreateQueue in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: This example creates a queue with the specified name.

```
New-SQSQueue -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- For API details, see CreateQueue in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see CreateQueue in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
# 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__
```

- For API details, see CreateQueue in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an Amazon SQS standard queue.

```
    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.
```

Create an Amazon SQS queue that waits for a message to arrive.

```
    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.
```

- For API details, see CreateQueue in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteMessage` with an AWS SDK or CLI

The following code examples show how to use `DeleteMessage`.

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Receive a message from an Amazon SQS queue and then delete the message.

```
        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;
        }
```

```
    }
```

- For API details, see DeleteMessage in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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();
}
```

- For API details, see DeleteMessage in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete a message

This example deletes the specified message.

Command:

```
aws sqs delete-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --receipt-handle AQEBRXTo...q2doVA==
```

Output:

```
None.
```

- For API details, see DeleteMessage in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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);
        }
```

- For API details, see DeleteMessage in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Receive and delete Amazon SQS messages.

```
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,
        })),
      }),
    );
  }
};
```

- For API details, see DeleteMessage in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Receive and delete Amazon SQS messages.

```
// 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);
      }
```

```
    });
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-send-receive-messages.html#sqs-examples-send-receive-messages-receiving).
- For API details, see [DeleteMessage](https://docs.aws.amazon.com/goto/AWSJavaScriptSDK/sqs-2012-11-05/DeleteMessage) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/kotlin/services/sqs#code-examples).

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
```

```
    }
}
```

- For API details, see DeleteMessage in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes the message with the specified receipt handle from the specified queue.

```
Remove-SQSMessage -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue -ReceiptHandle AQEBd329...v6gl8Q==
```

- For API details, see DeleteMessage in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see DeleteMessage in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteMessageBatch` with an AWS SDK or CLI

The following code examples show how to use `DeleteMessageBatch`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Process S3 event notifications
- Publish messages to queues
- Send and receive batches of messages

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <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();
    }
```

- For API details, see DeleteMessageBatch in *AWS SDK for .NET API Reference*.

C++

SDK for C++

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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;
            }
```

- For API details, see DeleteMessageBatch in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete multiple messages as a batch

This example deletes the specified messages.

Command:

```
aws sqs delete-message-batch --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://delete-message-
batch.json
```

Input file (delete-message-batch.json):

```
[
  {
        "Id": "FirstMessage",
        "ReceiptHandle": "AQEB1mgl...Z4GuLw=="
  },
  {
    "Id": "SecondMessage",
        "ReceiptHandle": "AQEBLsYM...VQubAA=="
  }
]
```

Output:

```
{
  "Successful": [
    {
      "Id": "FirstMessage"
    },
    {
      "Id": "SecondMessage"
    }
  ]
}
```

- For API details, see DeleteMessageBatch in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

```
// 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(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(context.TODO(),
 &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
}
```

- For API details, see DeleteMessageBatch in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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,
        })),
      }),
    );
  }
};
```

- For API details, see DeleteMessageBatch in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes 2 messages with the specified receipt handles from the specified queue.

```
$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
```

Output:

```
Failed    Successful

------    ----------

{}        {Request1, Request2}
```

- For API details, see DeleteMessageBatch in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see DeleteMessageBatch in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteQueue` with an AWS SDK or CLI

The following code examples show how to use `DeleteQueue`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Publish messages to queues
- Send and receive batches of messages

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Delete a queue by using its URL.

```
/// <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;
}
```

- For API details, see [DeleteQueue](#) in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
        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();
}
```

- For API details, see DeleteQueue in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete a queue

This example deletes the specified queue.

Command:

```
aws sqs delete-queue --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewerQueue
```

Output:

```
None.
```

- For API details, see DeleteQueue in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// 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(queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(context.TODO(), &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
}
```

- For API details, see DeleteQueue in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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);
        }
    }
}
```

- For API details, see DeleteQueue in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete an Amazon SQS queue.

```
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;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteQueue in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Delete an Amazon SQS queue.

```
// 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);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see DeleteQueue in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun deleteMessages(queueUrlVal: String) {
    println("Delete Messages from $queueUrlVal")

    val purgeRequest =
        PurgeQueueRequest {
            queueUrl = queueUrlVal
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient.deleteQueue(request)
        println("$queueUrlVal was deleted!")
    }
}
```

- For API details, see DeleteQueue in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes the specified queue.

```
Remove-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyQueue
```

- For API details, see DeleteQueue in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see DeleteQueue in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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)
```

- For API details, see DeleteQueue in *AWS SDK for Ruby API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
```

```
        lo_sqs->deletequeue( iv_queueurl = iv_queue_url ).
        MESSAGE 'SQS queue deleted' TYPE 'I'.
    ENDTRY.
```

- For API details, see DeleteQueue in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetQueueAttributes` with an AWS SDK or CLI

The following code examples show how to use `GetQueueAttributes`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Process S3 event notifications
- Publish messages to queues

.NET

AWS SDK for .NET

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <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;
    }
```

- For API details, see GetQueueAttributes in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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;


                }
```

- For API details, see GetQueueAttributes in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To get a queue's attributes

This example gets all of the specified queue's attributes.

Command:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-names All
```

Output:

```
{
  "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"
  }
}
```

This example gets only the specified queue's maximum message size and visibility timeout attributes.

Command:

```
aws sqs get-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attribute-
names MaximumMessageSize VisibilityTimeout
```

Output:

```
{
  "Attributes": {
    "VisibilityTimeout": "30",
    "MaximumMessageSize": "262144"
  }
}
```

- For API details, see [GetQueueAttributes](#) in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
// 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(queueUrl string) (string, error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(context.TODO(),
 &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
}
```

- For API details, see GetQueueAttributes in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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;
};
```

- For API details, see GetQueueAttributes in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists all attributes for the specified queue.

```
Get-SQSQueueAttribute -AttributeName All -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
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]...}
```

Example 2: This example lists separately only the specified attributes for the specified queue.

```
Get-SQSQueueAttribute -AttributeName MaximumMessageSize, VisibilityTimeout -
QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
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]}
```

- For API details, see GetQueueAttributes in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetQueueUrl` with an AWS SDK or CLI

The following code examples show how to use `GetQueueUrl`.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    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.");
            }
        }
    }
```

- For API details, see GetQueueUrl in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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();
```

```
}
```

- For API details, see GetQueueUrl in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To get a queue URL

This example gets the specified queue's URL.

Command:

```
aws sqs get-queue-url --queue-name MyQueue
```

Output:

```
{
  "QueueUrl": "https://queue.amazonaws.com/80398EXAMPLE/MyQueue"
}
```

- For API details, see GetQueueUrl in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
            GetQueueUrlResponse getQueueUrlResponse = sqsClient

 .getQueueUrl(GetQueueUrlRequest.builder().queueName(queueName).build());
            return getQueueUrlResponse.queueUrl();
```

- For API details, see GetQueueUrl in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get the URL for an Amazon SQS queue.

```
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;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetQueueUrl in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Get the URL for an Amazon SQS queue.

```
// 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);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see GetQueueUrl in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the URL of the queue with the specified name.

```
Get-SQSQueueUrl -QueueName MyQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- For API details, see GetQueueUrl in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see GetQueueUrl in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    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.
```

- For API details, see GetQueueUrl in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListDeadLetterSourceQueues` with an AWS SDK or CLI

The following code examples show how to use `ListDeadLetterSourceQueues`.

CLI

AWS CLI

To list dead letter source queues

This example lists the queues that are associated with the specified dead letter source queue.

Command:

```
aws sqs list-dead-letter-source-queues --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Output:

```
{
  "queueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- For API details, see ListDeadLetterSourceQueues in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists the URLs of any queues that rely on the specified queue as their dead letter queue.

```
Get-SQSDeadLetterSourceQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue
```

Output:

```
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyOtherQueue
```

- For API details, see ListDeadLetterSourceQueues in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListQueues` with an AWS SDK or CLI

The following code examples show how to use `ListQueues`.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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;
}
```

- For API details, see ListQueues in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To list queues

This example lists all queues.

Command:

```
aws sqs list-queues
```

Output:

```
{
  "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"
  ]
}
```

This example lists only queues that start with "My".

Command:

```
aws sqs list-queues --queue-name-prefix My
```

Output:

```
{
  "QueueUrls": [
    "https://queue.amazonaws.com/80398EXAMPLE/MyDeadLetterQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyQueue",
    "https://queue.amazonaws.com/80398EXAMPLE/MyOtherQueue"
  ]
}
```

- For API details, see ListQueues in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```

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() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 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(context.TODO())
  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)
  }
 }
}
```

- For API details, see ListQueues in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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);
        }
```

- For API details, see ListQueues in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List your Amazon SQS queues.

```
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);
    urls.forEach((url) => console.log(url));
  }
```

```
  return urls;
};
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListQueues](#) in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

List your Amazon SQS queues.

```
// 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);
  }
});
```

- For more information, see [AWS SDK for JavaScript Developer Guide](#).
- For API details, see [ListQueues](#) in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listQueues() {
    println("\nList Queues")

    val prefix = "que"
    val listQueuesRequest =
        ListQueuesRequest {
            queueNamePrefix = prefix
        }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.listQueues(listQueuesRequest)
        response.queueUrls?.forEach { url ->
            println(url)
        }
    }
}
```

- For API details, see ListQueues in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists all queues.

```
Get-SQSQueue
```

Output:

```
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
```

Example 2: This example lists any queues that start with the specified name.

```
Get-SQSQueue -QueueNamePrefix My
```

Output:

```
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
```

- For API details, see ListQueues in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see [ListQueues](#) in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
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
```

- For API details, see ListQueues in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Retrieve the first Amazon SQS queue listed in the Region.

```
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())
}
```

- For API details, see [ListQueues](#) in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

```
    TRY.
        oo_result = lo_sqs->listqueues( ).        " oo_result is returned for
 testing purposes. "
        MESSAGE 'Retrieved list of queues.' TYPE 'I'.
    ENDTRY.
```

- For API details, see ListQueues in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use PurgeQueue with an AWS SDK or CLI

The following code examples show how to use PurgeQueue.

CLI

AWS CLI

To purge a queue

This example deletes all messages in the specified queue.

Command:

```
aws sqs purge-queue --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/
MyNewQueue
```

Output:

```
None.
```

- For API details, see PurgeQueue in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example deletes all messages from the specified queue.

```
Clear-SQSQueue -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- For API details, see PurgeQueue in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ReceiveMessage` with an AWS SDK or CLI

The following code examples show how to use `ReceiveMessage`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Process S3 event notifications
- Publish messages to queues
- Send and receive batches of messages

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Receive messages from a queue by using its URL.

```
    /// <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;
    }
```

Receive a message from an Amazon SQS queue, and then delete the message.

```
        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;
        }
```

```
    }
```

- For API details, see ReceiveMessage in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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();
}
```

- For API details, see ReceiveMessage in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To receive a message

This example receives up to 10 available messages, returning all available attributes.

Command:

```
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
```

Output:

```
{
```

```
  "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"
        }
      }
    }
  ]
}
```

This example receives the next available message, returning only the SenderId and SentTimestamp attributes as well as the PostalCode message attribute.

Command:

```
aws sqs receive-message --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --attribute-
names SenderId SentTimestamp --message-attribute-names PostalCode
```

Output:

```
{
  "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"
        }
      }
    }
  ]
}
```

- For API details, see ReceiveMessage in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// 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(queueUrl string, maxMessages int32, waitTime
 int32) ([]types.Message, error) {
```

```
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(context.TODO(),
 &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
}
```

- For API details, see ReceiveMessage in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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;
```

- For API details, see ReceiveMessage in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Receive a message from an Amazon SQS queue.

```
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,
        })),
      }),
    );
  }
};
```

Receive a message from an Amazon SQS queue using long-poll support.

```
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;
};
```

- For API details, see ReceiveMessage in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Receive a message from an Amazon SQS queue using long-poll support.

```
// 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);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see ReceiveMessage in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun receiveMessages(queueUrlVal: String?) {
    println("Retrieving messages from $queueUrlVal")

    val receiveMessageRequest =
        ReceiveMessageRequest {
            queueUrl = queueUrlVal
            maxNumberOfMessages = 5
        }

    SqsClient { region = "us-east-1" }.use { sqsClient ->
        val response = sqsClient.receiveMessage(receiveMessageRequest)
        response.messages?.forEach { message ->
            println(message.body)
        }
    }
}
```

- For API details, see ReceiveMessage in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: This example lists information for up to the next 10 messages to be received for the specified queue. The information will contain values for the specified message attributes, if they exist.

```
Receive-SQSMessage -AttributeName SenderId, SentTimestamp -MessageAttributeName
 StudentName, StudentGrade -MessageCount 10 -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

Output:

```
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==
```

- For API details, see ReceiveMessage in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see ReceiveMessage in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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__
```

- For API details, see ReceiveMessage in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see [ReceiveMessage](#) in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](#).

Receive a message from an Amazon SQS queue.

```
    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.
```

Receive a message from an Amazon SQS queue using long-poll support.

```
    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.
```

- For API details, see ReceiveMessage in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `RemovePermission` with an AWS SDK or CLI

The following code examples show how to use `RemovePermission`.

CLI

AWS CLI

To remove a permission

This example removes the permission with the specified label from the specified queue.

Command:

```
aws sqs remove-permission --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue --label SendMessagesFromMyQueue
```

Output:

```
None.
```

- For API details, see RemovePermission in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell

Example 1: This example removes the permission settings with the specified label from the specified queue.

```
Remove-SQSPermission -Label SendMessagesFromMyQueue -QueueUrl https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- For API details, see RemovePermission in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use SendMessage with an AWS SDK or CLI

The following code examples show how to use `SendMessage`.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an Amazon SQS queue and send a message to it.

```
    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;
        }
    }
```

- For API details, see SendMessage in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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();
}
```

- For API details, see SendMessage in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To send a message

This example sends a message with the specified message body, delay period, and message attributes, to the specified queue.

Command:

```
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
```

Input file (send-message.json):

```
{
  "City": {
    "DataType": "String",
    "StringValue": "Any City"
  },
  "Greeting": {
    "DataType": "Binary",
    "BinaryValue": "Hello, World!"
```

```
  },
  "Population": {
    "DataType": "Number",
    "StringValue": "1250800"
  }
}
```

Output:

```
{
  "MD5OfMessageBody": "51b0a325...39163aa0",
  "MD5OfMessageAttributes": "00484c68...59e48f06",
  "MessageId": "da68f62c-0c07-4bee-bf5f-7e856EXAMPLE"
}
```

- For API details, see SendMessage in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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);
        }
    }
}
```

- For API details, see SendMessage in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a message to an Amazon SQS queue.

```
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;
};
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see SendMessage in *AWS SDK for JavaScript API Reference*.

SDK for JavaScript (v2)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a message to an Amazon SQS queue.

```
// 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);
  }
});
```

- For more information, see AWS SDK for JavaScript Developer Guide.
- For API details, see SendMessage in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [AWS Code Examples Repository](https://github.com/awsdocs/aws-doc-sdk-examples).

```
suspend fun sendMessages(
    queueUrlVal: String,
    message: String,
) {
    println("Sending multiple messages")
    println("\nSend message")
    val sendRequest =
        SendMessageRequest {
            queueUrl = queueUrlVal
            messageBody = message
            delaySeconds = 10
        }

    SqsClient { 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 { region = "us-east-1" }.use { sqsClient ->
        sqsClient.sendMessageBatch(sendMessageBatchRequest)
        println("Batch message were successfully sent.")
    }
}
```

- For API details, see SendMessage in *AWS SDK for Kotlin API reference*.

PowerShell

Tools for PowerShell

Example 1: This example sends a message with the specified attributes and message body to the specified queue with message delivery delayed for 10 seconds.

```
$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
```

Output:

```
MD5OfMessageAttributes              MD5OfMessageBody                    MessageId

----------------------              ----------------                    ---------

1d3e51347bc042efbdf6dda31EXAMPLE    51b0a3256d59467f973009b73EXAMPLE    c35fed8f-
c739-4d0c-818b-1820eEXAMPLE
```

- For API details, see SendMessage in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see SendMessage in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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__
```

- For API details, see SendMessage in *AWS SDK for Ruby API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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(())
}
```

- For API details, see SendMessage in *AWS SDK for Rust API reference*.

SAP ABAP

SDK for SAP ABAP

Note

There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    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.
```

- For API details, see SendMessage in *AWS SDK for SAP ABAP API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SendMessageBatch` with an AWS SDK or CLI

The following code examples show how to use `SendMessageBatch`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Send and receive batches of messages

CLI

AWS CLI

To send multiple messages as a batch

This example sends 2 messages with the specified message bodies, delay periods, and message attributes, to the specified queue.

Command:

```
aws sqs send-message-batch --queue-url https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue --entries file://send-message-batch.json
```

Input file (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"
      }
        }
    }
]
```

Output:

```
{
  "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"
    }
  ]
}
```

- For API details, see SendMessageBatch in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
            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);
```

- For API details, see SendMessageBatch in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example sends 2 messages with the specified attributes and message bodies to the specified queue. Delivery is delayed for 15 seconds for the first message and 10 seconds for the second message.

```
$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
```

Output:

```
Failed    Successful

------    ----------

{}        {FirstMessage, SecondMessage}
```

- For API details, see SendMessageBatch in *AWS Tools for PowerShell Cmdlet Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see SendMessageBatch in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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
```

- For API details, see SendMessageBatch in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetQueueAttributes` with an AWS SDK or CLI

The following code examples show how to use `SetQueueAttributes`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Publish messages to queues

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Set the policy attribute of a queue for a topic.

```
    /// <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;
    }
```

- For API details, see SetQueueAttributes in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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();
}
```

Configure a dead-letter queue.

```
        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();
}
```

Configure an Amazon SQS queue to use long polling.

```
        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();
}
```

- For API details, see SetQueueAttributes in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To set queue attributes

This example sets the specified queue to a delivery delay of 10 seconds, a maximum message size of 128 KB (128 KB * 1,024 bytes), a message retention period of 3 days (3 days * 24 hours * 60 minutes * 60 seconds), a receive message wait time of 20 seconds, and a default visibility timeout of 60 seconds. This example also associates the specified dead letter queue with a maximum receive count of 1,000 messages.

Command:

```
aws sqs set-queue-attributes --queue-url https://sqs.us-
east-1.amazonaws.com/80398EXAMPLE/MyNewQueue --attributes file://set-queue-
attributes.json
```

Input file (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"
}
```

Output:

```
None.
```

- For API details, see SetQueueAttributes in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// 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(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(context.TODO(),
 &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
```

- For API details, see SetQueueAttributes in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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;
};
```

Configure an Amazon SQS queue to use long polling.

```
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;
};
```

Configure a dead-letter queue.

```
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;
};
```

- For API details, see [SetQueueAttributes](#) in *AWS SDK for JavaScript API Reference*.

PowerShell

Tools for PowerShell

Example 1: This example shows how to set a policy subscribing a queue to an SNS topic. When a message is published to the topic, a message is sent to the subscribed queue.

```
# 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 }
```

Example 2: This example sets the specified attributes for the specified queue.

```
Set-SQSQueueAttribute -Attribute @{"DelaySeconds" = "10"; "MaximumMessageSize" =
 "131072"} -QueueUrl https://sqs.us-east-1.amazonaws.com/80398EXAMPLE/MyQueue
```

- For API details, see SetQueueAttributes in *AWS Tools for PowerShell Cmdlet Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon SQS using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon SQS with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon SQS. Each scenario includes a link to GitHub, where you can find instructions on how to set up and run the code.

Examples

- Create and publish to a FIFO Amazon SNS topic using an AWS SDK
- Receive and process Amazon S3 event notifications by using an AWS SDK.
- Publish Amazon SNS messages to Amazon SQS queues using an AWS SDK
- Send and receive batches of messages with Amazon SQS using an AWS SDK

Create and publish to a FIFO Amazon SNS topic using an AWS SDK

The following code examples show how to create and publish to a FIFO Amazon SNS topic.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example

- creates an Amazon SNS FIFO topic, two Amazon SQS FIFO queues, and one Standard queue.
- subscribes the queues to the topic and publishes a message to the topic.

The test verifies the receipt of the message to each queue. The complete example also shows the addition of access policies and deletes the resources at the end.

```
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");

            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);
        }
    }
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateTopic
  - Publish
  - Subscribe

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create an Amazon SNS FIFO topic, subscribe Amazon SQS FIFO and standard queues to the topic, and publish a message to the topic.

```
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),
                },
            )
            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
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateTopic
  - Publish
  - Subscribe

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a FIFO topic, subscribe an Amazon SQS FIFO queue to the topic, and publish a message to an Amazon SNS topic.

```
    " 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.
```

- For API details, see the following topics in *AWS SDK for SAP ABAP API reference*.
  - CreateTopic
  - Publish
  - Subscribe

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Receive and process Amazon S3 event notifications by using an AWS SDK.

The following code example shows how to work with S3 event notifications in an object-oriented way.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example show how to process S3 notification event by using Amazon SQS.

```
    /**
     * 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);
        }
    }
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - DeleteMessageBatch
  - GetQueueAttributes
  - PutBucketNotificationConfiguration
  - ReceiveMessage

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Publish Amazon SNS messages to Amazon SQS queues using an AWS SDK

The following code examples show how to:

- Create topic (FIFO or non-FIFO).
- Subscribe several queues to the topic with an option to apply a filter.
- Publish messages to the topic.
- Poll the queues for messages received.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
/// <summary>
/// Console application to run a workflow 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 workflow 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 workflow.
    /// </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 workflow, 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;
    }
}
```

Create a class that wraps Amazon SQS operations.

```
/// <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;
```

```
    }
}
```

Create a class that wraps Amazon SNS operations.

```
/// <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;
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.

- CreateQueue
- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        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();
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
  - [CreateQueue]
  - [CreateTopic]

- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
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() (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(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(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(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(
 queueName string, queueUrl string, topicName string, topicArn string, ordinal
 string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)

 err = runner.sqsActor.AttachSendMessagePolicy(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(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(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(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(queueUrls []string) {
  log.Println("Polling queues for messages...")
  for _, queueUrl := range queueUrls {
   var messages []types.Message
   for {
    currentMsgs, err := runner.sqsActor.GetMessages(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(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(
 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()
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this workflow, 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()
 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(ordinal, isFifoTopic)
  resources.queueUrls = append(resources.queueUrls, queueUrl)

  _, filtering := runner.SubscribeQueueToTopic(queueName, queueUrl, topicName,
 topicArn, ordinal, isFifoTopic)
  usingFilters = usingFilters || filtering
 }

 log.Println(strings.Repeat("-", 88))
 runner.PublishMessages(topicArn, isFifoTopic, contentBasedDeduplication,
 usingFilters)
 log.Println(strings.Repeat("-", 88))
 runner.PollForMessages(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()
 }

 log.Println(strings.Repeat("-", 88))
```

```
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define a struct that wraps Amazon SNS actions used in this example.

```
// 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(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(context.TODO(), &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(topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(context.TODO(), &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(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(context.TODO(), &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(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(context.TODO(), &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn,
 err)
 }
 return err
}
```

Define a struct that wraps Amazon SQS actions used in this example.

```
// 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(queueName string, isFifoQueue bool) (string,
 error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(context.TODO(), &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(queueUrl string) (string, error) {
 var queueArn string
```

```
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(context.TODO(),
 &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(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(context.TODO(),
 &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(queueUrl string, maxMessages int32, waitTime
 int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(context.TODO(),
 &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(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(context.TODO(),
 &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(queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(context.TODO(), &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
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This Java example performs these tasks:
 *
 * 1. 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 = "814548047983";
        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 workflow, 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 = "{\n" +
                "     \"Statement\": [\n" +
                "     {\n" +
                "         \"Effect\": \"Allow\",\n" +
                "                 \"Principal\": {\n" +
                "             \"Service\": \"sns.amazonaws.com\"\n" +
                "         },\n" +
                "         \"Action\": \"sqs:SendMessage\",\n" +
                "                 \"Resource\": \"arn:aws:sqs:us-east-1:" +
 accountId + ":" + sqsQueueName + "\",\n" +
                "                 \"Condition\": {\n" +
                "             \"ArnEquals\": {\n" +
                "                 \"aws:SourceArn\": \"arn:aws:sns:us-east-1:" +
 accountId + ":" + topicName + "\"\n" +
                "             }\n" +
                "         }\n" +
                "     }\n" +
                "     ]\n" +
                " }";

        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 "";
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue

- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This is the entry point for this workflow.

