aws

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.

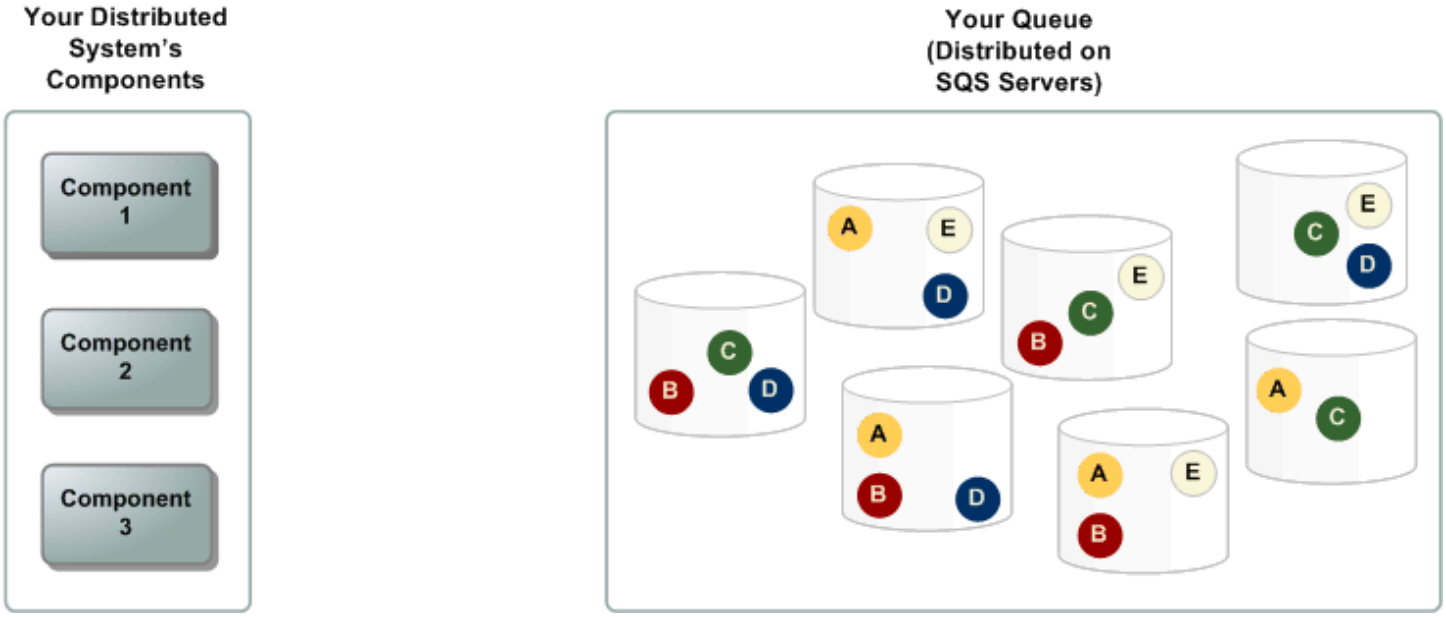
Basic Amazon SQS architecture

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

Distributed queues

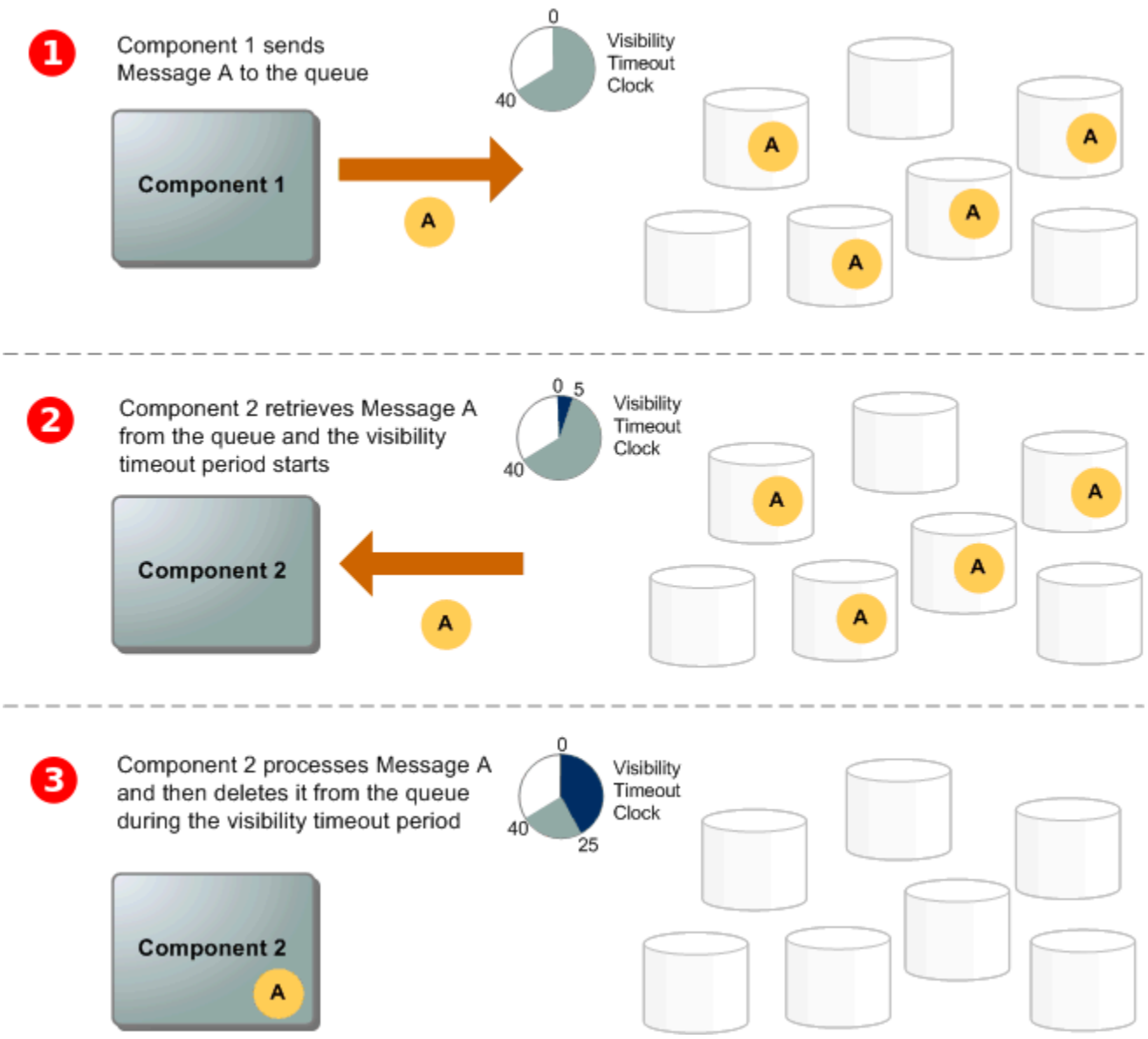
There are three main parts in a distributed messaging system: the components of your distributed system, your queue (distributed on Amazon SQS servers), and the messages in the queue.

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 are managed messaging services that are highly scalable and simple to use. The following is an overview of the differences between these services:

Amazon SQS offers hosted queues that integrate and decouple distributed software systems and components. Amazon SQS provides a generic web services API that you can access using any programming language supported by AWS SDK. Messages in the queue are typically processed by a single subscriber. Amazon SQS and Amazon SNS are often used together to create a fanout messaging application.

Amazon SNS is a publish-subscribe service that provides message delivery from publishers (also known as producers) to multiple subscriber endpoints(also known as consumers). Publishers communicate asynchronously with subscribers by sending messages to a topic, which is a logical access point and communication channel. Subscribers can subscribe to an Amazon SNS topic and 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). Amazon SNS acts as a message router and delivers messages to subscribers in real time. If a subscriber is not available at the time of message publication, the message is not stored for later retrieval.