```
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";
import { SlowLogger } from "@aws-doc-sdk-examples/lib/slow-logger.js";

export const startSnsWorkflow = () => {
  const noLoggerDelay = process.argv.find((arg) => arg === "--no-logger-delay");
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = noLoggerDelay ? console : new SlowLogger(25);

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

The preceding code provides the necessary dependencies and starts the workflow. The next section contains the bulk of the example.

```
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.
    let 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, deduplicationId, 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();
    }
  }
}
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
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 workflow, 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
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.

- CreateQueue
- CreateTopic
- DeleteMessageBatch
- DeleteQueue
- DeleteTopic
- GetQueueAttributes
- Publish
- ReceiveMessage
- SetQueueAttributes
- Subscribe
- Unsubscribe

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Send and receive batches of messages with Amazon SQS using an AWS SDK

The following code example shows how to:

- Create an Amazon SQS queue.
- Send batches of messages to the queue.
- Receive batches of messages from the queue.
- Delete batches of messages from the queue.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create functions to wrap Amazon SQS message functions.

```
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
```

Use the wrapper functions to send and receive messages in batches.

```
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)
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - CreateQueue
  - DeleteMessageBatch
  - DeleteQueue
  - ReceiveMessage
  - SendMessageBatch

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Serverless examples for Amazon SQS using AWS SDKs

The following code examples show how to use Amazon SQS with AWS SDKs.

Examples

- Invoke a Lambda function from an Amazon SQS trigger
- Reporting batch item failures for Lambda functions with an Amazon SQS trigger

Invoke a Lambda function from an Amazon SQS trigger

The following code examples show how to implement a Lambda function that receives an event triggered by receiving messages from an SQS queue. The function retrieves the messages from the event parameter and logs the content of each message.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using .NET.

```
// 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using Go.

```
// 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming an SQS event with Lambda using Java.

```
// 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](https://github.com/aws-samples/serverless-snippets) repository.

Consuming an SQS event with Lambda using JavaScript.

```
// 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;
  }
}
```

Consuming an SQS event with Lambda using TypeScript.

```
// 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming an SQS event with Lambda using PHP.

```
// 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](#) repository.

Consuming an SQS event with Lambda using Python.

```
# 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using Ruby.

```
# 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using Rust.

```
// 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
}
```

For a complete list of AWS SDK developer guides and code examples, see [Using Amazon SQS with an AWS SDK](). This topic also includes information about getting started and details about previous SDK versions.

Reporting batch item failures for Lambda functions with an Amazon SQS trigger

The following code examples show how to implement partial batch response for Lambda functions that receive events from an SQS queue. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

.NET

AWS SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples]() repository.

Reporting SQS batch item failures with Lambda using .NET.

```
// 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting SQS batch item failures with Lambda using Go.

```
// 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](https://github.com/aws-samples/serverless-snippets) repository.

Reporting SQS batch item failures with Lambda using Java.

```
// 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
                 messageId = message.getMessageId();
             } catch (Exception e) {
                 //Add failed message identifier to the batchItemFailures list
                 batchItemFailures.add(new
 SQSBatchResponse.BatchItemFailure(messageId));
```

```
            }
        }
        return new SQSBatchResponse(batchItemFailures);
    }
}
```

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](https://github.com/aws-samples/serverless-snippets) repository.

Reporting SQS batch item failures with Lambda using JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
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}`);
}
```

Reporting SQS batch item failures with Lambda using TypeScript.

```
// 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting SQS batch item failures with Lambda using PHP.

```
// 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

There's more on GitHub. Find the complete example and learn how to set up and run in the [Serverless examples](https://github.com/aws-samples/serverless-snippets) repository.

Reporting SQS batch item failures with Lambda using Python.

```
# 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting SQS batch item failures with Lambda using Ruby.

```
# 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**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting SQS batch item failures with Lambda using Rust.

```
// 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
}
```

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Cross-service examples for Amazon SQS using AWS SDKs

The following sample applications use AWS SDKs to combine Amazon SQS with other AWS services. Each example includes a link to GitHub, where you can find instructions on how to set up and run the application.

Examples

- Create a web application that sends and retrieves messages by using Amazon SQS
- Create a messenger application with Step Functions
- Create an Amazon Textract explorer application
- Detect people and objects in a video with Amazon Rekognition using an AWS SDK
- Use the AWS Message Processing Framework for .NET to publish and receive Amazon SQS messages

Create a web application that sends and retrieves messages by using Amazon SQS

The following code examples show how to create a messaging application by using Amazon SQS.

Java

SDK for Java 2.x

Shows how to use the Amazon SQS API to develop a Spring REST API that sends and retrieves messages.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Comprehend
- Amazon SQS

Kotlin

SDK for Kotlin

Shows how to use the Amazon SQS API to develop a Spring REST API that sends and retrieves messages.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Comprehend
- Amazon SQS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create a messenger application with Step Functions

The following code example shows how to create an AWS Step Functions messenger application that retrieves message records from a database table.

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) with AWS Step Functions to create a messenger application that retrieves message records from an Amazon DynamoDB table and sends them with Amazon Simple Queue Service (Amazon SQS). The state machine integrates with an AWS Lambda function to scan the database for unsent messages.

- Create a state machine that retrieves and updates message records from an Amazon DynamoDB table.

- Update the state machine definition to also send messages to Amazon Simple Queue Service (Amazon SQS).
- Start and stop state machine runs.
- Connect to Lambda, DynamoDB, and Amazon SQS from a state machine by using service integrations.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Create an Amazon Textract explorer application

The following code examples show how to explore Amazon Textract output through an interactive application.

JavaScript

SDK for JavaScript (v3)

Shows how to use the AWS SDK for JavaScript to build a React application that uses Amazon Textract to extract data from a document image and display it in an interactive web page. This example runs in a web browser and requires an authenticated Amazon Cognito identity for credentials. It uses Amazon Simple Storage Service (Amazon S3) for storage, and for notifications it polls an Amazon Simple Queue Service (Amazon SQS) queue that is subscribed to an Amazon Simple Notification Service (Amazon SNS) topic.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) with Amazon Textract to detect text, form, and table elements in a document image. The input image and Amazon Textract output are shown in a Tkinter application that lets you explore the detected elements.

- Submit a document image to Amazon Textract and explore the output of detected elements.
- Submit images directly to Amazon Textract or through an Amazon Simple Storage Service (Amazon S3) bucket.
- Use asynchronous APIs to start a job that publishes a notification to an Amazon Simple Notification Service (Amazon SNS) topic when the job completes.
- Poll an Amazon Simple Queue Service (Amazon SQS) queue for a job completion message and display the results.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Detect people and objects in a video with Amazon Rekognition using an AWS SDK

The following code examples show how to detect people and objects in a video with Amazon Rekognition.

Python

SDK for Python (Boto3)

Use Amazon Rekognition to detect faces, objects, and people in videos by starting asynchronous detection jobs. This example also configures Amazon Rekognition to notify an Amazon Simple Notification Service (Amazon SNS) topic when jobs complete and subscribes an Amazon Simple Queue Service (Amazon SQS) queue to the topic. When the queue receives a message about a job, the job is retrieved and the results are output.

This example is best viewed on GitHub. For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Rekognition
- Amazon SNS
- Amazon SQS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use the AWS Message Processing Framework for .NET to publish and receive Amazon SQS messages

The following code example shows how to create applications that publish and receive Amazon SQS messages using the AWS Message Processing Framework for .NET.

.NET

AWS SDK for .NET

Provides a tutorial for the AWS Message Processing Framework for .NET. The tutorial creates a web application that allows the user to publish an Amazon SQS message and a command-line application that receives the message.

For complete source code and instructions on how to set up and run, see the full tutorial in the AWS SDK for .NET Developer Guide and the example on GitHub.

Services used in this example

- Amazon SQS

For a complete list of AWS SDK developer guides and code examples, see Using Amazon SQS with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Troubleshooting issues in Amazon SQS

The following topics provide troubleshooting advice for common errors and issues that you might encounter when using the Amazon SQS console, Amazon SQS API, or other tools with Amazon SQS. If you find an issue that is not listed here, you can use the **Feedback** button on this page to report it.

For more troubleshooting advice and answers to common support questions, visit the AWS Knowledge Center.

Topics

- Troubleshoot an access denied error in Amazon SQS
- Troubleshoot Amazon SQS API errors
- Troubleshoot Amazon SQS dead-letter queue and DLQ redrive issues
- Troubleshoot FIFO throttling issues in Amazon SQS
- Troubleshoot messages not returned for an Amazon SQS ReceiveMessage API call
- Troubleshoot Amazon SQS network errors
- Troubleshooting Amazon Simple Queue Service queues using AWS X-Ray

Troubleshoot an access denied error in Amazon SQS

The following topics cover the most common causes of `AccessDenied` or `AccessDeniedException` errors on Amazon SQS API calls. For more information on how to troubleshoot these errors, see How do I troubleshoot "AccessDenied" or "AccessDeniedException" errors on Amazon SQS API calls? in the *AWS Knowledge Center Guide*.

Error message examples:

```
An error occurred (AccessDenied) when calling the SendMessage operation: Access to
        the resource https://sqs.us-east-1.amazonaws.com/ is denied.
```

- or -

```
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.
```

Topics

Amazon SQS queue policy and IAM policy

To verify if the requester has proper permissions to perform an Amazon SQS operation, do the following:

- Identify the IAM principal that's making the Amazon SQS API call. If the IAM principal is from the same account, then either the Amazon SQS queue policy or the AWS Identity and Access Management (IAM) policy must include permissions to explicitly allow access for the action.
- If the principal is an IAM entity:
  - You can identify your IAM user or role by checking the upper-right corner of the AWS Management Console, or by using the `aws sts get-caller-identity` command.
  - Check the IAM policies that are related to the IAM user or role. You can use one of the following methods:
    - Test IAM policies with the IAM Policy Simulator.
    - Review the different IAM policy types.
  - If needed, edit your IAM user policy.
  - Check the queue policy and edit if required.
- If the principal is an AWS service, then the Amazon SQS queue policy must explicitly allow access.
- If the principal is a cross-account principal, then both the Amazon SQS queue policy and the IAM policy must explicitly allow access.
- If the policy uses a condition element, then check that the condition restricts access.

> **Important**
>
> An explicit deny in either policy overrides an explicit allow. Here are some basic examples of Amazon SQS policies.

AWS Key Management Service permissions

If your Amazon SQS queue has server-side encryption (SSE) turned on with a customer managed AWS KMS key, then permissions must be granted to both producers and consumers. To confirm if a queue is encrypted, you can use the `GetQueueAttributes` API `KmsMasterKeyId` attribute, or from the queue console under **Encryption**.

- Required permissions for producers:

```
{
"Effect": "Allow",
"Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
],
"Resource": "<Key ARN>"
}
```

- Required permissions for consumers:

```
{
"Effect": "Allow",
"Action": [
    "kms:Decrypt"
],
"Resource": "<Key ARN>"
}
```

- Required permissions for cross-account access:

```
{
"Effect": "Allow",
"Action": [
    "kms:DescribeKey",
    "kms:Decrypt",
```

```
        "kms:ReEncrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "<Key ARN>"
    }
```

You can use any of the following to enable encryption for an Amazon SQS queue:

- SSE-Amazon SQS (Encryption key created and managed by the Amazon SQS service.)
- AWS managed default key (alias/aws/sqs)
- Customer managed key

However, if you are using an AWS-managed KMS key, you can't modify the default key policy. Therefore, to provide access to other services and cross-accounts, use customer managed key. Doing this allows you to edit the key policy.

VPC endpoint policy

If you access Amazon SQS through an Amazon Virtual Private Cloud (Amazon VPC) endpoint, the Amazon SQS VPC endpoint policy must allow access. You can create a policy for Amazon VPC endpoints for Amazon SQS, where you can specify the following:

1. The principal that can perform actions.
2. The actions that can be performed.
3. The resources on which actions can be performed.

In the following example, the VPC endpoint policy specifies that the IAM user *MyUser* is allowed to send messages to the Amazon SQS queue *MyQueue*. Other actions, IAM users, and Amazon SQS resources are denied access through the VPC endpoint.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
```

```
    }]
}
```

Organization service control policy

If your AWS account belongs to an organization, AWS Organizations policies can block you from accessing your Amazon SQS queues. By default, AWS Organizations policies do not block any requests to Amazon SQS. However, make sure that your AWS Organizations policies haven't been configured to block access to Amazon SQS queues. For instructions on how to check your AWS Organizations policies, see Listing all policies in the *AWS Organizations User Guide*.

Troubleshoot Amazon SQS API errors

The following topics cover the most common errors returned when making Amazon SQS API calls, and how to troubleshoot them.

Topics

QueueDoesNotExist error

This error will be returned when the Amazon SQS service can't find the mentioned queue for the Amazon SQS action.

Possible causes and mitigations:

- **Incorrect region:** Review the Amazon SQS client configuration to confirm that you configured the correct Region on the client. When you don't configure a Region on the client, then the SDK or AWS CLI chooses the Region from the configuration file or the environment variable. If the SDK doesn't find a Region in the configuration file, then the SDK sets the Region to us-east-1 by default.
- **Queue might be recently deleted:** If the queue was deleted before the API call was made, then the API call will return this error. Check CloudTrail for any DeleteQueue operations before the time of the error.

- **Permission issues:** If the requesting AWS Identity and Access Management (IAM) user or role doesn't have the required permissions, then you might receive the following error:

  ```
  The specified queue does not exist or you do not have access to it.
  ```

  Check the permissions, and make the API call with correct permissions.

For more details on troubleshooting the `QueueDoesNotExist` error, see How do I troubleshoot the QueueDoesNotExist error when I make API calls to my Amazon SQS queue? in the *AWS Knowledge Center Guide*.

InvalidAttributeValue error

This error will be returned upon updating the Amazon SQS queue resource policy, or properties with an incorrect policy or a principal.

Possible causes and mitigations:

- **Invalid resource policy:** Check that the resource policy has all the required fields. For more information, see IAM JSON policy elements reference and Validating IAM policies. You can also use the IAM policy generator to create and test an Amazon SQS resource policy. Make sure that the policy is in JSON format.
- **Invalid principal:** Ensure that the `Principal` element exists in the resource policy and that the value is valid. If your Amazon SQS resource policy `Principal` element includes an IAM entity, make sure that the entity exists before you use the policy. Amazon SQS validates the resource policy and checks for the IAM entity. If the IAM entity doesn't exist, you will receive an error. To confirm IAM entities, use the `GetRole` and `GetUser` APIs.

For additional information on how to troubleshoot an `InvalidAttributeValue` error, see How do I troubleshoot the QueueDoesNotExist error when I make API calls to my Amazon SQS queue? in the *AWS Knowledge Center Guide*.

ReceiptHandle error

Upon making a `DeleteMessage` API call, the error `ReceiptHandleIsInvalid` or `InvalidParameterValue` might be returned if the receipt handle is incorrect or expired.

- **ReceiptHandleIsInvalid error:** If the receipt handle is incorrect, you'll receive an error similar to this example:

```
An error occurred (ReceiptHandleIsInvalid) when calling the DeleteMessage operation:
 The input receipt handle <YOUR RECEIPT HANDLE> is not a valid receipt handle.
```

- **InvalidParameterValue error:** If the receipt handle is expired, you'll receive an error similar to this example:

```
An error occurred (InvalidParameterValue) when calling the DeleteMessage operation:
 Value <YOUR RECEIPT HANDLE> for parameter ReceiptHandle is invalid. Reason: The
 receipt handle has expired.
```

Possible causes and mitigations:

The receipt handle is created for every received message, and is only valid for the visibility timeout period. When the visibility timeout period expires, the message becomes visible on the queue for consumers. When you receive the message again from the consumer, you receive a new receipt handle. To prevent incorrect or expired receipt handle errors, use the correct receipt handle to delete the message within the Amazon SQS queue visibility timeout period.

For additional information on how to troubleshoot a `ReceiptHandle` error, see How do I troubleshoot "ReceiptHandleIsInvalid" and "InvalidParameterValue" errors when I use the Amazon SQS DeleteMessage API call? in the *AWS Knowledge Center Guide*.

Troubleshoot Amazon SQS dead-letter queue and DLQ redrive issues

The following topics cover the most common causes of Amazon SQS DLQ and DLQ redrive issues, and how to troubleshoot them. For more information, see How do I troubleshoot Amazon SQS DLQ redrive issues? in the *AWS Knowledge Center Guide*.

Topics

DLQ issues

Learn about common DLQ issues and how to solve them.

Topics

Viewing messages using the console might cause messages to be moved to a dead-letter queue

Amazon SQS counts viewing a message in the console against the corresponding queue's redrive policy. Therefore, if you view a message in the console the number of times specified in the corresponding queue's redrive policy, the message is moved to the corresponding queue's dead-letter queue.

To adjust this behavior, you can do one of the following:

- Increase the **Maximum Receives** setting for the corresponding queue's redrive policy.
- Avoid viewing the corresponding queue's messages in the console.

The `NumberOfMessagesSent` and `NumberOfMessagesReceived` for a dead-letter queue don't match

If you send a message to a dead-letter queue manually, it is captured by the `NumberOfMessagesSent` metric. However, if a message is sent to a dead-letter queue as a result of a failed processing attempt, it isn't captured by this metric. Therefore, it's possible for the values of `NumberOfMessagesSent` and `NumberOfMessagesReceived` to be different.

Creating and configuring a dead-letter queue redrive

Dead-letter queue redrive requires you to set appropriate permissions for Amazon SQS to receive messages from the dead-letter queue, and send messages to the destination queue. If you don't have the correct permissions, the dead-letter queue redrive task can fail. You can view the status of your message redrive task to remediate the issues, and try again.

Standard and FIFO queue message failure handling

Standard queues keep processing messages until the expiration of the retention period. This continuous processing minimizes chances of the queue being blocked by unconsumed messages. Having a large number of messages that the consumer repeatedly fails to delete can increase costs, and place extra load on the hardware. To keep costs down, move failed messages to the dead-letter queue.

Standard queues also allow a high number of in-flight messages. If the majority of your messages can't be consumed, and aren't sent to a dead-letter queue, your rate of processing messages can slow down. To maintain the efficiency of your queue, make sure that your application correctly handles message processing.

FIFO queues provide exactly-once processing by consuming messages in sequence from a message group. Therefore, although the consumer can continue to retrieve ordered messages from another message group, the first message group remains unavailable until the message blocking the queue is processed successfully or moved to a dead-letter queue.

Additionally, FIFO queues allow a lower number of in-flight messages. To keep your FIFO queue from getting blocked by a message, make sure that your application correctly handles message processing.

For more information, see Amazon SQS message quotas and Amazon SQS best practices.

DLQ-redrive issues

Learn about common DLQ-redrive issues and how to solve them.

Topics

AccessDenied permission issue

The `AccessDenied` error occurs when the DLQ redrive fails because the AWS Identity and Access Management (IAM) entity doesn't have the required permissions.

Example error message:

```
Failed to create redrive task. Error code: AccessDenied - Queue Permissions to Redrive.
```

The following API permissions are required to make DLQ redrive requests:

To start a message redrive:

- Dead-letter queue permissions:
  - `sqs:StartMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt` – When either the dead-letter queue or the original source queue are encrypted.
- Destination queue permissions:
  - `sqs:SendMessage`
  - `kms:GenerateDataKey` – When the destination queue is encrypted.
  - `kms:Decrypt` – When the destination queue is encrypted.

To cancel an in-progress message redrive:

- Dead-letter queue permissions:
  - `sqs:CancelMessageMoveTask`
  - `sqs:ReceiveMessage`
  - `sqs:DeleteMessage`
  - `sqs:GetQueueAttributes`
  - `kms:Decrypt` – When either the dead-letter queue or the original source queue are encrypted.

To show a message move status:

- Dead-letter queue permissions:
  - `sqs:ListMessageMoveTasks`
  - `sqs:GetQueueAttributes`

NonExistentQueue error

The `NonExistentQueue` error occurs when the Amazon SQS source queue doesn't exist, or was deleted. Check and redrive to an Amazon SQS queue that is present.

Example error message:

```
Failed: AWS.SimpleQueueService.NonExistentQueue
```

CouldNotDetermineMessageSource error

The `CouldNotDetermineMessageSource` error occurs when you attempt to start a DLQ redrive with the following scenarios:

- An Amazon SQS message sent directly to the DLQ with `SendMessage` API.
- A message from the Amazon Simple Notification Service (Amazon SNS) topic or AWS Lambda function with the DLQ configured.

To resolve this error, choose **Redrive to a custom destination** when you start the redrive. Then, enter the Amazon SQS queue ARN to move all messages from the DLQ to the destination queue.

Example error message:

```
Failed: CouldNotDetermineMessageSource
```

Troubleshoot FIFO throttling issues in Amazon SQS

By default, FIFO queues support 300 transactions per second, per API action for `SendMessage`, `ReceiveMessage`, and `DeleteMessage`. Requests over 300 TPS get the `ThrottlingException` error even if messages in the queue are available. To mitigate this, you can use following methods:

- Enabling high throughput for FIFO queues in Amazon SQS.

- Use the Amazon SQS API batch actions `SendMessageBatch`, `DeleteMessageBatch`, and `ChangeMessageVisibilityBatch` to increase the TPS limit of up to 3,000 messages per second per API action, and to reduce cost. For the `ReceiveMessage` API, set the `MaxNumberofMessages` parameter to receive up to ten messages per transaction. For more information, see Amazon SQS batch actions.
- For FIFO queues with high throughput, follow the recommendations to optimize partition utilization. Send messages with the same message group IDs in batches. Delete messages, or change the message visibility timeout values in batches with receipt handles from the same `ReceiveMessage` API requests.
- Increase the number of unique `MessageGroupId` values. This allows for an even distribution across FIFO queue partitions. For more information, see Using the Amazon SQS message group ID.

For more information, see Why doesn't my Amazon SQS FIFO queue return all messages or messages in other message groups? in the *AWS Knowledge Center Guide*.

Troubleshoot messages not returned for an Amazon SQS ReceiveMessage API call

The following topics cover the most common causes why an Amazon SQS message may not be returned to consumers, and how to troubleshoot them. For more information, see Why can't I receive messages from my Amazon SQS queue? in the *AWS Knowledge Center Guide*.

Topics

Empty queue

To determine if a queue is empty, use long polling to call the `ReceiveMessage` API. You can also use the `ApproximateNumberOfMessagesVisible`, `ApproximateNumberOfMessagesNotVisible`, and `ApproximateNumberOfMessagesDelayed` CloudWatch metrics. If all the metric values are set to 0 for several minutes, the queue is considered empty.

In flight limit reached

If you use long polling and if the queue's in flight limit (20000 for FIFO, 120000 for standard by default) is breached, Amazon SQS won't return error messages that exceed quota limits.

Message delay

If the Amazon SQS queue is configured as a delay queue, or the messages were sent with message timers, then the messages aren't visible until the delay time ends. To verify if a queue is configured as a delay queue, use the `GetQueueAttributes` API `DelaySeconds` attribute, or from the queue console under **Delivery delay**. Check the `ApproximateNumberOfMessagesDelayed` CloudWatch metric to understand if any messages are delayed.

Message is in flight

If a different consumer has polled the message, the message will be in flight or invisible for the visibility timeout period. The additional polls might return an empty receive. Check the ApproximateNumberOfMessagesVisible CloudWatch metric to understand the number of messages that are available to be received. In the case of FIFO queues, if a message with the message group ID is in flight, then no more messages will be returned unless you delete the message, or it becomes visible. This is because message ordering is maintained at the message group level in a FIFO queue.

Polling method

If you are using short polling, (WaitTimeSeconds is 0) Amazon SQS samples a subset of its servers, and returns messages from only those servers. Therefore, you might not get the messages even if they are available for to be received. Subsequent poll requests will return the messages.

If you are using long polling, Amazon SQS polls all the servers and sends a response after collecting at least one available message, and up to the maximum number that's specified. If the value for ReceiveMessage WaitTimeSeconds is too low, you might not receive all the available messages.

Troubleshoot Amazon SQS network errors

The following topics cover the most common causes for network issues in Amazon SQS, and how to troubleshoot them.

Topics

ETIMEOUT error

The ETIMEOUT error occurs when the client can't establish a TCP connection to an Amazon SQS endpoint.

Troubleshooting:

- **Check the network connection**

  Test your network connection to Amazon SQS by running commands like `telnet`.

  Example: telnet sqs.us-east-1.amazonaws.com 443
- **Check network settings**
  - Make sure that your local firewall rules, routes, and access control lists (ACLs) allow traffic on the port that you use.
  - The security group outbound (egress) rules must allow traffic to the port 80 or 443.
  - The network ACL outbound (egress) rules must allow traffic to TCP port 80 or 443.
  - The network ACL inbound (ingress) rules must allow traffic on TCP ports 1024-65535.
  - Amazon Elastic Compute Cloud (Amazon EC2) instances that connect to the public internet must have internet connectivity.
- **Amazon Virtual Private Cloud (Amazon VPC) endpoints**

  If you access Amazon SQS through an Amazon VPC endpoint, then the endpoints security group must allow inbound traffic to the clients security group on port 443. The network ACL associated with the subnet of the VPC endpoint must have this configuration:

- The network ACL outbound (egress) rules must allow traffic on TCP ports 1024-65535 (ephemeral ports).
- The network ACL inbound (ingress) rules must allow traffic on port 443.

Also, the Amazon SQS VPC endpoint AWS Identity and Access Management (IAM) policy must allow access. The following example VPC endpoint policy specifies that the IAM user *MyUser* is allowed to send messages to the Amazon SQS queue *MyQueue*. Other actions, IAM users, and Amazon SQS resources are denied access through the VPC endpoint.

```
{
    "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

The UnknownHostException error occurs when the host IP address couldn't be determined.

Troubleshooting:

Use the nslookup utility to return the IP address associated with the host name:

- Windows and Linux OS

```
nslookup sqs.<region>.amazonaws.com
```

- AWS CLI or SDK for Python legacy endpoints:

```
nslookup <region>.queue.amazonaws.com
```

If you received an unsuccessful output, follow the instructions in How does DNS work and how do I troubleshoot partial or intermittent DNS failures? in the *AWS Knowledge Center Guide*.

If you received a valid output, then it is likely to be an application-level issue. To resolve application-level issues, try the following methods:

- Restart your application.
- Confirm that your Java application doesn't have a bad DNS cache. If possible, configure your application to adhere to the DNS TTL. For more information, see Setting the JVM TTL for DNS name lookups.

For additional information on how to troubleshoot network errors, see How do I troubleshoot Amazon SQS “ETIMEOUT” and “UnknownHostException” connection errors? in the *AWS Knowledge Center Guide*.

Troubleshooting Amazon Simple Queue Service queues using AWS X-Ray

AWS X-Ray collects data about requests that your application serves and lets you view and filter data to identify potential issues and opportunities for optimization. For any traced request to your application, you can see detailed information about the request, the response, and the calls that your application makes to downstream AWS resources, microservices, databases and HTTP web APIs.

To send AWS X-Ray trace headers through Amazon SQS, you can do one of the following:

- Use the `X-Amzn-Trace-Id` tracing header.
- Use the `AWSTraceHeader` message system attribute.

To collect data on errors and latency, you must instrument the `AmazonSQS` client using the AWS X-Ray SDK.

You can use the AWS X-Ray console to view the map of connections between Amazon SQS and other services that your application uses. You can also use the console to view metrics such as average latency and failure rates. For more information, see Amazon SQS and AWS X-Ray in the *AWS X-Ray Developer Guide*.

Security in Amazon SQS

This section provides information about Amazon SQS security, authentication and access control, and the Amazon SQS Access Policy Language.

Topics

Data protection in Amazon SQS

The AWS shared responsibility model applies to data protection in Amazon Simple Queue Service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.

- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon SQS or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

The following sections provide information about data protection in Amazon SQS.

Topics

- Data encryption in Amazon SQS
- Internetwork traffic privacy in Amazon SQS

Data encryption in Amazon SQS

Data protection refers to protecting data while in-transit (as it travels to and from Amazon SQS) and at rest (while it is stored on disks in Amazon SQS data centers). You can protect data in transit using Secure Sockets Layer (SSL) or client-side encryption. By default, Amazon SQS stores messages and files using disk encryption. You can protect data at rest by requesting Amazon SQS to encrypt your messages before saving them to the encrypted file system in its data centers. Amazon SQS recommends using SSE for optimized data encryption.

Topics

- Encryption at rest in Amazon SQS
- Amazon SQS Key management

Encryption at rest in Amazon SQS

Server-side encryption (SSE) lets you transmit sensitive data in encrypted queues. SSE protects the contents of messages in queues using SQS-managed encryption keys (SSE-SQS) or keys managed in the AWS Key Management Service (SSE-KMS). For information about managing SSE using the AWS Management Console, see the following:

- Configuring SSE-SQS for a queue (console)
- Configuring SSE-KMS for a queue (console)

For information about managing SSE using the AWS SDK for Java (and the `CreateQueue`, `SetQueueAttributes`, and `GetQueueAttributes` actions), see the following examples:

- Using server-side encryption with Amazon SQS queues
- Configuring KMS permissions for AWS services

SSE encrypts messages as soon as Amazon SQS receives them. The messages are stored in encrypted form and Amazon SQS decrypts messages only when they are sent to an authorized consumer.

> **Important**
>
> All requests to queues with SSE enabled must use HTTPS and Signature Version 4.
> An encrypted queue that uses the default key (AWS managed KMS key for Amazon SQS) cannot invoke a Lambda function in a different AWS account.
> Some features of AWS services that can send notifications to Amazon SQS using the AWS Security Token Service `AssumeRole` action are compatible with SSE but work *only with standard queues:*
>
> - Auto Scaling Lifecycle Hooks
> - AWS Lambda Dead-Letter Queues
>
> For information about compatibility of other services with encrypted queues, see Configure KMS permissions for AWS services and your service documentation.

AWS KMS combines secure, highly available hardware and software to provide a key management system scaled for the cloud. When you use Amazon SQS with AWS KMS, the data keys that encrypt your message data are also encrypted and stored with the data they protect.

The following are benefits of using AWS KMS:

- You can create and manage AWS KMS keys yourself.
- You can also use the AWS managed KMS key for Amazon SQS, which is unique for each account and region.
- The AWS KMS security standards can help you meet encryption-related compliance requirements.

For more information, see What is AWS Key Management Service? in the *AWS Key Management Service Developer Guide*.

Topics

Encryption scope

SSE encrypts the body of a message in an Amazon SQS queue.

SSE doesn't encrypt the following:

- Queue metadata (queue name and attributes)
- Message metadata (message ID, timestamp, and attributes)
- Per-queue metrics

Encrypting a message makes its contents unavailable to unauthorized or anonymous users. With SSE enabled, anonymous `SendMessage` and `ReceiveMessage` requests to the encrypted queue will be rejected. Amazon SQS security best practices recommends against using anonymous requests. If you wish to send anonymous requests to an Amazon SQS queue, make sure you disable SSE. This doesn't affect the normal functioning of Amazon SQS:

- A message is encrypted only if it is sent after the encryption of a queue is enabled. Amazon SQS doesn't encrypt backlogged messages.
- Any encrypted message remains encrypted even if the encryption of its queue is disabled.

Moving a message to a dead-letter queue doesn't affect its encryption:

- When Amazon SQS moves a message from an encrypted source queue to an unencrypted dead-letter queue, the message remains encrypted.
- When Amazon SQS moves a message from an unencrypted source queue to an encrypted dead-letter queue, the message remains unencrypted.

Key terms

The following key terms can help you better understand the functionality of SSE. For detailed descriptions, see the *Amazon Simple Queue Service API Reference*.

Data key

The key (DEK) responsible for encrypting the contents of Amazon SQS messages.

For more information, see Data Keys in the *AWS Key Management Service Developer Guide* in the *AWS Encryption SDK Developer Guide*.

Data key reuse period

The length of time, in seconds, for which Amazon SQS can reuse a data key to encrypt or decrypt messages before calling AWS KMS again. An integer representing seconds, between 60 seconds (1 minute) and 86,400 seconds (24 hours). The default is 300 (5 minutes). For more information, see Understanding the data key reuse period.

> **Note**
>
> In the unlikely event of being unable to reach AWS KMS, Amazon SQS continues to use the cached data key until a connection is reestablished.

KMS key ID

The alias, alias ARN, key ID, or key ARN of an AWS managed KMS key or a custom KMS key —in your account or in another account. While the alias of the AWS managed KMS key for Amazon SQS is always `alias/aws/sqs`, the alias of a custom KMS key can, for example, be `alias/MyAlias`. You can use these KMS keys to protect the messages in Amazon SQS queues.

> **Note**
>
> Keep the following in mind:

- If you don't specify a custom KMS key, Amazon SQS uses the AWS managed KMS key for Amazon SQS.
- The first time you use the AWS Management Console to specify the AWS managed KMS key for Amazon SQS for a queue, AWS KMS creates the AWS managed KMS key for Amazon SQS.
- Alternatively, the first time you use the `SendMessage` or `SendMessageBatch` action on a queue with SSE enabled, AWS KMS creates the AWS managed KMS key for Amazon SQS.

You can create KMS keys, define the policies that control how KMS keys can be used, and audit KMS key usage using the **Customer managed keys** section of the AWS KMS console or the `CreateKey` AWS KMS action. For more information, see KMS keys and Creating Keys in the *AWS Key Management Service Developer Guide*. For more examples of KMS key identifiers, see KeyId in the *AWS Key Management Service API Reference*. For information about finding KMS key identifiers, see Find the Key ID and ARN in the *AWS Key Management Service Developer Guide*.

Important

There are additional charges for using AWS KMS. For more information, see Estimating AWS KMS costs and AWS Key Management Service Pricing.

Envelope Encryption

The security of your encrypted data depends in part on protecting the data key that can decrypt it. Amazon SQS uses the KMS key to encrypt the data key and then the encrypted data key is stored with the encrypted message. This practice of using a KMS key to encrypt data keys is known as envelope encryption.

For more information, see Envelope Encryption in the *AWS Encryption SDK Developer Guide*.

Amazon SQS Key management

Amazon SQS integrates with the AWS Key Management Service (KMS) to manage KMS keys for server-side encryption (SSE). See Encryption at rest in Amazon SQS for SSE information and key management definitions. Amazon SQS uses KMS keys to validate and secure the data keys that encrypt and decrypt the messages. The following sections provide information about working with KMS keys and data keys in the Amazon SQS service.

Topics

Configuring AWS KMS permissions

Every KMS key must have a key policy. Note that you cannot modify the key policy of an AWS managed KMS key for Amazon SQS. The policy for this KMS key includes permissions for all principals in the account (that are authorized to use Amazon SQS) to use encrypted queues.

For a customer managed KMS key, you must configure the key policy to add permissions for each queue producer and consumer. To do this, you name the producer and consumer as users in the KMS key policy. For more information about AWS KMS permissions, see AWS KMS resources and operations or AWS KMS API permissions reference in the *AWS Key Management Service Developer Guide*.

Alternatively, you can specify the required permissions in an IAM policy assigned to the principals that produce and consume encrypted messages. For more information, see Using IAM Policies with AWS KMS in the *AWS Key Management Service Developer Guide*.

> **Note**
>
> While you can configure global permissions to send to and receive from Amazon SQS, AWS KMS requires explicitly naming the full ARN of KMS keys in specific regions in the `Resource` section of an IAM policy.

Configure KMS permissions for AWS services

Several AWS services act as event sources that can send events to Amazon SQS queues. To allow these event sources to work with encrypted queues, you must create a customer managed KMS key and add permissions in the key policy for the service to use the required AWS KMS API methods. Perform the following steps to configure the permissions.

> **Warning**
>
> When changing the KMS key for encrypting your Amazon SQS messages, be aware that existing messages encrypted with the old KMS key will remain encrypted with that key. To decrypt these messages, you must retain the old KMS key and ensure that its key policy grants Amazon SQS the permissions for `kms:Decrypt` and `kms:GenerateDataKey`. After updating to a new KMS key for encrypting new messages, ensure all existing messages encrypted with the old KMS key are processed and removed from the queue before deleting or disabling the old KMS key.

1. Create a customer managed KMS key. For more information, see Creating Keys in the *AWS Key Management Service Developer Guide*.
2. To allow the AWS service event source to use the `kms:GenerateDataKey` and `kms:Decrypt` API methods, add the following statement to the KMS key policy.

```
{
   "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Principal": {
            "Service": "service.amazonaws.com"
         },
         "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
         ],
         "Resource": "*"
       }]
}
```

Replace "service" in the above example with the *Service name* of the event source. Event sources include the following services.

Event source	Service name
Amazon CloudWatch Events	`events.amazonaws.com`

Event source	Service name
Amazon S3 event notifications	`s3.amazonaws.com`
Amazon SNS topic subscriptions	`sns.amazonaws.com`

3. Configure an existing SSE queue using the ARN of your KMS key.
4. Provide the ARN of the encrypted queue to the event source.

Configure AWS KMS permissions for producers

When the data key reuse period expires, the producer's next call to `SendMessage` or `SendMessageBatch` also triggers calls to `kms:GenerateDataKey` and `kms:Decrypt`. The call to `kms:Decrypt` is to verify the integrity of the new data key before using it. Therefore, the producer must have the `kms:GenerateDataKey` and `kms:Decrypt` permissions for the KMS key.

Add the following statement to the IAM policy of the producer. Remember to use the correct ARN values for the key resource and the queue resource.

```
{
   "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Action": [
            "kms:GenerateDataKey",
            "kms:Decrypt"
         ],
         "Resource":  "arn:aws:kms:us-
east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
         }, {
         "Effect": "Allow",
         "Action": [
            "sqs:SendMessage"
         ],
         "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
      }]
}
```

Configure AWS KMS permissions for consumers

When the data key reuse period expires, the consumer's next call to `ReceiveMessage` also triggers a call to `kms:Decrypt`, to verify the integrity of the new data key before using it. Therefore, the consumer must have the `kms:Decrypt` permission for any KMS key that is used to encrypt the messages in the specified queue. If the queue acts as a dead-letter queue, the consumer must also have the `kms:Decrypt` permission for any KMS key that is used to encrypt the messages in the source queue. Add the following statement to the IAM policy of the consumer. Remember to use the correct ARN values for the key resource and the queue resource.

```
{
   "Version": "2012-10-17",
      "Statement": [{
         "Effect": "Allow",
         "Action": [
            "kms:Decrypt"
         ],
         "Resource": "arn:aws:kms:us-
east-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
         }, {
         "Effect": "Allow",
         "Action": [
            "sqs:ReceiveMessage"
         ],
         "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
      }]
}
```

Configure AWS KMS permissions with confused deputy protection

When the principal in a key policy statement is an AWS service principal, you can use the `aws:SourceArn` or `aws:SourceAccount` global condition keys to protect against the confused deputy scenario. To use these condition keys, set the value to the Amazon Resource Name (ARN) of the resource that is being encrypted. If you don't know the ARN of the resource, use `aws:SourceAccount` instead.

In this KMS key policy, a specific resource from *service* that is owned by account `111122223333` is allowed to call KMS for `Decrypt` and `GenerateDataKey` actions, which occur during SSE usage of Amazon SQS.

```
{
```

```
  "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:service::111122223333:resource"
     ]
    }
   }
  }]
}
```

When using SSE enabled Amazon SQS queues, the following services support `aws:SourceArn`:

- Amazon SNS
- Amazon S3
- CloudWatch Events
- AWS Lambda
- CodeBuild
- Amazon Connect Customer Profiles
- AWS Auto Scaling
- Amazon Chime

Understanding the data key reuse period

The data key reuse period defines the maximum duration for Amazon SQS to reuse the same data key. When the data key reuse period ends, Amazon SQS generates a new data key. Note the following guidelines about the reuse period.

- A shorter reuse period provides better security but results in more calls to AWS KMS, which might incur charges beyond the Free Tier.
- Although the data key is cached separately for encryption and for decryption, the reuse period applies to both copies of the data key.
- When the data key reuse period ends, the next call to `SendMessage` or `SendMessageBatch` typically triggers a call to the AWS KMS `GenerateDataKey` method to get a new data key. Also, the next calls to `SendMessage` and `ReceiveMessage` will each trigger a call to AWS KMS `Decrypt` to verify the integrity of the data key before using it.
- Principals (AWS accounts or users) don't share data keys (messages sent by unique principals always get unique data keys). Therefore, the volume of calls to AWS KMS is a multiple of the number of unique principals in use during the data key reuse period.

Estimating AWS KMS costs

To predict costs and better understand your AWS bill, you might want to know how often Amazon SQS uses your KMS key.

> **Note**
>
> Although the following formula can give you a very good idea of expected costs, actual costs might be higher because of the distributed nature of Amazon SQS.

To calculate the number of API requests (R) *per queue*, use the following formula:

```
R = (B / D) * (2 * P + C)
```

B is the billing period (in seconds).

D is the data key reuse period (in seconds).

P is the number of producing principals that send to the Amazon SQS queue.

C is the number of consuming principals that receive from the Amazon SQS queue.

> **Important**
>
> In general, producing principals incur double the cost of consuming principals. For more information, see Understanding the data key reuse period.

If the producer and consumer have different users, the cost increases.

The following are example calculations. For exact pricing information, see AWS Key Management Service Pricing.

Example 1: Calculating the number of AWS KMS API calls for 2 principals and 1 queue

This example assumes the following:

- The billing period is January 1-31 (2,678,400 seconds).
- The data key reuse period is set to 5 minutes (300 seconds).
- There is 1 queue.
- There is 1 producing principal and 1 consuming principal.

```
(2,678,400 / 300) * (2 * 1 + 1) = 26,784
```

Example 2: Calculating the number of AWS KMS API calls for multiple producers and consumers and 2 queues

This example assumes the following:

- The billing period is February 1-28 (2,419,200 seconds).
- The data key reuse period is set to 24 hours (86,400 seconds).
- There are 2 queues.
- The first queue has 3 producing principals and 1 consuming principal.
- The second queue has 5 producing principals and 2 consuming principals.

```
(2,419,200 / 86,400 * (2 * 3 + 1)) + (2,419,200 / 86,400 * (2 * 5 + 2)) = 532
```

AWS KMS errors

When you work with Amazon SQS and AWS KMS, you might encounter errors. The following references describe the errors and possible troubleshooting solutions.

- Common AWS KMS errors

- AWS KMS Decrypt errors
- AWS KMS GenerateDataKey errors

Internetwork traffic privacy in Amazon SQS

An Amazon Virtual Private Cloud (Amazon VPC) endpoint for Amazon SQS is a logical entity within a VPC that allows connectivity only to Amazon SQS. The VPC routes requests to Amazon SQS and routes responses back to the VPC. The following sections provide information about working with VPC endpoints and creating VPC endpoint policies.

Topics

Amazon Virtual Private Cloud endpoints for Amazon SQS

If you use Amazon VPC to host your AWS resources, you can establish a connection between your VPC and Amazon SQS. You can use this connection to send messages to your Amazon SQS queues without crossing the public internet.

Amazon VPC lets you launch AWS resources in a custom virtual network. You can use a VPC to control your network settings, such as the IP address range, subnets, route tables, and network gateways. For more information about VPCs, see the *Amazon VPC User Guide*.

To connect your VPC to Amazon SQS, you must first define an *interface VPC endpoint*, which lets you connect your VPC to other AWS services. The endpoint provides reliable, scalable connectivity to Amazon SQS without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For more information, see Tutorial: Sending a message to an Amazon SQS queue from Amazon Virtual Private Cloud and Example 5: Deny access if it isn't from a VPC endpoint in this guide and Interface VPC Endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

> **Important**
>
> - You can use Amazon Virtual Private Cloud only with HTTPS Amazon SQS endpoints.
> - When you configure Amazon SQS to send messages from Amazon VPC, you must enable private DNS and specify endpoints in the format `sqs.`*`us-east-2`*`.amazonaws.com`.

- Private DNS doesn't support legacy endpoints such as `queue.amazonaws.com` or *`us-east-2`*`.queue.amazonaws.com`.

Creating an Amazon VPC endpoint policy for Amazon SQS

You can create a policy for Amazon VPC endpoints for Amazon SQS in which you specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling Access to Services with VPC Endpoints in the *Amazon VPC User Guide*

The following example VPC endpoint policy specifies that the user `MyUser` is allowed to send messages to the Amazon SQS queue `MyQueue`.

```
{
   "Statement": [{
      "Action": ["sqs:SendMessage"],
      "Effect": "Allow",
      "Resource": "arn:aws:sqs:us-east-2:123456789012:MyQueue",
      "Principal": {
        "AWS": "arn:aws:iam:123456789012:user/MyUser"
      }
   }]
}
```

The following are denied:

- Other Amazon SQS API actions, such as `sqs:CreateQueue` and `sqs:DeleteQueue`.
- Other users and rules which attempt to use this VPC endpoint.
- `MyUser` sending messages to a different Amazon SQS queue.

> **Note**
>
> The user can still use other Amazon SQS API actions from *outside* the VPC. For more information, see Example 5: Deny access if it isn't from a VPC endpoint.

Identity and access management in Amazon SQS

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon SQS resources. IAM is an AWS service that you can use with no additional charge.

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon SQS.

Service user – If you use the Amazon SQS service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon SQS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon SQS, see Troubleshooting Amazon Simple Queue Service identity and access.

Service administrator – If you're in charge of Amazon SQS resources at your company, you probably have full access to Amazon SQS. It's your job to determine which Amazon SQS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon SQS, see How Amazon Simple Queue Service works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon SQS. To view example Amazon SQS identity-based policies that you can use in IAM, see Policy best practices.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or

AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

  - **Service-linked role –** A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2 –** You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

Overview of managing access in Amazon SQS

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (users, groups, and roles), and some services (such as Amazon SQS) also support attaching permissions policies to resources.

> **Note**
>
> An *account administrator* (or administrator user) is a user with administrative privileges. For more information, see IAM Best Practices in the *IAM User Guide*.

When granting permissions, you specify what users get permissions, the resource they get permissions for, and the specific actions that you want to allow on the resource.

Topics

- Amazon Simple Queue Service resource and operations
- Understanding resource ownership
- Managing access to resources
- Specifying policy elements: Actions, effects, resources, and principals

Amazon Simple Queue Service resource and operations

In Amazon SQS, the only resource is the *queue*. In a policy, use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. The following resource has a unique ARN associated with it:

Resource type	ARN format
Queue	`arn:aws:sqs:` *`region`*`:`*`account_id`* `:`*`queue_name`*

The following are examples of the ARN format for queues:

- An ARN for a queue named `my_queue` in the US East (Ohio) region, belonging to AWS Account 123456789012:

```
arn:aws:sqs:us-east-2:123456789012:my_queue
```

- An ARN for a queue named `my_queue` in each of the different regions that Amazon SQS supports:

```
arn:aws:sqs:*:123456789012:my_queue
```

- An ARN that uses * or ? as a wildcard for the queue name. In the following examples, the ARN matches all queues prefixed with `my_prefix_`:

```
arn:aws:sqs:*:123456789012:my_prefix_*
```

You can get the ARN value for an existing queue by calling the [`GetQueueAttributes`] action. The value of the `QueueArn` attribute is the ARN of the queue. For more information about ARNs, see IAM ARNs in the *IAM User Guide*.

Amazon SQS provides a set of actions that work with the queue resource. For more information, see Amazon SQS API permissions: Actions and resource reference.

Understanding resource ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the *principal entity* (that is, the root account, a user , or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create an Amazon SQS queue, your AWS account is the owner of the resource (in Amazon SQS, the resource is the Amazon SQS queue).
- If you create a user in your AWS account and grant permissions to create a queue to the user, the user can create the queue. However, your AWS account (to which the user belongs) owns the queue resource.
- If you create an IAM role in your AWS account with permissions to create an Amazon SQS queue, anyone who can assume the role can create a queue. Your AWS account (to which the role belongs) owns the queue resource.

Managing access to resources

A *permissions policy* describes the permissions granted to accounts. The following section explains the available options for creating permissions policies.

> **Note**
>
> This section discusses using IAM in the context of Amazon SQS. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see What is IAM? in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as *identity-based* policies (IAM policies) and policies attached to a resource are referred to as *resource-based* policies.

Identity-based policies

There are two ways to give your users permissions to your Amazon SQS queues: using the Amazon SQS policy system and using the IAM policy system. You can use either system, or both, to attach policies to users or roles. In most cases, you can achieve the same result using either system. For example, you can do the following:

- **Attach a permission policy to a user or a group in your account** – To grant user permissions to create an Amazon SQS queue, attach a permissions policy to a user or group that the user belongs to.
- **Attach a permission policy to a user in another AWS account** – To grant user permissions to create an Amazon SQS queue, attach an Amazon SQS permissions policy to a user in another AWS account.

  Cross-account permissions don't apply to the following actions:

  - `AddPermission`
  - `CancelMessageMoveTask`
  - `CreateQueue`
  - `DeleteQueue`
  - `ListMessageMoveTask`
  - `ListQueues`
  - `ListQueueTags`
  - `RemovePermission`
  - `SetQueueAttributes`
  - `StartMessageMoveTask`

  - [TagQueue]
  - [UntagQueue]

- **Attach a permission policy to a role (grant cross-account permissions)** – To grant cross-account permissions, attach an identity-based permissions policy to an IAM role. For example, the AWS account A administrator can create a role to grant cross-account permissions to AWS account B (or an AWS service) as follows:
  - The account A administrator creates an IAM role and attaches a permissions policy — that grants permissions on resources in account A — to the role.
  - The account A administrator attaches a trust policy to the role that identifies account B as the principal who can assume the role.
  - The account B administrator delegates the permission to assume the role to any users in account B. This allows users in account B to create or access queues in account A.

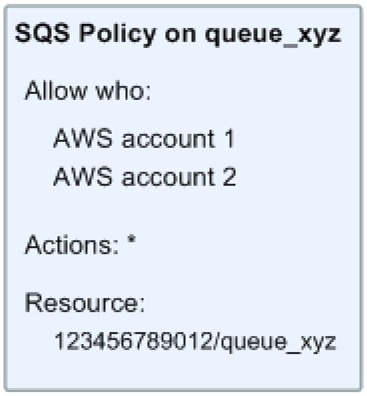
> **Note**
>
> If you want to grant the permission to assume the role to an AWS service, the principal in the trust policy can also be an AWS service principal.

For more information about using IAM to delegate permissions, see [Access Management] in the *IAM User Guide*.

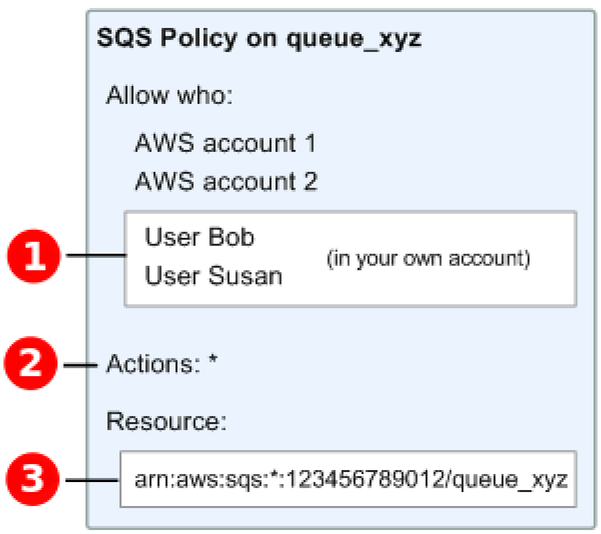
While Amazon SQS works with IAM policies, it has its own policy infrastructure. You can use an Amazon SQS policy with a queue to specify which AWS Accounts have access to the queue. You can specify the type of access and conditions (for example, a condition that grants permissions to use `SendMessage`, `ReceiveMessage` if the request is made before December 31, 2010). The specific actions you can grant permissions for are a subset of the overall list of Amazon SQS actions. When you write an Amazon SQS policy and specify * to "allow all Amazon SQS actions," it means that a user can perform all actions in this subset.

The following diagram illustrates the concept of one of these basic Amazon SQS policies that covers the subset of actions. The policy is for `queue_xyz`, and it gives AWS Account 1 and AWS Account 2 permissions to use any of the allowed actions with the specified queue.

> **Note**
>
> The resource in the policy is specified as `123456789012/queue_xyz`, where `123456789012` is the AWS Account ID of the account that owns the queue.



With the introduction of IAM and the concepts of *Users* and *Amazon Resource Names (ARNs)*, a few things have changed about SQS policies. The following diagram and table describe the changes.



1

For information about giving permissions to users in different accounts, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.

2

The subset of actions included in * has expanded. For a list of allowed actions, see Amazon SQS API permissions: Actions and resource reference.

3

You can specify the resource using the Amazon Resource Name (ARN), the standard means of specifying resources in IAM policies. For information about the ARN format for Amazon SQS queues, see Amazon Simple Queue Service resource and operations.

For example, according to the Amazon SQS policy in the preceding diagram, anyone who possesses the security credentials for AWS Account 1 or AWS Account 2 can access `queue_xyz`. In addition, Users Bob and Susan in your own AWS Account (with ID `123456789012`) can access the queue.

Before the introduction of IAM, Amazon SQS automatically gave the creator of a queue full control over the queue (that is, access to all of the possible Amazon SQS actions on that queue). This is no longer true, unless the creator uses AWS security credentials. Any user who has permissions to create a queue must also have permissions to use other Amazon SQS actions in order to do anything with the created queues.

The following is an example policy that allows a user to use all Amazon SQS actions, but only with queues whose names are prefixed with the literal string `bob_queue_`.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:bob_queue_*"
   }]
}
```

For more information, see Using policies with Amazon SQS, and Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Specifying policy elements: Actions, effects, resources, and principals

For each Amazon Simple Queue Service resource, the service defines a set of actions. To grant permissions for these actions, Amazon SQS defines a set of actions that you can specify in a policy.

> **Note**
>
> Performing an action can require permissions for more than one action. When granting permissions for specific actions, you also identify the resource for which the actions are allowed or denied.

The following are the most basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies.
- **Action** – You use action keywords to identify resource actions that you want to allow or deny. For example, the `sqs:CreateQueue` permission allows the user to perform the Amazon Simple Queue Service `CreateQueue` action.
- **Effect** – You specify the effect when the user requests the specific action—this can be either allow or deny. If you don't explicitly grant access to a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user can't access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only).

To learn more about Amazon SQS policy syntax and descriptions, see AWS IAM Policy Reference in the *IAM User Guide*.

For a table of all Amazon Simple Queue Service actions and the resources that they apply to, see Amazon SQS API permissions: Actions and resource reference.

How Amazon Simple Queue Service works with IAM

Before you use IAM to manage access to Amazon SQS, learn what IAM features are available to use with Amazon SQS.

IAM features you can use with Amazon Simple Queue Service

IAM feature	Amazon SQS support
Identity-based policies	Yes
Resource-based policies	Yes
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Forward access sessions (FAS)	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how Amazon SQS and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Access control

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

> **Note**
>
> It is important to understand that all AWS accounts can delegate their permissions to users under their accounts. Cross-account access allows you to share access to your AWS resources without having to manage additional users. For information about using cross-account access, see Enabling Cross-Account Access in the *IAM User Guide*.
> See Limitations of Amazon SQS custom policies for further details on cross-content permissions and condition keys within Amazon SQS custom policies.

Identity-based policies for Amazon SQS

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon SQS

To view examples of Amazon SQS identity-based policies, see Policy best practices.

Resource-based policies within Amazon SQS

Supports resource-based policies: Yes

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon SQS

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon SQS actions, see Resources Defined by Amazon Simple Queue Service in the *Service Authorization Reference*.

Policy actions in Amazon SQS use the following prefix before the action:

```
sqs
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "sqs:action1",
      "sqs:action2"
```

```
        ]
```

To view examples of Amazon SQS identity-based policies, see [Policy best practices](#).

Policy resources for Amazon SQS

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its [Amazon Resource Name (ARN)](#). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of Amazon SQS resource types and their ARNs, see [Actions Defined by Amazon Simple Queue Service](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Resources Defined by Amazon Simple Queue Service](#).

To view examples of Amazon SQS identity-based policies, see [Policy best practices](#).

Policy condition keys for Amazon SQS

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Amazon SQS condition keys, see Condition Keys for Amazon Simple Queue Service in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Resources Defined by Amazon Simple Queue Service.

To view examples of Amazon SQS identity-based policies, see Policy best practices.

ACLs in Amazon SQS

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon SQS

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Amazon SQS

Supports temporary credentials: Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Forward access sessions for Amazon SQS

Supports forward access sessions (FAS): Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon SQS

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Amazon SQS functionality. Edit service roles only when Amazon SQS provides guidance to do so.

Service-linked roles for Amazon SQS

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Amazon SQS updates to AWS managed policies

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed policy: AmazonSQSFullAccess

You can attach the `AmazonSQSFullAccess` policy to your Amazon SQS identities. This policy grants permissions that allow full access to Amazon SQS.

To view the permissions for this policy, see AmazonSQSFullAccess in the *AWS Managed Policy Reference*.

AWS managed policy: AmazonSQSReadOnlyAccess

You can attach the `AmazonSQSReadOnlyAccess` policy to your Amazon SQS identities. This policy grants permissions that allow read-only access to Amazon SQS.

To view the permissions for this policy, see AmazonSQSReadOnlyAccess in the *AWS Managed Policy Reference*.

Amazon SQS updates to AWS managed policies

View details about updates to AWS managed policies for Amazon SQS since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon SQS Document history page.

Change	Description	Date
AmazonSQSReadOnlyAccess	Amazon SQS added the `ListQueueTags` action, which retrieves all tags	June 20, 2024

Change	Description	Date
	associated with a specified Amazon SQS queue. It allows you to view the key-value pairs that have been assigned to the queue for organizational or metadata purposes. This action is associated with the `ListQueueTags` API operation.	
AmazonSQSReadOnlyAccess	Amazon SQS added a new action that allows you to list the most recent message movement tasks (up to 10) under a specific source queue. This action is associated with the `ListMessageMoveTasks` API operation.	June 9, 2023

Troubleshooting Amazon Simple Queue Service identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon SQS and IAM.

Topics

I am not authorized to perform an action in Amazon SQS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `sqs:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 sqs:GetWidget on resource: my-example-widget
```

In this case, Mateo's policy must be updated to allow him to access the *`my-example-widget`* resource using the `sqs:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon SQS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon SQS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon SQS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon SQS supports these features, see How Amazon Simple Queue Service works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Using policies with Amazon SQS

This topic provides examples of identity-based policies in which an account administrator can attach permissions policies to IAM identities (users, groups, and roles).

> **Important**
>
> We recommend that you first review the introductory topics that explain the basic concepts and options available for you to manage access to your Amazon Simple Queue Service resources. For more information, see Overview of managing access in Amazon SQS.
> With the exception of `ListQueues`, all Amazon SQS actions support resource-level permissions. For more information, see Amazon SQS API permissions: Actions and resource reference.

Topics

- Using Amazon SQS and IAM policies
- Permissions required to use the Amazon SQS console
- Identity-based policy examples for Amazon SQS
- Basic examples of Amazon SQS policies

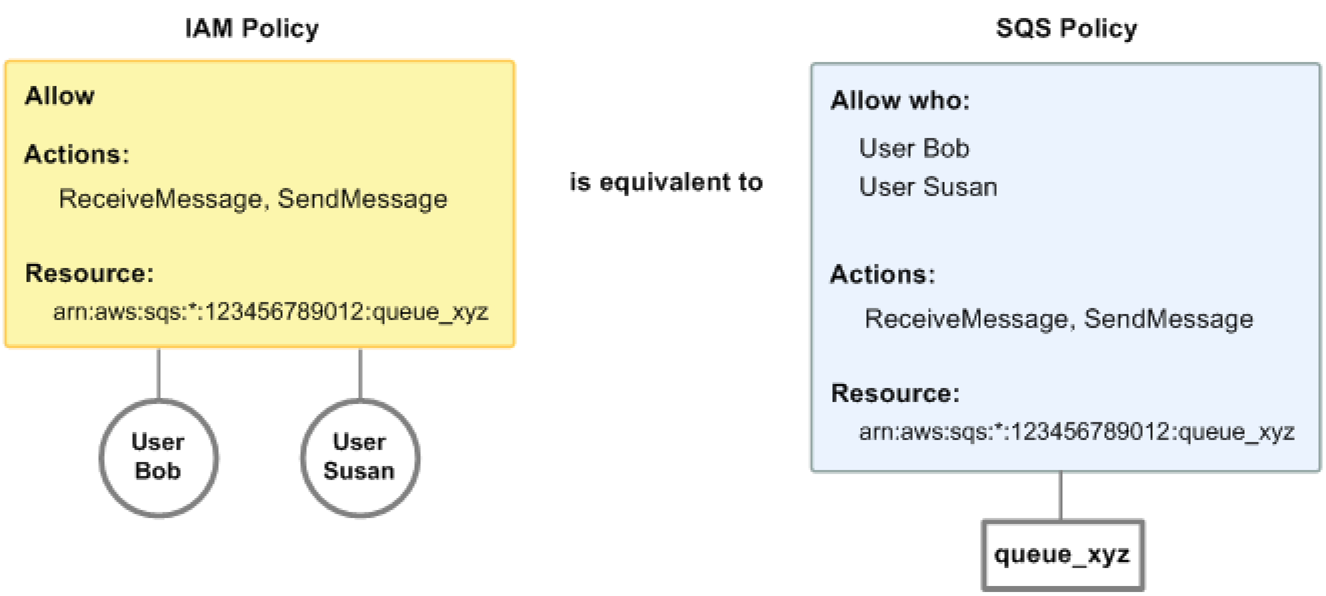
- [Using custom policies with the Amazon SQS Access Policy Language]

Using Amazon SQS and IAM policies

There are two ways to give your users permissions to your Amazon SQS resources: using the Amazon SQS policy system and using the IAM policy system. You can use one or the other, or both. For the most part, you can achieve the same result with either one.

For example, the following diagram shows an IAM policy and an Amazon SQS policy equivalent to it. The IAM policy grants the rights to the Amazon SQS `ReceiveMessage` and `SendMessage` actions for the queue called `queue_xyz` in your AWS Account, and the policy is attached to users named Bob and Susan (Bob and Susan have the permissions stated in the policy). This Amazon SQS policy also gives Bob and Susan rights to the `ReceiveMessage` and `SendMessage` actions for the same queue.

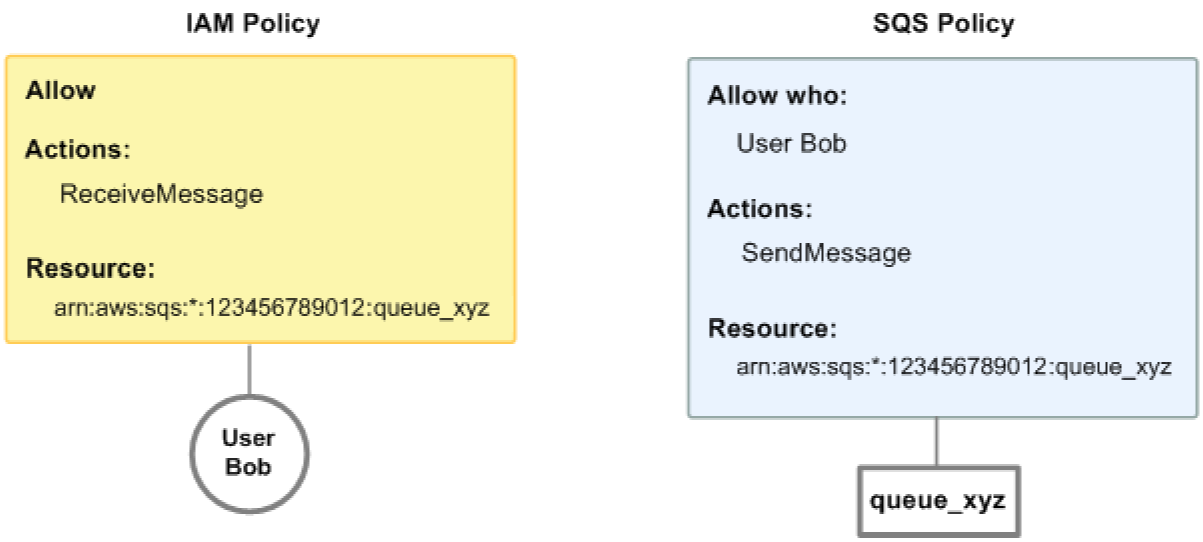
> **Note**
>
> The following example shows simple policies without conditions. You can specify a particular condition in either policy and get the same result.



There is one major difference between IAM and Amazon SQS policies: the Amazon SQS policy system lets you grant permission to other AWS Accounts, whereas IAM doesn't.

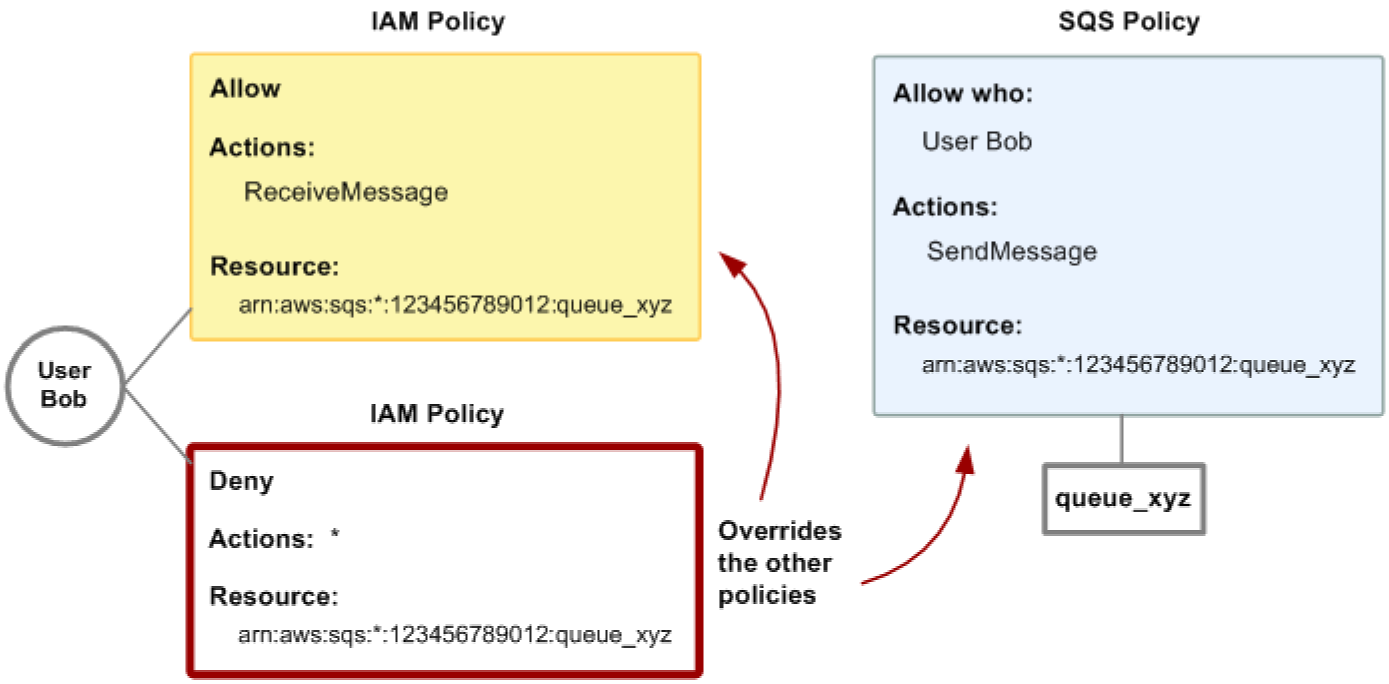
It is up to you how you use both of the systems together to manage your permissions. The following examples show how the two policy systems work together.

- In the first example, Bob has both an IAM policy and an Amazon SQS policy that apply to his account. The IAM policy grants his account permission for the `ReceiveMessage` action on `queue_xyz`, whereas the Amazon SQS policy gives his account permission for the `SendMessage` action on the same queue. The following diagram illustrates the concept.



  If Bob sends a `ReceiveMessage` request to `queue_xyz`, the IAM policy allows the action. If Bob sends a `SendMessage` request to `queue_xyz`, the Amazon SQS policy allows the action.

- In the second example, Bob abuses his access to `queue_xyz`, so it becomes necessary to remove his entire access to the queue. The easiest thing to do is to add a policy that denies him access to all actions for the queue. This policy overrides the other two because an explicit `deny` always overrides an `allow`. For more information about policy evaluation logic, see Using custom policies with the Amazon SQS Access Policy Language. The following diagram illustrates the concept.



You can also add an additional statement to the Amazon SQS policy that denies Bob any type of access to the queue. It has the same effect as adding an IAM policy that denies Bob access to the queue. For examples of policies that cover Amazon SQS actions and resources, see Basic examples of Amazon SQS policies. For more information about writing Amazon SQS policies, see Using custom policies with the Amazon SQS Access Policy Language.

Permissions required to use the Amazon SQS console

A user who wants to work with the Amazon SQS console must have the minimum set of permissions to work with the Amazon SQS queues in the user's AWS account. For example, the user must have the permission to call the `ListQueues` action to be able to list queues, or the permission to call the `CreateQueue` action to be able to create queues. In addition to Amazon SQS permissions, to subscribe an Amazon SQS queue to an Amazon SNS topic, the console also requires permissions for Amazon SNS actions.

If you create an IAM policy that is more restrictive than the minimum required permissions, the console might not function as intended for users with that IAM policy.

You don't need to allow minimum console permissions for users that make calls only to the AWS CLI or Amazon SQS actions.

Identity-based policy examples for Amazon SQS

By default, users and roles don't have permission to create or modify Amazon SQS resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

For details about actions and resource types defined by Amazon SQS, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon Simple Queue Service in the *Service Authorization Reference*.

> **Note**
>
> When you configure lifecycle hooks for Amazon EC2 Auto Scaling, you don't need to write a policy to send messages to an Amazon SQS queue. For more information, see Amazon EC2 Auto Scaling Lifecycle Hooks in the *Amazon EC2 User Guide*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon SQS resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon SQS console

To access the Amazon Simple Queue Service console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon SQS resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the Amazon SQS console, also attach the Amazon SQS `AmazonSQSReadOnlyAccess` AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "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": "*"
        }
    ]
}
```

Allow a user to create queues

In the following example, we create a policy for Bob that lets him access all Amazon SQS actions, but only with queues whose names are prefixed with the literal string `alice_queue_`.

Amazon SQS doesn't automatically grant the creator of a queue permissions to use the queue. Therefore, we must explicitly grant Bob permissions to use all Amazon SQS actions in addition to `CreateQueue` action in the IAM policy.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:123456789012:alice_queue_*"
   }]
}
```

Allow developers to write messages to a shared queue

In the following example, we create a group for developers and attach a policy that lets the group use the Amazon SQS `SendMessage` action, but only with the queue that belongs to the specified AWS account and is named `MyCompanyQueue`.

```
{
   "Version": "2012-10-17",
   "Statement": [{
      "Effect": "Allow",
      "Action": "sqs:SendMessage",
      "Resource": "arn:aws:sqs:*:123456789012:MyCompanyQueue"
   }]
}
```

You can use * instead of `SendMessage` to grant the following actions to a principal on a shared queue: `ChangeMessageVisibility`, `DeleteMessage`, `GetQueueAttributes`, `GetQueueUrl`, `ReceiveMessage`, and `SendMessage`.

> **Note**
>
> Although * includes access provided by other permission types, Amazon SQS considers permissions separately. For example, it is possible to grant both * and `SendMessage` permissions to a user, even though a * includes the access provided by `SendMessage`. This concept also applies when you remove a permission. If a principal has only a * permission, requesting to remove a `SendMessage` permission *doesn't* leave the principal with an *everything-but* permission. Instead, the request has no effect, because the principal doesn't possess an explicit `SendMessage` permission. To leave the principal with only the `ReceiveMessage` permission, first add the `ReceiveMessage` permission and then remove the * permission.

Allow managers to get the general size of queues

In the following example, we create a group for managers and attach a policy that lets the group use the Amazon SQS `GetQueueAttributes` action with all of the queues that belong to the specified AWS account.

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

Allow a partner to send messages to a specific queue

You can accomplish this task using an Amazon SQS policy or an IAM policy. If your partner has an AWS account, it might be easier to use an Amazon SQS policy. However, any user in the partner's company who possesses the AWS security credentials can send messages to the queue. If you want to limit access to a particular user or application, you must treat the partner like a user in your own company and use an IAM policy instead of an Amazon SQS policy.

This example performs the following actions:

1. Create a group called WidgetCo to represent the partner company.
2. Create a user for the specific user or application at the partner's company who needs access.
3. Add the user to the group.
4. Attach a policy that gives the group access only to the `SendMessage` action for only the queue named `WidgetPartnerQueue`.

```
{
   "Version": "2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "sqs:SendMessage",
         "Resource": "arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
   }]
}
```

Basic examples of Amazon SQS policies

This section shows example policies for common Amazon SQS use cases.

You can use the console to verify the effects of each policy as you attach the policy to the user. Initially, the user doesn't have permissions and won't be able to do anything in the console. As you attach policies to the user, you can verify that the user can perform various actions in the console.

> **Note**
>
> We recommend that you use two browser windows: one to grant permissions and the other to sign into the AWS Management Console using the user's credentials to verify permissions as you grant them to the user.

Example 1: Grant one permission to one AWS account

The following example policy grants AWS account number 111122223333 the `SendMessage` permission for the queue named `444455556666/queue1` in the US East (Ohio) region.

```
{
   "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"
    }]
}
```

Example 2: Grant two permissions to one AWS account

The following example policy grants AWS account number `111122223333` both the `SendMessage` and `ReceiveMessage` permission for the queue named `444455556666/queue1`.

```
{
    "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"
    }]
}
```

Example 3: Grant all permissions to two AWS accounts

The following example policy grants two different AWS accounts numbers (`111122223333` and `444455556666`) permission to use all actions to which Amazon SQS allows shared access for the queue named `123456789012/queue1` in the US East (Ohio) region.

```
{
   "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"
   }]
}
```

Example 4: Grant cross-account permissions to a role and a username

The following example policy grants `role1` and `username1` under AWS account number `111122223333` cross-account permission to use all actions to which Amazon SQS allows shared access for the queue named `123456789012/queue1` in the US East (Ohio) region.

Cross-account permissions don't apply to the following actions:

- `AddPermission`
- `CancelMessageMoveTask`
- `CreateQueue`
- `DeleteQueue`
- `ListMessageMoveTask`
- `ListQueues`
- `ListQueueTags`
- `RemovePermission`
- `SetQueueAttributes`
- `StartMessageMoveTask`
- `TagQueue`
- `UntagQueue`

```
{
   "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"
   }]
}
```

Example 5: Grant a permission to all users

The following example policy grants all users (anonymous users) `ReceiveMessage` permission for the queue named `111122223333/queue1`.

```
{
   "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"
   }]
}
```

Example 6: Grant a time-limited permission to all users

The following example policy grants all users (anonymous users) `ReceiveMessage` permission for the queue named `111122223333/queue1`, but only between 12:00 p.m. (noon) and 3:00 p.m. on January 31, 2009.

```
{
```

```
    "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"
            }
        }
    }]
}
```

Example 7: Grant all permissions to all users in a CIDR range

The following example policy grants all users (anonymous users) permission to use all possible Amazon SQS actions that can be shared for the queue named `111122223333/queue1`, but only if the request comes from the `192.0.2.0/24` CIDR range.

```
{
    "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"
            }
        }
    }]
}
```

Example 8: Allowlist and blocklist permissions for users in different CIDR ranges

The following example policy has two statements:

- The first statement grants all users (anonymous users) in the `192.0.2.0/24` CIDR range (except for `192.0.2.188`) permission to use the `SendMessage` action for the queue named 111122223333/queue1.
- The second statement blocks all users (anonymous users) in the `12.148.72.0/23` CIDR range from using the queue.

```
{
   "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"
         }
      }
   }]
}
```

Using custom policies with the Amazon SQS Access Policy Language

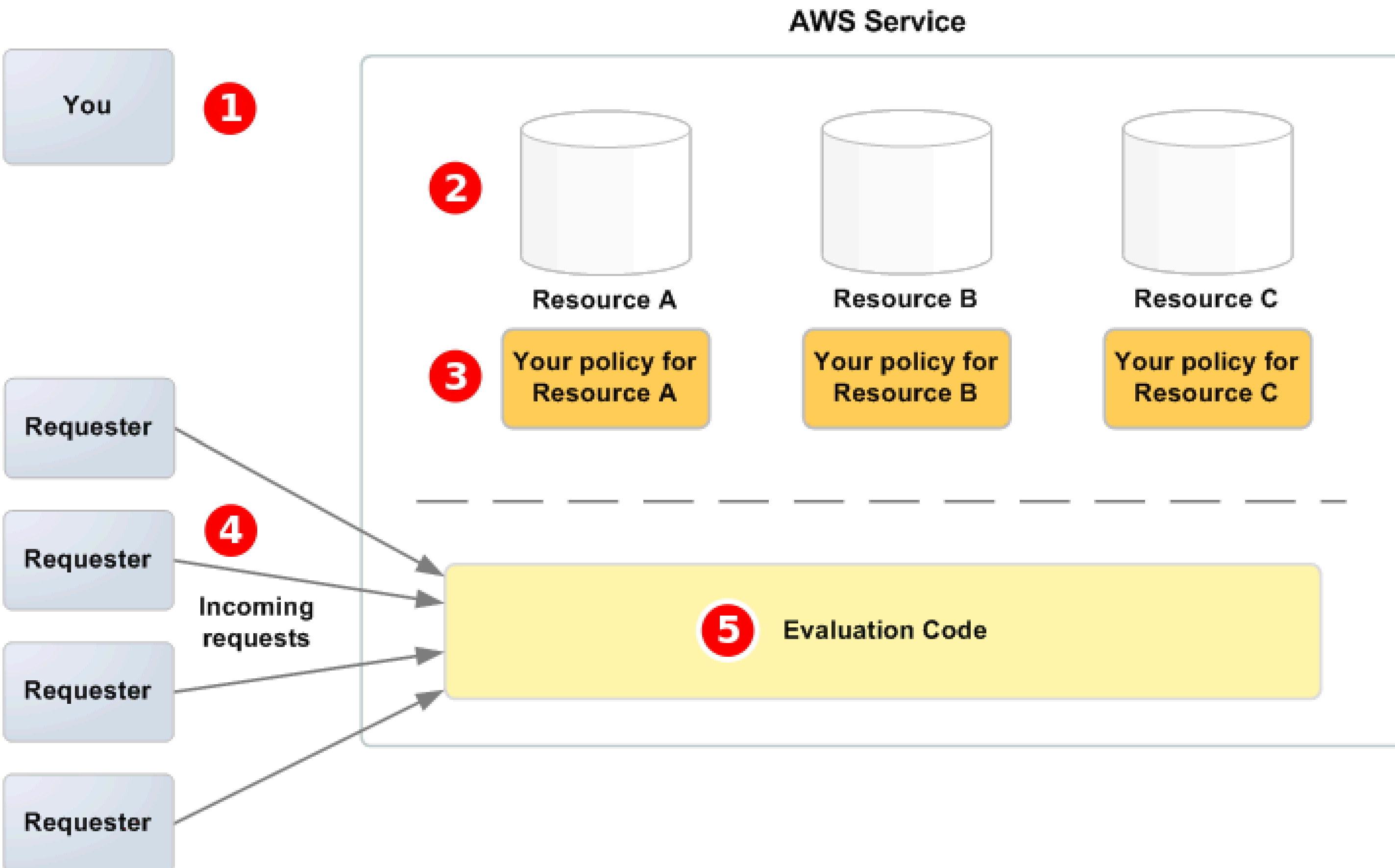
If you want to allow Amazon SQS access based only on an AWS account ID and basic permissions (such as for `SendMessage` or `ReceiveMessage`), you don't need to write your own policies. You can just use the Amazon SQS `AddPermission` action.

If you want to explicitly deny or allow access based on more specific conditions (such as the time the request comes in or the IP address of the requester), you need to write your own Amazon SQS policies and upload them to the AWS system using the Amazon SQS `SetQueueAttributes` action.

Topics

Amazon SQS access control architecture

The following diagram describes the access control for your Amazon SQS resources.



1

You, the resource owner.

2

Your resources contained within the AWS service (for example, Amazon SQS queues).

3

Your policies. It is a good practice to have one policy per resource. The AWS service provides an API you use to upload and manage your policies.

4

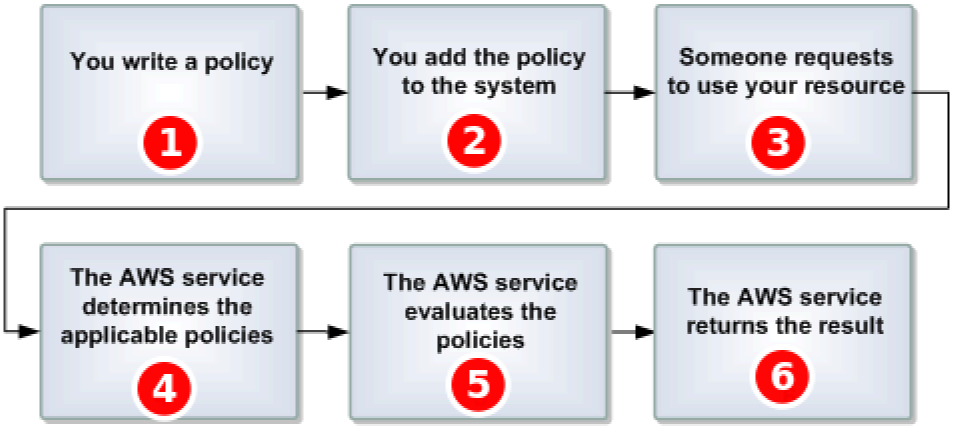
Requesters and their incoming requests to the AWS service.

5

The access policy language evaluation code. This is the set of code within the AWS service that evaluates incoming requests against the applicable policies and determines whether the requester is allowed access to the resource.

Amazon SQS access control process workflow

The following diagram describes the general workflow of access control with the Amazon SQS access policy language.



1

You write an Amazon SQS policy for your queue.

2

You upload your policy to AWS. The AWS service provides an API that you use to upload your policies. For example, you use the Amazon SQS `SetQueueAttributes` action to upload a policy for a particular Amazon SQS queue.

3

Someone sends a request to use your Amazon SQS queue.

4

Amazon SQS examines all available Amazon SQS policies and determines which ones are applicable.

5

Amazon SQS evaluates the policies and determines whether the requester is allowed to use your queue.

6

Based on the policy evaluation result, Amazon SQS either returns an `Access denied` error to the requester or continues to process the request.

Amazon SQS Access Policy Language key concepts

To write your own policies, you must be familiar with JSON and a number of key concepts.

Allow

The result of a **Statement** that has **Effect** set to `allow`.

Action

The activity that the **Principal** has permission to perform, typically a request to AWS.

Default-deny

The result of a **Statement** that has no **Allow** or **Explicit-deny** settings.

Condition

Any restriction or detail about a **Permission**. Typical conditions are related to date and time and IP addresses.

Effect

The result that you want the **Statement** of a **Policy** to return at evaluation time. You specify the `deny` or `allow` value when you write the policy statement. There can be three possible results at policy evaluation time: **Default-deny**, **Allow**, and **Explicit-deny**.

Explicit-deny

The result of a **Statement** that has **Effect** set to `deny`.

Evaluation

The process that Amazon SQS uses to determine whether an incoming request should be denied or allowed based on a **Policy**.

Issuer

The user who writes a **Policy** to grant permissions to a resource. The issuer, by definition is always the resource owner. AWS doesn't permit Amazon SQS users to create policies for resources they don't own.

Key

The specific characteristic that is the basis for access restriction.

Permission

The concept of allowing or disallowing access to a resource using a **Condition** and a **Key**.

Policy

The document that acts as a container for one or more **statements**.

Amazon SQS uses the policy to determine whether to grant access to a user for a resource.

Principal

The user who receives **Permission** in the **Policy**.

Resource

The object that the **Principal** requests access to.

Statement

The formal description of a single permission, written in the access policy language as part of a broader **Policy** document.

Requester

The user who sends a request for access to a **Resource**.

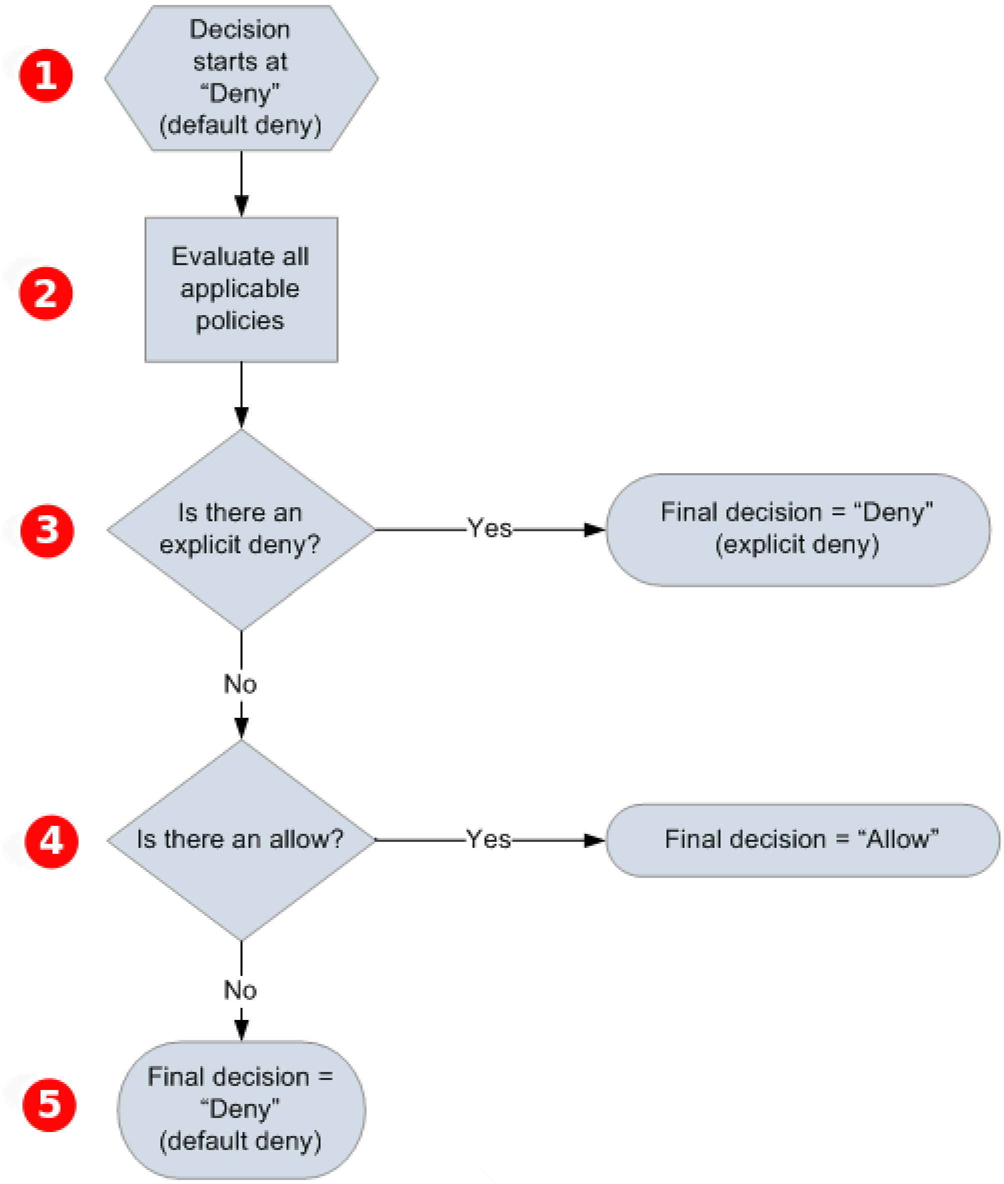
Amazon SQS Access Policy Language evaluation logic

At evaluation time, Amazon SQS determines whether a request from someone other than the resource owner should be allowed or denied. The evaluation logic follows several basic rules:

- By default, all requests to use your resource coming from anyone but you are denied.
- An ***Allow*** overrides any ***Default-deny***.

- An ***Explicit-deny*** overrides any **allow**.
- The order in which the policies are evaluated isn't important.

The following diagram describes in detail how Amazon SQS evaluates decisions about access permissions.



1

The decision starts with a **default-deny**.

2

The enforcement code evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies isn't important.

3

The enforcement code looks for an **explicit-deny** instruction that can apply to the request. If it finds even one, the enforcement code returns a decision of **deny** and the process finishes.

4

If no **explicit-deny** instruction is found, the enforcement code looks for any **allow** instructions that can apply to the request. If it finds even one, the enforcement code returns a decision of **allow** and the process finishes (the service continues to process the request).

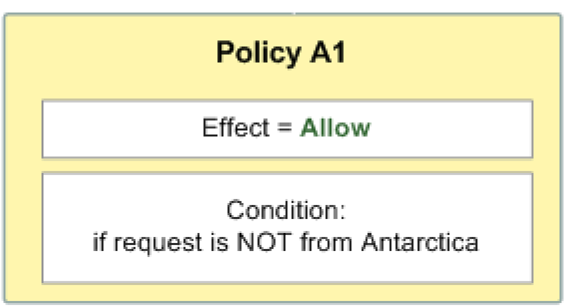
5

If no **allow** instruction is found, then the final decision is **deny** (because there is no **explicit-deny** or **allow**, this is considered a **default-deny**).

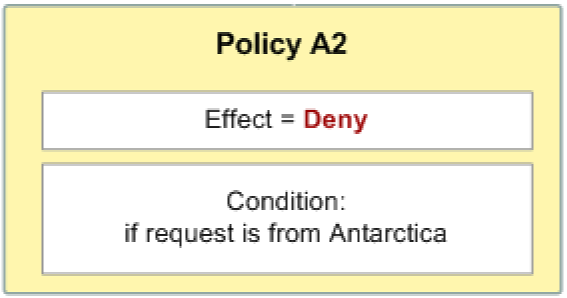
Relationships between explicit and default denials in the Amazon SQS Access Policy Language

If an Amazon SQS policy doesn't directly apply to a request, the request results in a ***Default-deny***. For example, if a user requests permission to use Amazon SQS but the only policy that applies to the user can use DynamoDB, the requests results in a **default-deny**.

If a condition in a statement isn't met, the request results in a **default-deny**. If all conditions in a statement are met, the request results in either an ***Allow*** or an ***Explicit-deny*** based on the value of the ***Effect*** element of the policy. Policies don't specify what to do if a condition isn't met, so the default result in this case is a **default-deny**. For example, you want to prevent requests that come from Antarctica. You write Policy A1 that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the Amazon SQS policy.

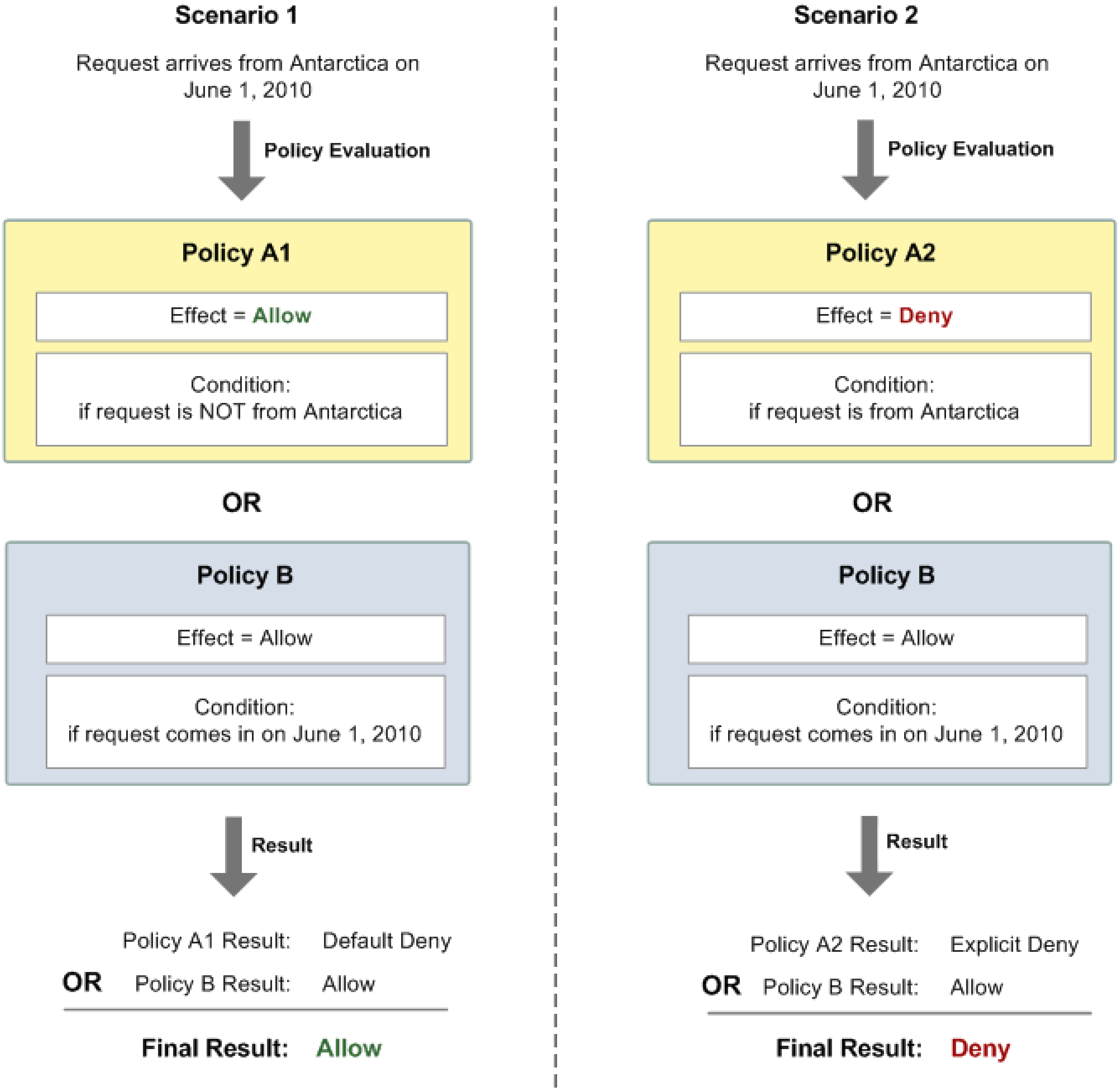


If a user sends a request from the U.S., the condition is met (the request isn't from Antarctica), and the request results in an **allow**. However, if a user sends a request from Antarctica, the condition isn't met and the request defaults to a **default-deny**. You can change the result to an **explicit-deny** by writing Policy A2 that explicitly denies a request if it comes from Antarctica. The following diagram illustrates the policy.



If a user sends a request from Antarctica, the condition is met and the request results in an **explicit-deny**.

The distinction between a **default-deny** and an **explicit-deny** is important because an **allow** can overwrite the former but not the latter. For example, Policy B allows requests if they arrive on June 1, 2010. The following diagram compares combining this policy with Policy A1 and Policy A2.



In Scenario 1, Policy A1 results in a **default-deny** and Policy B results in an **allow** because the policy allows requests that come in on June 1, 2010. The **allow** from Policy B overrides the **default-deny** from Policy A1, and the request is allowed.

In Scenario 2, Policy B2 results in an **explicit-deny** and Policy B results in an **allow**. The **explicit-deny** from Policy A2 overrides the **allow** from Policy B, and the request is denied.

Limitations of Amazon SQS custom policies

Cross-account access

Cross-account permissions don't apply to the following actions:

- `AddPermission`
- `CancelMessageMoveTask`
- `CreateQueue`
- `DeleteQueue`
- `ListMessageMoveTask`
- `ListQueues`
- `ListQueueTags`
- `RemovePermission`
- `SetQueueAttributes`
- `StartMessageMoveTask`
- `TagQueue`
- `UntagQueue`

Condition keys

Currently, Amazon SQS supports only a limited subset of the condition keys available in IAM. For more information, see Amazon SQS API permissions: Actions and resource reference.

Custom Amazon SQS Access Policy Language examples

The following are examples of typical Amazon SQS access policies.

Example 1: Give permission to one account

The following example Amazon SQS policy gives AWS account 111122223333 permission to send to and receive from `queue2` owned by AWS account 444455556666.

```
{
   "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"
   }]
}
```

Example 2: Give permission to one or more accounts

The following example Amazon SQS policy gives one or more AWS accounts access to queues owned by your account for a specific time period. It is necessary to write this policy and to upload it to Amazon SQS using the `SetQueueAttributes` action because the `AddPermission` action doesn't permit specifying a time restriction when granting access to a queue.

```
{
   "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"
         }
      }
   }]
}
```

Example 3: Give permission to requests from Amazon EC2 instances

The following example Amazon SQS policy gives access to requests that come from Amazon EC2 instances. This example builds on the "Example 2: Give permission to one or more accounts" example: it restricts access to before June 30, 2009 at 12 noon (UTC), it restricts access to the IP range `203.0.113.0/24`. It is necessary to write this policy and to upload it to Amazon SQS using the `SetQueueAttributes` action because the `AddPermission` action doesn't permit specifying an IP address restriction when granting access to a queue.

```
{
   "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"
         }
      }
   }]
}
```

Example 4: Deny access to a specific account

The following example Amazon SQS policy denies a specific AWS account access to your queue. This example builds on the "Example 1: Give permission to one account" example: it denies access to the specified AWS account. It is necessary to write this policy and to upload it to Amazon SQS using the `SetQueueAttributes` action because the `AddPermission` action doesn't permit deny access to a queue (it allows only granting access to a queue).

```
{
   "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"
   }]
}
```

Example 5: Deny access if it isn't from a VPC endpoint

The following example Amazon SQS policy restricts access to `queue1`: 111122223333 can perform the `SendMessage` and `ReceiveMessage` actions only from the VPC endpoint ID `vpce-1a2b3c4d` (specified using the `aws:sourceVpce` condition). For more information, see Amazon Virtual Private Cloud endpoints for Amazon SQS.

> **Note**
>
> - The `aws:sourceVpce` condition doesn't require an ARN for the VPC endpoint resource, only the VPC endpoint ID.
> - You can modify the following example to restrict all actions to a specific VPC endpoint by denying all Amazon SQS actions (`sqs:*`) in the second statement. However, such a

policy statement would stipulate that all actions (including administrative actions needed to modify queue permissions) must be made through the specific VPC endpoint defined in the policy, potentially preventing the user from modifying queue permissions in the future.

```
{
   "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"
            }
         }
      }
   ]
}
```

Using temporary security credentials with Amazon SQS

In addition to creating users with their own security credentials, IAM also allows you to grant temporary security credentials to any user, allowing the user to access your AWS services and resources. You can manage users who have AWS accounts. You can also manage users for your system who don't have AWS accounts (federated users). In addition, applications that you create to access your AWS resources can also be considered to be "users."

You can use these temporary security credentials to make requests to Amazon SQS. The API libraries compute the necessary signature value using those credentials to authenticate your request. If you send requests using expired credentials, Amazon SQS denies the request.

> **Note**
>
> You can't set a policy based on temporary credentials.

Prerequisites

1. Use IAM to create temporary security credentials:
   - Security token
   - Access Key ID
   - Secret Access Key
2. Prepare your string to sign with the temporary Access Key ID and the security token.
3. Use the temporary Secret Access Key instead of your own Secret Access Key to sign your Query API request.

> **Note**
>
> When you submit the signed Query API request, use the temporary Access Key ID instead of your own Access Key ID and to include the security token. For more information about IAM support for temporary security credentials, see [Granting Temporary Access to Your AWS Resources]() in the *IAM User Guide*.

To call an Amazon SQS Query API action using temporary security credentials

1. Request a temporary security token using AWS Identity and Access Management. For more information, see Creating Temporary Security Credentials to Enable Access for IAM Users in the *IAM User Guide*.

   IAM returns a security token, an Access Key ID, and a Secret Access Key.

2. Prepare your query using the temporary Access Key ID instead of your own Access Key ID and include the security token. Sign your request using the temporary Secret Access Key instead of your own.

3. Submit your signed query string with the temporary Access Key ID and the security token.

   The following example demonstrates how to use temporary security credentials to authenticate an Amazon SQS request. The structure of *AUTHPARAMS* depends on the signature of the API request. For more information, see Signing AWS API Requests in the *Amazon Web Services General Reference*.

```
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
```

   The following example uses temporary security credentials to send two messages using the `SendMessageBatch` action.

```
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
```

Access management for encrypted Amazon SQS queues with least privilege policies

You can use Amazon SQS to exchange sensitive data between applications by using server-side encryption (SSE) integrated with AWS Key Management Service (KMS). With the integration of Amazon SQS and AWS KMS, you can centrally manage the keys that protect Amazon SQS, as well as the keys that protect your other AWS resources.

Multiple AWS services can act as event sources that send events to Amazon SQS. To enable an event source to access the encrypted Amazon SQS queue, you need to configure the queue with a customer-managed AWS KMS key. Then, use the key policy to allow the service to use the required AWS KMS API methods. The service also requires permissions to authenticate access to enable the queue to send events. You can achieve this by using an Amazon SQS policy, which is a resource-based policy that you can use to control access to the Amazon SQS queue and its data.

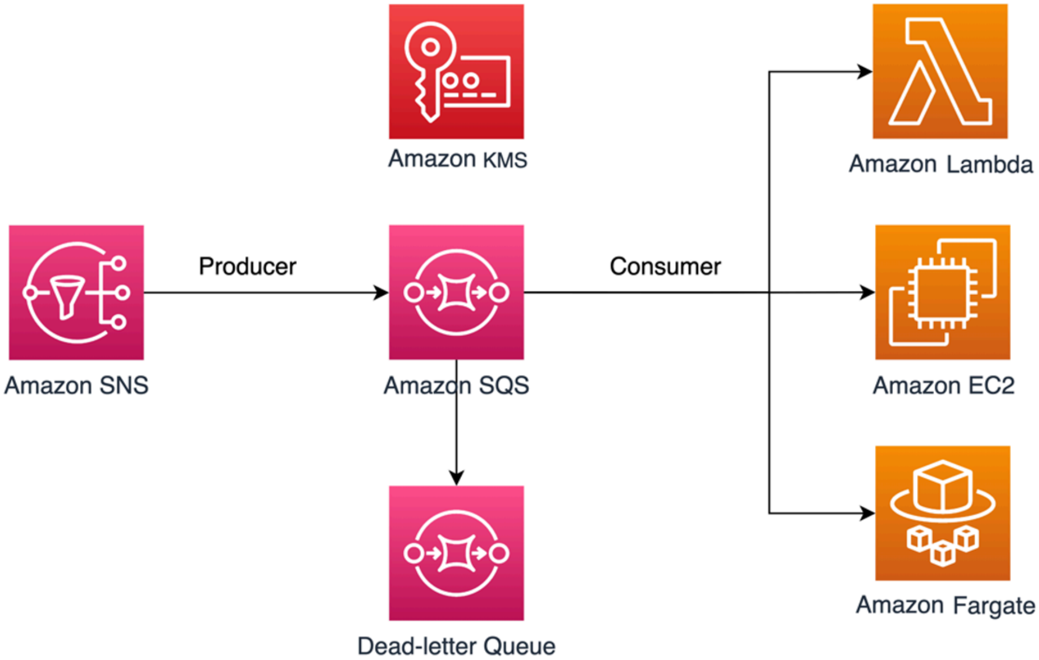
The following sections provide information on how to control access to your encrypted Amazon SQS queue through the Amazon SQS policy and the AWS KMS key policy. The policies in this guide will help you achieve least privilege.

This guide also describes how resource-based policies address the confused-deputy problem by using the `aws:SourceArn`, `aws:SourceAccount`, and `aws:PrincipalOrgID` global IAM condition context keys.

Topics

Overview

In this topic, we will walk you through a common use case to illustrate how you can build the key policy and the Amazon SQS queue policy. This use case is shown in the following image.



In this example, the message producer is an [Amazon Simple Notification Service (SNS)] topic, which is configured to fanout messages to your encrypted Amazon SQS queue. The message consumer is a compute service, such as an AWS Lambda function, an Amazon Elastic Compute Cloud (EC2) instance, or an AWS Fargate container. Your Amazon SQS queue is then configured to send failed messages to a Dead-letter Queue (DLQ). This is useful for debugging your application or messaging system because DLQs let you isolate unconsumed messages to determine why their processing didn't succeed. In the solution defined in this topic, a compute service such as a Lambda function is used to process messages stored in the Amazon SQS queue. If the message consumer is located in a virtual private cloud (VPC), the `DenyReceivingIfNotThroughVPCE` policy statement included in this guide lets you restrict message reception to that specific VPC.

> **Note**
>
> This guide contains only the required IAM permissions in the form of policy statements. To construct the policy, you need to add the statements to your Amazon SQS policy or your AWS KMS key policy. This guide doesn't provide instructions on how to create the Amazon SQS queue or the AWS KMS key. For instructions on how to create these resources, see Creating an Amazon SQS queue and Creating keys.
> The Amazon SQS policy defined in this guide doesn’t support redriving messages directly to the same or a different Amazon SQS queue.

Least privilege key policy for Amazon SQS

In this section, we describe the required least privilege permissions in AWS KMS for the customer-managed key that you use to encrypt your Amazon SQS queue. With these permissions, you can limit access to only the intended entities while implementing least privilege. The key policy must consist of the following policy statements, which we describe in detail below:

- Grant administrator permissions to the AWS KMS key
- Grant read-only access to the key metadata
- Grant Amazon SNS KMS permissions to Amazon SNS to publish messages to the queue
- Allow consumers to decrypt messages from the queue

Grant administrator permissions to the AWS KMS key

To create an AWS KMS key, you need to provide AWS KMS administrator permissions to the IAM role that you use to deploy the AWS KMS key. These administrator permissions are defined in the following `AllowKeyAdminPermissions` policy statement. When you add this statement to your AWS KMS key policy, make sure to replace *<admin-role ARN>* with the Amazon Resource Name (ARN) of the IAM role used to deploy the AWS KMS key, manage the AWS KMS key, or both. This can be the IAM role of your deployment pipeline, or the administrator role for your organization in your AWS Organizations.

```
{
  "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**
>
> In an AWS KMS key policy, the value of the `Resource` element needs to be `*`, which means "this AWS KMS key". The asterisk (`*`) identifies the AWS KMS key to which the key policy is attached.

Grant read-only access to the key metadata

To grant other IAM roles read-only access to your key metadata, add the `AllowReadAccessToKeyMetaData` statement to your key policy. For example, the following statement lets you list all of the AWS KMS keys in your account for auditing purposes. This statement grants the AWS root user read-only access to the key metadata. Therefore, any IAM principal in the account can have access to the key metadata when their identity-based policies have the permissions listed in the following statement: `kms:Describe*`, `kms:Get*`, and `kms:List*`. Make sure to replace *<account-ID>* with your own information.

```
{
  "Sid": "AllowReadAcesssToKeyMetaData",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "arn:aws:iam::<accountID>:root"
    ]
  },
  "Action": [
    "kms:Describe*",
    "kms:Get*",
    "kms:List*"
  ],
  "Resource": "*"
```

```
}
```

Grant Amazon SNS KMS permissions to Amazon SNS to publish messages to the queue

To allow your Amazon SNS topic to publish messages to your encrypted Amazon SQS queue, add the `AllowSNSToSendToSQS` policy statement to your key policy. This statement grants Amazon SNS permissions to use the AWS KMS key to publish to your Amazon SQS queue. Make sure to replace *<account-ID>* with your own information.

> **Note**
>
> The `Condition` in the statement limits access to only the Amazon SNS service in the same AWS account.

```
{
  "Sid": "AllowSNSToSendToSQS",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "sns.amazonaws.com"
    ]
  },
  "Action": [
    "kms:GenerateDataKey",
    "kms:Decrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "aws:SourceAccount": "<account-id>"
    }
  }
}
```

Allow consumers to decrypt messages from the queue

The following `AllowConsumersToReceiveFromTheQueue` statement grants the Amazon SQS message consumer the required permissions to decrypt messages received from the encrypted Amazon SQS queue. When you attach the policy statement, replace *<consumer's runtime role ARN>* with the IAM runtime role ARN of the message consumer.

```
{
  "Sid": "AllowConsumersToReceiveFromTheQueue",
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "<consumer's execution role ARN>"
    ]
  },
  "Action": [
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

Least privilege Amazon SQS policy

This section walks you through the least privilege Amazon SQS queue policies for the use case covered by this guide (for example, Amazon SNS to Amazon SQS). The defined policy is designed to prevent unintended access by using a mix of both `Deny` and `Allow` statements. The `Allow` statements grant access to the intended entity or entities. The `Deny` statements prevent other unintended entities from accessing the Amazon SQS queue, while excluding the intended entity within the policy condition.

The Amazon SQS policy includes the following statements, which we describe in detail below:

- Restrict Amazon SQS management permissions
- Restrict Amazon SQS queue actions from the specified organization
- Grant Amazon SQS permissions to consumers
- Enforce encryption in transit
- Restrict message transmission to a specific Amazon SNS topic
- (Optional) Restrict message reception to a specific VPC endpoint

Restrict Amazon SQS management permissions

The following `RestrictAdminQueueActions` policy statement restricts the Amazon SQS management permissions to only the IAM role or roles that you use to deploy the queue, manage the queue, or both. Make sure to replace the *`<placeholder values>`* with your own information. Specify the ARN of the IAM role used to deploy the Amazon SQS queue, as well as the ARNs of any administrator roles that should have Amazon SQS management permissions.

```
{
  "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>"
      ]
    }
  }
}
```

Restrict Amazon SQS queue actions from the specified organization

To help protect your Amazon SQS resources from external access (access by an entity outside of your AWS organization), use the following statement. This statement limits Amazon SQS queue access to the organization that you specify in the `Condition`. Make sure to replace *`<SQS queue ARN>`* with the ARN of the IAM role used to deploy the Amazon SQS queue; and the *`<org-id>`*, with your organization ID.

```
{
  "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>"
        ]
      }
    }
}
```

Grant Amazon SQS permissions to consumers

To receive messages from the Amazon SQS queue, you need to provide the message consumer with the necessary permissions. The following policy statement grants the consumer, which you specify, the required permissions to consume messages from the Amazon SQS queue. When adding the statement to your Amazon SQS policy, make sure to replace *`<consumer's IAM runtime role ARN>`* with the ARN of the IAM runtime role used by the consumer; and *`<SQS queue ARN>`*, with the ARN of the IAM role used to deploy the Amazon SQS queue.

```
{
  "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>"
}
```

To prevent other entities from receiving messages from the Amazon SQS queue, add the `DenyOtherConsumersFromReceiving` statement to the Amazon SQS queue policy. This statement restricts message consumption to the consumer that you specify—allowing no other consumers to have access, even when their identity-permissions would grant them access. Make sure to replace *<SQS queue ARN>* and *<consumer’s runtime role ARN>* with your own information.

```
{
  "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>"
    }
  }
}
```

Enforce encryption in transit

The following `DenyUnsecureTransport` policy statement enforces the consumers and producers to use secure channels (TLS connections) to send and receive messages from the Amazon SQS queue. Make sure to replace *<SQS queue ARN>* with the ARN of the IAM role used to deploy the Amazon SQS queue.

```
{
  "Sid": "DenyUnsecureTransport",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": [
    "sqs:ReceiveMessage",
    "sqs:SendMessage"
  ],
  "Resource": "<SQS queue ARN>",
```

```
  "Condition": {
    "Bool": {
      "aws:SecureTransport": "false"
    }
  }
}
```

Restrict message transmission to a specific Amazon SNS topic

The following `AllowSNSToSendToTheQueue` policy statement allows the specified Amazon SNS topic to send messages to the Amazon SQS queue. Make sure to replace *<SQS queue ARN>* with the ARN of the IAM role used to deploy the Amazon SQS queue; and *<SNS topic ARN>*, with the Amazon SNS topic ARN.

```
{
  "Sid": "AllowSNSToSendToTheQueue",
  "Effect": "Allow",
  "Principal": {
    "Service": "sns.amazonaws.com"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

The following `DenyAllProducersExceptSNSFromSending` policy statement prevents other producers from sending messages to the queue. Replace *<SQS queue ARN>* and *<SNS topic ARN>* with your own information.

```
{
  "Sid": "DenyAllProducersExceptSNSFromSending",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
```

```
  "Action": "sqs:SendMessage",
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "<SNS topic ARN>"
    }
  }
}
```

(Optional) Restrict message reception to a specific VPC endpoint

To restrict the receipt of messages to only a specific VPC endpoint, add the following policy statement to your Amazon SQS queue policy. This statement prevents a message consumer from receiving messages from the queue unless the messages are from the desired VPC endpoint. Replace *<SQS queue ARN>* with the ARN of the IAM role used to deploy the Amazon SQS queue; and *<vpce_id>* with the ID of the VPC endpoint.

```
{
  "Sid": "DenyReceivingIfNotThroughVPCE",
  "Effect": "Deny",
  "Principal": "*",
  "Action": [
    "sqs:ReceiveMessage"
  ],
  "Resource": "<SQS queue ARN>",
  "Condition": {
    "StringNotEquals": {
      "aws:sourceVpce": "<vpce id>"
    }
  }
}
```

Amazon SQS policy statements for the dead-letter queue

Add the following policy statements, identified by their statement ID, to your DLQ access policy:

- `RestrictAdminQueueActions`
- `DenyQueueActionsOutsideOrg`
- `AllowConsumersToReceiveFromTheQueue`

- `DenyOtherConsumersFromReceiving`
- `DenyUnsecureTransport`

In addition to adding the preceding policy statements to your DLQ access policy, you should also add a statement to restrict message transmission to Amazon SQS queues, as described in the following section.

Restrict message transmission to Amazon SQS queues

To restrict access to only Amazon SQS queues from the same account, add the following `DenyAnyProducersExceptSQS` policy statement to the DLQ queue policy. This statement doesn't limit message transmission to a specific queue because you need to deploy the DLQ before you create the main queue, so you won't know the Amazon SQS ARN when you create the DLQ. If you need to limit access to only one Amazon SQS queue, modify the `aws:SourceArn` in the `Condition` with the ARN of your Amazon SQS source queue when you know it.

```
{
  "Sid": "DenyAnyProducersExceptSQS",
  "Effect": "Deny",
  "Principal": {
    "AWS": "*"
  },
  "Action": "sqs:SendMessage",
  "Resource": "<SQS DLQ ARN>",
  "Condition": {
    "ArnNotLike": {
      "aws:SourceArn": "arn:aws:sqs:<region>:<account-id>:*"
    }
  }
}
```

> **Important**
>
> The Amazon SQS queue policies defined in this guide don't restrict the `sqs:PurgeQueue` action to a certain IAM role or roles. The `sqs:PurgeQueue` action enables you to delete all messages in the Amazon SQS queue. You can also use this action to make changes to the message format without replacing the Amazon SQS queue. When debugging an application, you can clear the Amazon SQS queue to remove potentially erroneous messages. When testing the application, you can drive a high message volume through the

Amazon SQS queue and then purge the queue to start fresh before entering production. The reason for not restricting this action to a certain role is that this role might not be known when deploying the Amazon SQS queue. You will need to add this permission to the role's identity-based policy to be able to purge the queue.

Prevent the cross-service confused deputy problem

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more privileged entity to perform the action. To prevent this, AWS provides tools that help you protect your account if you provide third parties (known as cross-account) or other AWS services (known as cross-service) access to resources in your account. The policy statements in this section can help you prevent the cross-service confused deputy problem.

Cross-service impersonation can occur when one service (the calling service) calls another service (the called service). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it shouldn't otherwise have permission to access. To help protect against this issue, the resource-based policies defined in this post use the `aws:SourceArn`, `aws:SourceAccount`, and `aws:PrincipalOrgID` global IAM condition context keys. This limits the permissions that a service has to a specific resource, a specific account, or a specific organization in AWS Organizations.

Use IAM Access Analyzer to review cross-account access

You can use AWS IAM Access Analyzer to review your Amazon SQS queue policies and AWS KMS key policies and alert you when an Amazon SQS queue or a AWS KMS key grants access to an external entity. IAM Access Analyzer helps identify resources in your organization and accounts that are shared with an entity outside the zone of trust. This zone of trust can be an AWS account or the organization within AWS Organizations that you specify when you enable IAM Access Analyzer.

IAM Access Analyzer identifies resources shared with external principals by using logic-based reasoning to analyze the resource-based policies in your AWS environment. For each instance of a resource shared outside of your zone of trust, Access Analyzer generates a finding. Findings include information about the access and the external principal granted to it. Review the findings to determine whether the access is intended and safe, or whether the access is unintended and a security risk. For any unintended access, review the affected policy and fix it. Refer to this blog post for more information on how AWS IAM Access Analyzer identifies unintended access to your AWS resources.

For more information on AWS IAM Access Analyzer, see the AWS IAM Access Analyzer documentation.

Amazon SQS API permissions: Actions and resource reference

When you set up Access control and write permissions policies that you can attach to an IAM identity, you can use the following table as a reference. The list includes each Amazon Simple Queue Service action, the corresponding actions for which you can grant permissions to perform the action, and the AWS resource for which you can grant the permissions.

Specify the actions in the policy's `Action` field, and the resource value in the policy's `Resource` field. To specify an action, use the `sqs:` prefix followed by the action name (for example, `sqs:CreateQueue`).

Currently, Amazon SQS supports the global condition context keys available in IAM.

Amazon Simple Queue Service API and required permissions for actions

AddPermission

Action(s): `sqs:AddPermission`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

ChangeMessageVisibility

Action(s): `sqs:ChangeMessageVisibility`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

ChangeMessageVisibilityBatch

Action(s): `sqs:ChangeMessageVisibilityBatch`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

CreateQueue

Action(s): `sqs:CreateQueue`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

DeleteMessage

Action(s): `sqs:DeleteMessage`

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

DeleteMessageBatch

Action(s): sqs:DeleteMessageBatch

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

DeleteQueue

Action(s): sqs:DeleteQueue

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

GetQueueAttributes

Action(s): sqs:GetQueueAttributes

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

GetQueueUrl

Action(s): sqs:GetQueueUrl

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ListDeadLetterSourceQueues

Action(s): sqs:ListDeadLetterSourceQueues

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ListQueues

Action(s): sqs:ListQueues

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ListQueueTags

Action(s): sqs:ListQueueTags

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

PurgeQueue

Action(s): sqs:PurgeQueue

Resource: arn:aws:sqs:*region*:*account_id*:*queue_name*

ReceiveMessage

Action(s): `sqs:ReceiveMessage`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

RemovePermission

Action(s): `sqs:RemovePermission`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

SendMessage and SendMessageBatch

Action(s): `sqs:SendMessage`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

SetQueueAttributes

Action(s): `sqs:SetQueueAttributes`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

TagQueue

Action(s): `sqs:TagQueue`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

UntagQueue

Action(s): `sqs:UntagQueue`

Resource: `arn:aws:sqs:`*`region`*`:`*`account_id`*`:`*`queue_name`*

Logging and monitoring in Amazon SQS

This section provides information about logging and monitoring options for Amazon SQS, including how to use CloudTrail to capture API calls, and CloudWatch metrics to gain insights into queue activity and performance.

Topics

- Logging Amazon SQS API calls using AWS CloudTrail
- Monitoring Amazon SQS queues using CloudWatch

Logging Amazon SQS API calls using AWS CloudTrail

Amazon SQS is integrated with AWS CloudTrail to record the Amazon SQS calls from a user, role, or AWS service. CloudTrail captures API calls related to Amazon SQS standard and FIFO queues as *events*, including interactions initiated through the Amazon SQS console as well as programmatically via calls to the Amazon SQS APIs.

Topics

Amazon SQS information in CloudTrail

CloudTrail is turned on by default when you create your AWS account. When a supported Amazon SQS event activity occurs, it is recorded in a CloudTrail event, along with other AWS service events, in the event history. You can view, search, and download recent events for your AWS account. For more information, see Viewing Events with CloudTrail Event History in the *AWS CloudTrail User Guide*.

Amazon SQS APIs that call queue management operations, such as `AddPermission` are categorized as management events and are logged in CloudTrail by default. Amazon SQS APIs that are high volume operations performed on an Amazon SQS queue, such as `SendMessage` are categorized as data events and are logged after you opt-in with CloudTrail.

Using the information that CloudTrail collects, you can identify a specific request to an Amazon SQS API, the IP address or identity of the requester, and the date and time of the request. If you configure a CloudTrail *trail*, you can continuously deliver CloudTrail events to an Amazon S3 bucket with an optional delivery to Amazon CloudWatch Logs and AWS EventBridge. If you do not configure a trail, you can only view the event history of management events in events in the CloudTrail console. For more information, see Overview for Creating a Trail in the *AWS CloudTrail User Guide*.

Management events in CloudTrail

Amazon SQS logs the following API actions as management events:

- `AddPermission`
- `CreateQueue`
- `CancelMessageMoveTask`
- `DeleteQueue`
- `ListMessageMoveTasks`
- `PurgeQueue`
- `RemovePermission`
- `SetQueueAttributes`
- `StartMessageMoveTask`
- `TagQueue`
- `UntagQueue`

The following Amazon SQS APIs are *not* supported for CloudTrail logging:

- `GetQueueAttributes`
- `GetQueueUrl`
- `ListDeadLetterSourceQueues`
- `ListQueueTags`
- `ListQueues`

Data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource, such as sending or receiving an Amazon SQS message to and from an Amazon SQS queue. Data events are high-volume activities that CloudTrail does not log by default. You can enable data events API action logging for your SQS queue by using CloudTrail APIs. For more information, see Logging data events in the *AWS CloudTrail User Guide*.

With CloudTrail, you can use advanced event selectors to decide which Amazon SQS API activities are logged and recorded. To log Amazon SQS data events, you must include the resource type `AWS::SQS::Queue`. Once this is set, you can refine your logging preferences further by selecting specific data events for recording, such as using the `eventName` filter to track `SendMessage` events. For more information, see `AdvancedEventSelector` in the *AWS CloudTrail API Reference*.

Amazon SQS data events:

- SendMessage
- SendMessageBatch
- ReceiveMessage
- DeleteMessage
- DeleteMessageBatch
- ChangeMessageVisibility
- ChangeMessageVisibilityBatch

Additional charges apply for data events. For more information, see AWS CloudTrail Pricing.

Examples: CloudTrail management events for Amazon SQS

The following examples show CloudTrail log entries for supported APIs:

AddPermission

The following example shows a CloudTrail log entry for an `AddPermission` API call.


```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
         "principalId": "AKIAI44QH8DHBEXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/Alice",
         "accountId": "123456789012",
         "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
         "userName": "Alice"
       },
       "eventTime": "2018-06-28T22:23:46Z",
       "eventSource": "sqs.amazonaws.com",
       "eventName": "AddPermission",
       "awsRegion": "us-east-2",
       "sourceIPAddress": "203.0.113.0",
       "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
 Firefox/24.0",
       "requestParameters": {
```

```
          "actions": [
            "SendMessage"
          ],
          "AWSAccountIds": [
            "123456789012"
          ],
          "label": "MyLabel",
          "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
        },
        "responseElements": null,
        "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
        "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
      }
    ]
  }
```


CreateQueue

The following example shows a CloudTrail log entry for a `CreateQueue` API call.


```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
         "principalId": "AKIAI44QH8DHBEXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/Alejandro",
         "accountId": "123456789012",
         "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
         "userName": "Alejandro"
       },
       "eventTime": "2018-06-28T22:23:46Z",
       "eventSource": "sqs.amazonaws.com",
       "eventName": "CreateQueue",
       "awsRegion": "us-east-2",
       "sourceIPAddress": "203.0.113.1",
       "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
 Firefox/24.0",
       "requestParameters": {
         "queueName": "MyQueue"
       },
       "responseElements": {
         "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
```

```
      },
      "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
      "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
    }
  ]
}
```

DeleteQueue

The following example shows a CloudTrail log entry for a `DeleteQueue` API call.


```
{
  "Records": [
    {
      "eventVersion": "1.06",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Carlos",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Carlos"
      },
      "eventTime": "2018-06-28T22:23:46Z",
      "eventSource": "sqs.amazonaws.com",
      "eventName": "DeleteQueue",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "203.0.113.2",
      "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
 Firefox/24.0",
      "requestParameters": {
        "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
      },
      "responseElements": null,
      "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
      "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
    }
  ]
}
```


RemovePermission

The following example shows a CloudTrail log entry for a `RemovePermission` API call.

```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
         "principalId": "AKIAI44QH8DHBEXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/Jane",
         "accountId": "123456789012",
         "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
         "userName": "Jane"
       },
       "eventTime": "2018-06-28T22:23:46Z",
       "eventSource": "sqs.amazonaws.com",
       "eventName": "RemovePermission",
       "awsRegion": "us-east-2",
       "sourceIPAddress": "203.0.113.3",
       "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
 Firefox/24.0",
       "requestParameters": {
         "label": "label",
         "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
       },
       "responseElements": null,
       "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
       "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
     }
   ]
 }
```


SetQueueAttributes

The following example shows a CloudTrail log entry for `SetQueueAttributes`:


```
{
   "Records": [
     {
       "eventVersion": "1.06",
       "userIdentity": {
         "type": "IAMUser",
         "principalId": "AKIAI44QH8DHBEXAMPLE",
         "arn": "arn:aws:iam::123456789012:user/Maria",
         "accountId": "123456789012",
```

```
          "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
          "userName": "Maria"
        },
        "eventTime": "2018-06-28T22:23:46Z",
        "eventSource": "sqs.amazonaws.com",
        "eventName": "SetQueueAttributes",
        "awsRegion": "us-east-2",
        "sourceIPAddress": "203.0.113.4",
        "userAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:24.0) Gecko/20100101
Firefox/24.0",
        "requestParameters": {
          "attributes": {
            "VisibilityTimeout": "100"
          },
          "queueUrl": "https://sqs.us-east-2.amazon.com/123456789012/MyQueue"
        },
        "responseElements": null,
        "requestID": "123abcde-f4gh-50ij-klmn-60o789012p30",
        "eventID": "0987g654-32f1-09e8-d765-c4f3fb2109fa"
      }
    ]
}
```

Examples: CloudTrail data events for Amazon SQS

The following are examples of CloudTrail events specific to Amazon SQS data event APIs:

SendMessage

The following example shows a CloudTrail data event for `SendMessage`.

```
   {
  "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"
  }
```


ReceiveMessage

The following example shows a CloudTrail data event for `ReceiveMessage`.


```
    {
  "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": "2023-11-07T23:59:24Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "ReceiveMessage",
  "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",
    "maxNumberOfMessages": 10
  },
  "responseElements": null,
  "requestID": "8b4d4643-8f49-52cd-a6e8-1b875ed54b99",
  "eventID": "f3f23ab7-b0a4-4b71-afc0-141209c49206",
  "readOnly": true,
  "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"
  }
}
```


DeleteMessageBatch

The following example shows a CloudTrail data event for `DeleteMessageBatch`.


```
    {
  "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": "2023-11-07T23:59:24Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "DeleteMessageBatch",
  "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",
    "entries": [
      {
        "id": "0",
        "receiptHandle": "AQEBefxM1O4zyZGF87DehbRbmri91w2W7mMdD0GrBjQa8e/
hpb4RbXHPZ9tLBVleECbChQIE5NtaDuoZhZPOkTy0eN46EyRR4jXDzE3AlkbPlX1mA9f2fUuTrXx8aeCoCA3I3woNg3f
hlLS94tjAZqV2krc4BaC2pYgjyHWcW019HwIV8T/bjNMIeZoQwOM5V
+o9vHPfewz5QGr5SKpDo7uE7Umyk5n5CJZvcn1efp/
mrwtaCIb9M7cCQUYcZm2ZmZDnIO9XpGTAi3m2dQ0M83pnNh0nvDfpkHpoa+hX1TrUmxCupCWHJwA8HFJ1O/
CCJsodMNFthLBA9S57dkBZCsw41G8jAmgQ0MkvZ0UL5mg0OFQQd1Yrw0zvthjCgiwdzn0yXoMzxIZMBxkY14E4nVVZ7N
h8oRk2C7gByzg2kYJ0LnUvLJFT8DQE28JZppEC9klvrdR/BWiPT7asc="
      }
    ]
  },
  "responseElements": {
    "successful": [
      {
        "id": "0"
      }
    ],
    "failed": []
  },
  "requestID": "fe423091-5642-5ba5-9256-6d5587de52f1",
  "eventID": "88c8020d-d769-4985-8ecb-ee0b59acc418",
  "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"
  }
}
```


ChangeMessageVisibilityBatch

The following example shows a CloudTrail data event for `ChangeMessageVisibilityBatch`.


```
    {
  "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": "2023-11-07T23:59:01Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "ChangeMessageVisibilityBatch",
  "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": {
    "visibilityTimeout": 0,
    "entries": [
      {
        "id": "0",
        "receiptHandle":
 "AQEB2M5cVYg5gslhWME6537hdjcaPnOYPA5M0W460TTb0DzPle631yPWm8qxd4O1hDj/
B4ntTMnsgBTa95t14tNx7Vn96jKJ5rIoZ7iI8TRmkT1caKodKIPs8w9yndZq50c2FPQxtyH+2L3UHf/
abV3szqVWXOLZR4PwX8zZkWVQGNCNnY2q2lGCG586F8QwvrOFYoXNwB8ymd1t77e1PDPknq1Io3JFuzkEsndkkETy4fV
l5PHX17nXxaC+DURVlMPXOuSFACGmWqAoyk50HKwGOjLQgpySL/
TcnQXClvFq8kNXGwyVzJsbwHpOHxI7oce69vaD6DaWFP75d3hx+PJeG9pauQCKzVP3skt3Hw/
zDC7YfKcALD3aCwMmeNDwT3w0BUG6XZdG5lYhtFtTQYV7YuS3i/
Jh3HShGbtm07JKOEFiPkxv2+XNaAX3gFEpbng6zamTanfyMXCJIiglAEqiyWHQ=",
        "visibilityTimeout": 2271
      }
    ],
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue"
  },
  "responseElements": {
    "successful": [
      {
        "id": "0"
      }
    ]
  },
  "requestID": "d49ab65f-9dc7-54b8-875c-eb9b4c42988b",
  "eventID": "ca16c8c2-c4ba-4eb5-a54c-e650a10266d4",
  "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"
  }
}
```

Monitoring Amazon SQS queues using CloudWatch

Amazon SQS and Amazon CloudWatch are integrated so you can use CloudWatch to view and analyze metrics for your Amazon SQS queues. You can view and analyze your queues' metrics from the Amazon SQS console, the CloudWatch console, using the AWS CLI, or using the CloudWatch API. You can also set CloudWatch alarms for Amazon SQS metrics.

CloudWatch metrics for your Amazon SQS queues are automatically collected and pushed to CloudWatch at one-minute intervals. These metrics are gathered on all queues that meet the CloudWatch guidelines for being *active*. CloudWatch considers a queue to be active for up to six hours if it contains any messages, or if any action accesses it.

When an Amazon SQS queue is inactive for more than six hours, the Amazon SQS service is considered asleep and stops delivering metrics to the CloudWatch service. Missing data, or data representing zero, can't be visualized in the CloudWatch metrics for Amazon SQS for the time period that your Amazon SQS queue was inactive.

Note

- An Amazon SQS queue can be activated when the user calling an API against the queue is not authorized, and the request fails.
- The Amazon SQS console performs a `GetQueueAttributes` API call when the queue's page is opened. The `GetQueueAttributes` API request activates the queue.
- A delay of up to 15 minutes occurs in CloudWatch metrics when a queue is activated from an inactive state.

- There is no charge for the Amazon SQS metrics reported in CloudWatch. They're provided as part of the Amazon SQS service.
- CloudWatch metrics are supported for both standard and FIFO queues.

Topics

Accessing CloudWatch metrics for Amazon SQS

Amazon SQS and Amazon CloudWatch are integrated so you can use CloudWatch to view and analyze metrics for your Amazon SQS queues. You can view and analyze your queues' metrics from the Amazon SQS console, the CloudWatch console, using the AWS CLI, or using the CloudWatch API. You can also set CloudWatch alarms for Amazon SQS metrics.

Amazon SQS console

1. Sign in to the Amazon SQS console.
2. In the list of queues, choose (check) the boxes for the queues that you want to access metrics for. You can show metrics for up to 10 queues.
3. Choose the **Monitoring** tab.

   Various graphs are displayed in the **SQS metrics** section.
4. To understand what a particular graph represents, hover over

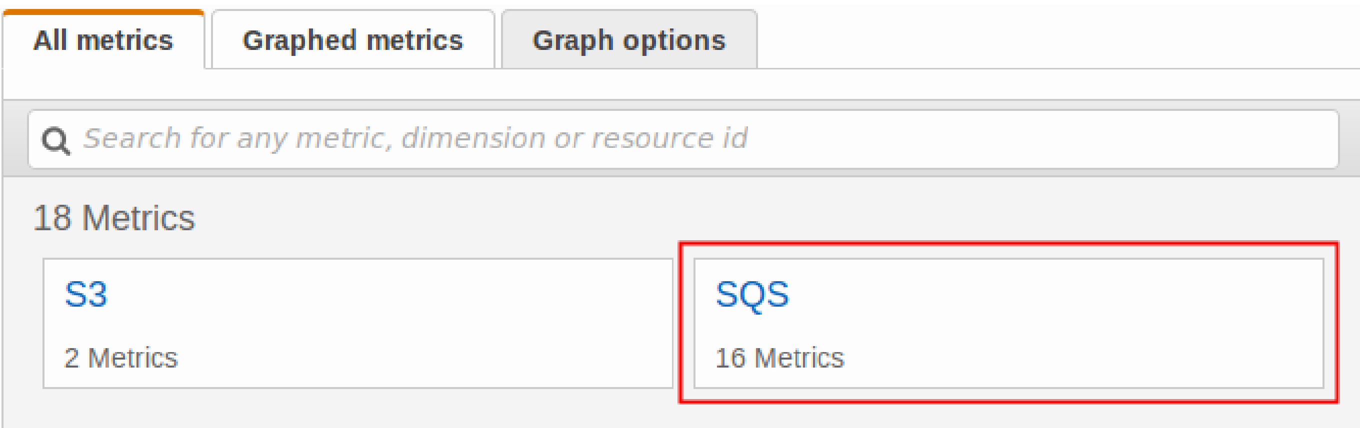
   next to the desired graph, or see Available CloudWatch metrics for Amazon SQS.
5. To change the time range for all of the graphs at the same time, for **Time Range**, choose the desired time range (for example, **Last Hour**).
6. To view additional statistics for an individual graph, choose the graph.
7. In the **CloudWatch Monitoring Details** dialog box, select a **Statistic**, (for example, **Sum**). For a list of supported statistics, see Available CloudWatch metrics for Amazon SQS.
8. To change the time range and time interval that an individual graph displays (for example, to show a time range of the last 24 hours instead of the last 5 minutes, or to show a time

period of every hour instead of every 5 minutes), with the graph's dialog box still displayed, for **Time Range**, choose the desired time range (for example, **Last 24 Hours**). For **Period**, choose the desired time period within the specified time range (for example, **1 Hour**). When you're finished looking at the graph, choose **Close**.

9. (Optional) To work with additional CloudWatch features, on the **Monitoring** tab, choose **View all CloudWatch metrics**, and then follow the instructions in the Amazon CloudWatch console procedure.

Amazon CloudWatch console

1. Sign in to the CloudWatch console.
2. On the navigation panel, choose **Metrics**.
3. Select the **SQS** metric namespace.

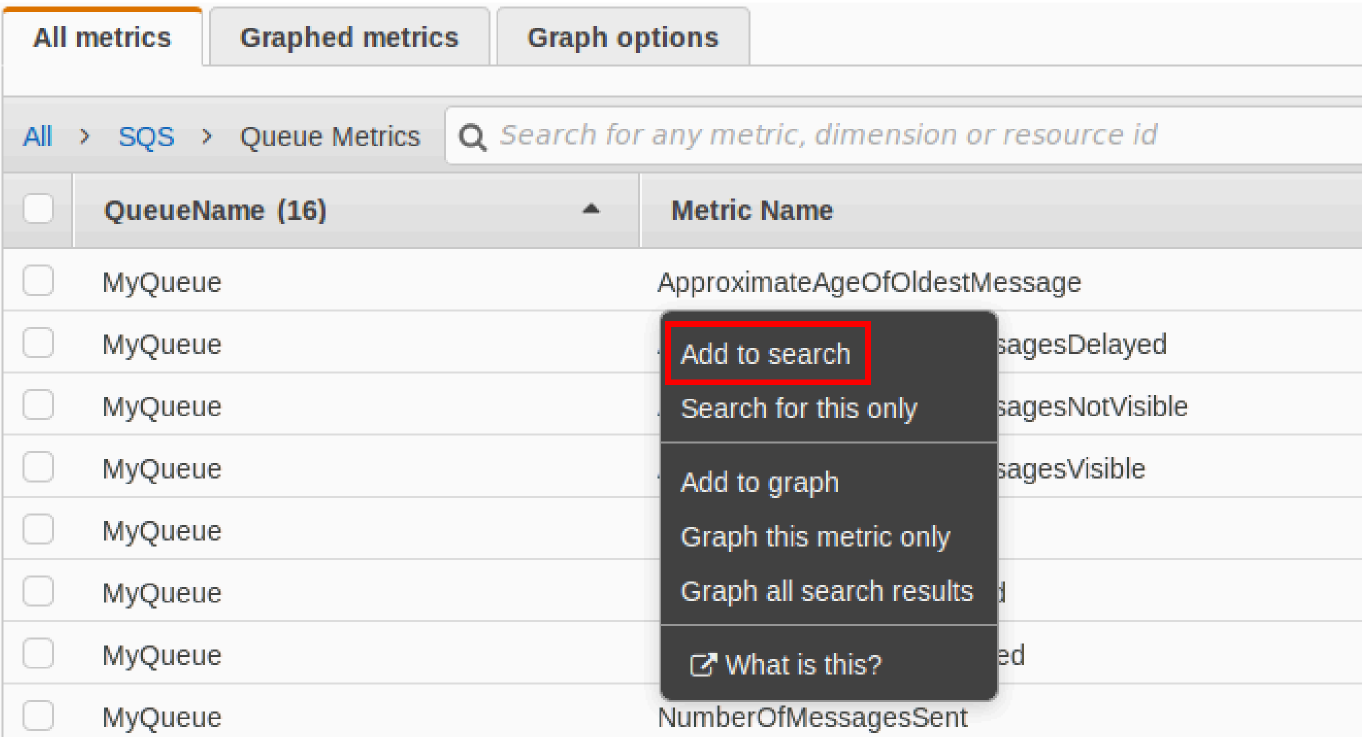


4. Select the **Queue Metrics** metric dimension.

5. You can now examine your Amazon SQS metrics:

   - To sort the metrics, use the column heading.

- To graph a metric, select the check box next to the metric.
- To filter by metric, choose the metric name and then choose **Add to search**.



For more information and additional options, see [Graph Metrics]() and [Using Amazon CloudWatch Dashboards]() in the *Amazon CloudWatch User Guide*.

AWS Command Line Interface

To access Amazon SQS metrics using the AWS CLI, run the `get-metric-statistics` command.

For more information, see Get Statistics for a Metric in the *Amazon CloudWatch User Guide*.

CloudWatch API

To access Amazon SQS metrics using the CloudWatch API, use the `GetMetricStatistics` action.

For more information, see Get Statistics for a Metric in the *Amazon CloudWatch User Guide*.

Creating CloudWatch alarms for Amazon SQS metrics

CloudWatch lets you trigger alarms based on a metric threshold. For example, you can create an alarm for the `NumberOfMessagesSent` metric. For example, if more than 100 messages are sent to the `MyQueue` queue in 1 hour, an email notification is sent out. For more information, see [Creating Amazon CloudWatch Alarms](#) in the *Amazon CloudWatch User Guide*.

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. Choose **Alarms**, and then choose **Create Alarm**.
3. In the **Select Metric** section of the **Create Alarm** dialog box, choose **Browse Metrics**, **SQS**.
4. For **SQS > Queue Metrics**, choose the **QueueName** and **Metric Name** for which to set an alarm, and then choose **Next**. For a list of available metrics, see Available CloudWatch metrics for Amazon SQS.

   In the following example, the selection is for an alarm for the `NumberOfMessagesSent` metric for the `MyQueue` queue. The alarm triggers when the number of sent messages exceeds 100.
5. In the **Define Alarm** section of the **Create Alarm** dialog box, do the following:

   a. Under **Alarm Threshold**, type the **Name** and **Description** for the alarm.

   b. Set **is** to **> 100**.

   c. Set **for** to **1 out of 1 datapoints**.

   d. Under **Alarm preview**, set **Period** to **1 Hour**.

   e. Set **Statistic** to **Standard**, **Sum**.

   f. Under **Actions**, set **Whenever this alarm** to **State is ALARM**.

      If you want CloudWatch to send a notification when the alarm is triggered, select an existing Amazon SNS topic or choose **New list** and enter email addresses separated by commas.

      > **Note**
      >
      > If you create a new Amazon SNS topic, the email addresses must be verified before they receive any notifications. If the alarm state changes before the email addresses are verified, the notifications aren't delivered.
6. Choose **Create Alarm**.

   The alarm is created.

Available CloudWatch metrics for Amazon SQS

Amazon SQS sends the following metrics to CloudWatch.

> **Note**
>
> For some metrics, the result is approximate because of the distributed architecture of Amazon SQS. In most cases, the count should be close to the actual number of messages in the queue.

Amazon SQS metrics

The `AWS/SQS` namespace includes the following metrics.

Metric	Description
`ApproximateAgeOfOldestMessage`	The approximate age of the oldest non-deleted message in the queue. **Note** • After a message is received three times (or more) and not processed, the message is moved to the back of the queue and the `ApproximateAgeOfOldestMessage` metric points at the second-oldest message that hasn't been received more than three times. This action occurs even if the queue has a redrive policy. • Because a single poison-pill message (received multiple

Metric	Description
	times but never deleted) can distort this metric, the age of a poison-pill message isn't included in the metric until the poison-pill message is consumed successfully. • When the queue has a redrive policy, the message is moved to a dead-letter queue after the configured maximum number of receives. When the message is moved to the dead-letter queue, the `ApproximateAgeOfOldestMessage` metric of the dead-letter queue represents the time when the message was moved to the dead-letter queue (not the original time the message was sent). • For FIFO queues, the message is not moved to the back of the queue because this will break the FIFO order guarantee. The message will instead go to the DLQ if there is one configured. Otherwise it will block the message group until successfully deleted or until it expires.

Metric	Description
	Reporting criteria: A non-negative value is reported if the queue is active. **Units:** Seconds **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)
`ApproximateNumberOfGroupsWithInflightMessages`	The approximate number of message groups with in flight messages, where a message is considered to be *in flight* after it's received from a queue by a consumer, but not yet deleted from the queue. This metric can help you troubleshoot and optimize your FIFO queue throughput by either increasing FIFO message groups, or scaling your consumers. **Reporting criteria:** A non-negative value is reported if the queue is active. **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console) For current FIFO throughput and in flight limits, see Amazon SQS message quotas.

Metric	Description
`ApproximateNumberOfMessagesDelayed`	The number of messages in the queue that are delayed and not available for reading immediately. This can happen when the queue is configured as a delay queue or when a message has been sent with a delay parameter. **Reporting criteria:** A non-negative value is reported [if the queue is active](#). **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)
`ApproximateNumberOfMessagesNotVisible`	The number of messages that are in flight. Messages are considered to be *in flight* if they have been sent to a client but have not yet been deleted or have not yet reached the end of their visibility window. **Reporting criteria:** A non-negative value is reported [if the queue is active](#). **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)

Metric	Description
`ApproximateNumberOfMessagesVisible`	The number of messages to be processed . **Reporting criteria:** A non-negative value is reported [if the queue is active](#). **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console) There is no limit on the number of messages to processes, however you can subject this backlog to a retention period.
`NumberOfEmptyReceives` [1]	The number of `ReceiveMessage` API calls that did not return a message. **Reporting criteria:** A non-negative value is reported [if the queue is active](#). **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)

Metric	Description
`NumberOfDeduplicatedSentMessages`	The number of messages sent to a queue that were deduplicated. This metric can help determine if a producer is sending duplicate messages to an Amazon SQS FIFO queue. **Reporting criteria:** A non-negative value is reported if the queue is active. **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)

Metric	Description
`NumberOfMessagesDeleted` [1]	The number of messages deleted from the queue. **Reporting criteria:** A non-negative value is reported if the queue is active. **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console) Amazon SQS emits the `NumberOfMessagesDeleted` metric for every successful deletion operation that uses a valid receipt handle, including duplicate deletions. The following scenarios might cause the value of the `NumberOfMessagesDeleted` metric to be higher than expected: • Calling the `DeleteMessage` action on different receipt handles that belong to the same message: If the message is not processed before the visibility timeout expires, the message becomes available to other consumers that can process it and delete it again, increasing the value of the `NumberOfMessagesDeleted` metric. • Calling the `DeleteMessage` action on the same receipt handle: If the

Metric	Description
	message is processed and deleted but you call the `DeleteMessage` action again using the same receipt handle, a success status is returned, increasing the value of the `NumberOfMessagesDeleted` metric.
`NumberOfMessagesReceived` [1]	The number of messages returned by calls to the `ReceiveMessage` action. **Reporting criteria:** A non-negative value is reported if the queue is active. **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)

Metric	Description
`NumberOfMessagesSent` [1]	The number of messages added to a queue. If you send a message to a dead-letter queue manually, it is captured by the `NumberOfMessagesSent` metric. However, if a message is sent to a dead-letter queue as a result of a failed processing attempt, it is not captured by this metric. Thus, it is possible for the values of `NumberOfMessagesSent` and `NumberOfMessagesReceived` to be different. **Reporting criteria:** A non-negative value is reported if the queue is active. **Units:** Count **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)

Metric	Description
`SentMessageSize` [1]	The size of messages added to a queue. **Reporting criteria:** A non-negative value is reported [if the queue is active](#). **Units:** Bytes **Valid statistics:** Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console) **Note** `SentMessageSize` does not display as an available metric in the CloudWatch console until at least one message is sent to the corresponding queue.

[1] These metrics are calculated from a service perspective, and can include retries. Don't rely on the absolute values of these metrics, or use them to estimate current queue status.

Dimensions for Amazon SQS metrics

The only dimension that Amazon SQS sends to CloudWatch is `QueueName`. This means that all available statistics are filtered by `QueueName`.

Compliance validation for Amazon SQS

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- Architecting for HIPAA Security and Compliance on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

> **Note**
>
> Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- Evaluating Resources with Rules in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- Amazon GuardDuty – This AWS service detects potential threats to your AWS accounts, workloads, containers, and data by monitoring your environment for suspicious and malicious activities. GuardDuty can help you address various compliance requirements, like PCI DSS, by meeting intrusion detection requirements mandated by certain compliance frameworks.

- [AWS Audit Manager](#) – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in Amazon SQS

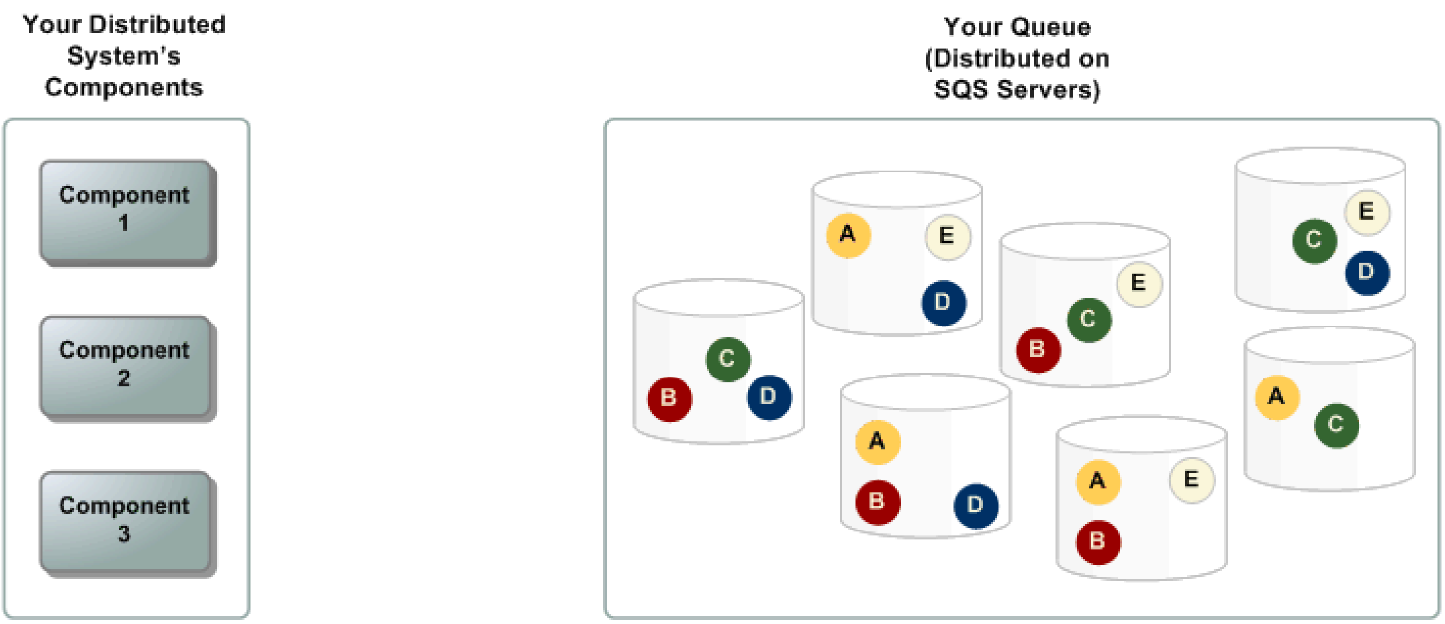
The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures. For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, Amazon SQS offers distributed queues.

Distributed queues

There are three main parts in a distributed messaging system: the components of your distributed system, your queue (distributed on Amazon SQS servers), and the messages in the queue.

In the following scenario, your system has several *producers* (components that send messages to the queue) and *consumers* (components that receive messages from the queue). The queue (which holds messages A through E) redundantly stores the messages across multiple Amazon SQS servers.



Infrastructure security in Amazon SQS

As a managed service, Amazon SQS is protected by the AWS global network security procedures described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API actions to access Amazon SQS through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with Perfect Forward Secrecy (PFS), such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE).

You must sign requests using an access key ID and a secret access key associated with an IAM principal. Alternatively, you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials for signing requests.

You can call these API actions from any network location, but Amazon SQS supports resource-based access policies, which can include restrictions based on the source IP address. You can also use Amazon SQS policies to control access from specific Amazon VPC endpoints or specific VPCs. This effectively isolates network access to a given Amazon SQS queue from only the specific VPC within the AWS network. For more information, see Example 5: Deny access if it isn't from a VPC endpoint.

Amazon SQS security best practices

AWS provides many security features for Amazon SQS, which you should review in the context of your own security policy. The following are preventative security best practices for Amazon SQS.

> **Note**
>
> The specific implementation guidance provided is for common use cases and implementations. We suggest that you view these best practices in the context of your specific use case, architecture, and threat model.

Topics

- Implement server-side encryption
- Enforce encryption of data in transit
- Consider using VPC endpoints to access Amazon SQS

Make sure that queues aren't publicly accessible

Unless you explicitly require anyone on the internet to be able to read or write to your Amazon SQS queue, you should make sure that your queue isn't publicly accessible (accessible by everyone in the world or by any authenticated AWS user).

- Avoid creating policies with `Principal` set to `""`.
- Avoid using a wildcard (*). Instead, name a specific user or users.

Implement least-privilege access

When you grant permissions, you decide who receives them, which queues the permissions are for, and specific API actions that you want to allow for these queues. Implementing least privilege is important to reducing security risks and reducing the effect of errors or malicious intent.

Follow the standard security advice of granting least privilege. That is, grant only the permissions required to perform a specific task. You can implement this using a combination of security policies.

Amazon SQS uses the producer-consumer model, requiring three types of user account access:

- **Administrators** – Access to creating, modifying, and deleting queues. Administrators also control queue policies.
- **Producers** – Access to sending messages to queues.
- **Consumers** – Access to receiving and deleting messages from queues.

For more information, see the following sections:

- Identity and access management in Amazon SQS
- Amazon SQS API permissions: Actions and resource reference
- Using custom policies with the Amazon SQS Access Policy Language

Use IAM roles for applications and AWS services which require Amazon SQS access

For applications or AWS services such as Amazon EC2 to access Amazon SQS queues, they must use valid AWS credentials in their AWS API requests. Because these credentials aren't rotated automatically, you shouldn't store AWS credentials directly in the application or EC2 instance.

You should use an IAM role to manage temporary credentials for applications or services that need to access Amazon SQS. When you use a role, you don't have to distribute long-term credentials (such as a username, password, and access keys) to an EC2 instance or AWS service such as AWS Lambda. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources.

For more information, see IAM Roles and Common Scenarios for Roles: Users, Applications, and Services in the *IAM User Guide*.

Implement server-side encryption

To mitigate data leakage issues, use encryption at rest to encrypt your messages using a key stored in a different location from the location that stores your messages. Server-side encryption (SSE) provides data encryption at rest. Amazon SQS encrypts your data at the message level when it stores it, and decrypts the messages for you when you access them. SSE uses keys managed in AWS Key Management Service. As long as you authenticate your request and have access permissions, there is no difference between accessing encrypted and unencrypted queues.

For more information, see Encryption at rest in Amazon SQS and Amazon SQS Key management.

Enforce encryption of data in transit

Without HTTPS (TLS), a network-based attacker can eavesdrop on network traffic or manipulate it, using an attack such as man-in-the-middle. Allow only encrypted connections over HTTPS (TLS) using the `aws:SecureTransport` condition in the queue policy to force requests to use SSL.

Consider using VPC endpoints to access Amazon SQS

If you have queues that you must be able to interact with but which must absolutely not be exposed to the internet, use VPC endpoints to queue access to only the hosts within a particular VPC. You can use queue policies to control access to queues from specific Amazon VPC endpoints or from specific VPCs.

Amazon SQS VPC endpoints provide two ways to control access to your messages:

- You can control the requests, users, or groups that are allowed through a specific VPC endpoint.
- You can control which VPCs or VPC endpoints have access to your queue using a queue policy.

For more information, see Amazon Virtual Private Cloud endpoints for Amazon SQS and Creating an Amazon VPC endpoint policy for Amazon SQS.

Related Amazon SQS resources

The following table lists related resources that you might find useful as you work with this service.

Resource	Description
Amazon Simple Queue Service API Reference	Descriptions of actions, parameters, and data types and a list of errors that the service returns.
Amazon SQS in the *AWS CLI Command Reference*	Descriptions of the AWS CLI commands that you can use to work with queues.
Regions and Endpoints	Information about Amazon SQS regions and endpoints
Product Page	The primary web page for information about Amazon SQS.
Discussion Forum	A community-based forum for developers to discuss technical questions related to Amazon SQS.
AWS Premium Support Information	The primary web page for information about AWS Premium Support, a one-on-one, fast-response support channel to help you build and run applications on AWS infrastructure services.

Documentation history

The following table describes the important changes to the *Amazon Simple Queue Service Developer Guide* since Jan 2019. For notifications about updates to this documentation, subscribe to the RSS feed.

Service features are sometimes rolled out incrementally to the AWS Regions where a service is available. We update this documentation for the first release only. We don't provide information about Region availability or announce subsequent Region rollouts. For information about Region availability of service features and to subscribe to notifications about updates, see What's New with AWS?.

Change	Description	Date
FIFO metrics update	Support for `NumberOfDeduplicatedSentMessages` and `ApproximateNumberOfGroupsWithInflightMessages` added to Amazon SQS FIFO metrics.	July 3, 2024
`ListQueueTags` action supported in the AmazonSQSReadOnlyAccess managed policy	The AmazonSQSReadOnlyAccess managed policy supports `ListQueueTags` to retrieve all tags associated with a specified Amazon SQS queue.	May 2, 2024
AWS JSON protocol	Make API requests using AWS JSON protocol.	July 27, 2023
New section describing AWS managed policies for Amazon SQS and updates to these policies	Amazon SQS added a new action that allows you to list the most recent message movement tasks (up to 10) under a specific source queue.	June 7, 2023

	This action is associated with the `ListMessageMoveTasks` API operation.	
Dead-letter queue redrive using APIs	Configure dead-letter queue redrives using Amazon SQS APIs.	June 7, 2023
ABAC for Amazon SQS	Attribute-based access control (ABAC) using queue tags for flexible and scalable access permissions.	November 10, 2022
FIFO high throughput limit increases	Increased default quotas for FIFO high throughput mode in commercial Regions, plus FIFO high throughput document optimization.	October 20, 2022
Default server-side encryption (SSE) is available	Server-side encryption (SSE) using SQS-owned encryption (SSE-SQS) by default.	September 26, 2022
Amazon SQS confused deputy protection support is available	Confused deputy protection allows you to specify new headers in their requests, which are checked against conditions in the KMS policy when using Amazon SQS managed SSE.	December 29, 2021
Managed SSE is available	Amazon SQS managed SSE (SSE-SQS) is managed server-side encryption that uses Amazon SQS-owned encryption keys to protect sensitive data sent over message queues.	November 23, 2021

Dead-letter queue redrive is available	Amazon SQS supports dead-letter queue redrive for standard queues.	November 10, 2021
High throughput for messages in FIFO queues is available	High throughput for Amazon SQS FIFO queues provides a higher number of transactions per second (TPS) for messages in FIFO queues. For information on throughput quotas, see Quotas related to messages.	May 27, 2021
High throughput for messages in FIFO queues is available in preview release	High throughput for Amazon SQS FIFO queues is in preview release and is subject to change. This feature provides a higher number of transactions per second (TPS) for messages in FIFO queues. For information on throughput quotas, see Quotas related to messages.	December 17, 2020
New Amazon SQS console design	To simplify development and production workflows, the Amazon SQS console has a new user experience.	July 8, 2020
Amazon SQS supports pagination for listQueues and listDeadLetterSourceQueues	You can specify the maximum number of results to return from a listQueues or listDeadLetterSourceQueues request.	June 22, 2020

Amazon SQS supports 1-minute Amazon CloudWatch metrics in all AWS Regions, except the AWS GovCloud (US) Regions	The one-minute CloudWatch metric for Amazon SQS is available in all Regions, except the AWS GovCloud (US) Regions.	January 9, 2020
Amazon SQS supports 1-minute CloudWatch metrics	The one-minute CloudWatch metric for Amazon SQS is currently available only in the following Regions: US East (Ohio), Europe (Ireland), Europe (Stockholm), and Asia Pacific (Tokyo).	November 25, 2019
AWS Lambda triggers for Amazon SQS FIFO queues are available	You can configure messages arriving in a FIFO queue as a Lambda function trigger.	November 25, 2019
Server-side encryption (SSE) for Amazon SQS is available in the China Regions	SSE for Amazon SQS is available in the China Regions.	November 13, 2019
FIFO queues are available in the Middle East (Bahrain) Region	FIFO queues are available in the Middle East (Bahrain) Region.	October 10, 2019
Amazon Virtual Private Cloud (Amazon VPC) endpoints for Amazon SQS are available in the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions	You can send messages to your Amazon SQS queues from Amazon VPC in the AWS GovCloud (US-East) and AWS GovCloud (US-West) Regions.	September 5, 2019

Amazon SQS allows troubleshooting of queues using AWS X-Ray using message system attributes	You can troubleshoot messages passing through Amazon SQS queues using X-Ray. This release adds the `MessageSystemAttribute` request parameter (which lets you send X-Ray trace headers through Amazon SQS) to the `SendMessage` and `SendMessageBatch` API operations, the `AWSTraceHeader` attribute to the `ReceiveMessage` API operation, and the `MessageSystemAttributeValue` data type.	August 28, 2019
You can tag Amazon SQS queues upon creation	You can use a single Amazon SQS API call, AWS SDK function, or AWS Command Line Interface (AWS CLI) command to simultaneously create a queue and specify its tags. In addition, Amazon SQS supports the `aws:TagKeys` and `aws:RequestTag` AWS Identity and Access Management (IAM) keys.	August 22, 2019

The temporary queue client for Amazon SQS is now available	Temporary queues help you save development time and deployment costs when using common message patterns such as *request-response*. You can use the Temporary Queue Client to create high-throughput, cost-effe ctive, application-managed temporary queues.	July 25, 2019
SSE for Amazon SQS is available in the AWS GovCloud (US-East) Region	Server-side encryption (SSE) for Amazon SQS is available in the AWS GovCloud (US-East) Region.	June 20, 2019
FIFO queues are available in the Asia Pacific (Hong Kong), China (Beijing), AWS GovCloud (US-East), and AWS GovCloud (US-West) Regions	FIFO queues are available in the Asia Pacific (Hong Kong), China (Beijing), AWS GovCloud (US-East), and AWS GovCloud (US-West) Regions.	May 15, 2019
Amazon VPC endpoint policies are available for Amazon SQS	You can create Amazon VPC endpoint policies for Amazon SQS.	April 4, 2019
FIFO queues are available in the Europe (Stockholm) and China (Ningxia) Regions	FIFO queues are available in the Europe (Stockholm) and China (Ningxia) Regions.	March 14, 2019

FIFO queues are available in all Regions where Amazon SQS is available	FIFO queues are available in the US East (Ohio), US East (N. Virginia), US West (N. California), US West (Oregon), Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), Canada (Central), Europe (Frankfurt), Europe (Ireland) , Europe (London), Europe (Paris), and South America (São Paulo) Regions.	February 7, 2019