Amazon MQis a managed message broker service that provides compatibility with industry standard messaging protocols such as Advanced Message Queueing Protocol (AMQP) and Message Queuing Telemetry Transport (MQTT). Amazon MQ currently supports Apache ActiveMQ and RabbitMQ engine types.

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

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. 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 an administrative 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 an administrative user

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 an administrative user

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 an administrative 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 administrative user

- 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*.

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 (Users managed in IAM Identity Center)	Use temporary 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 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

Which user needs programmatic access?	To	By
		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*.
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 for developing and managing applications 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

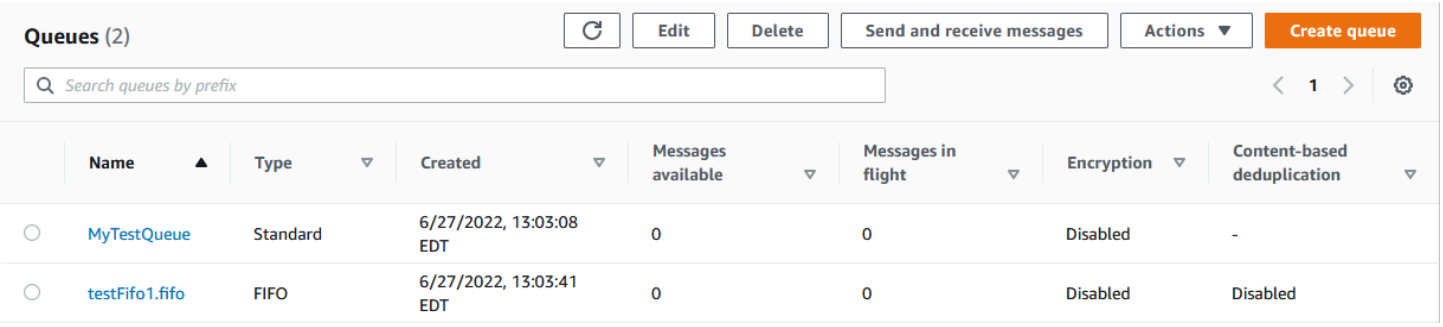
In this section, you will learn how to create standard or FIFO queues using the Amazon SQS console.

Prerequisites

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

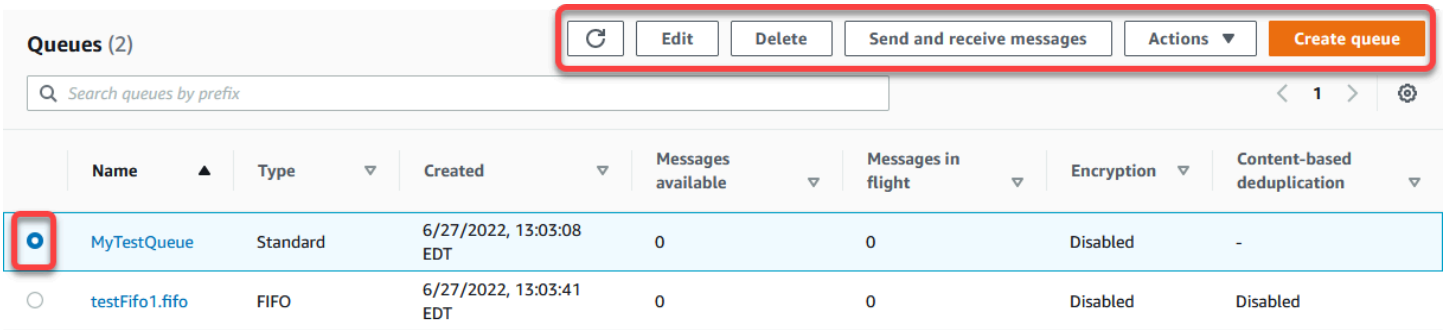
Understanding the Amazon SQS console

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.

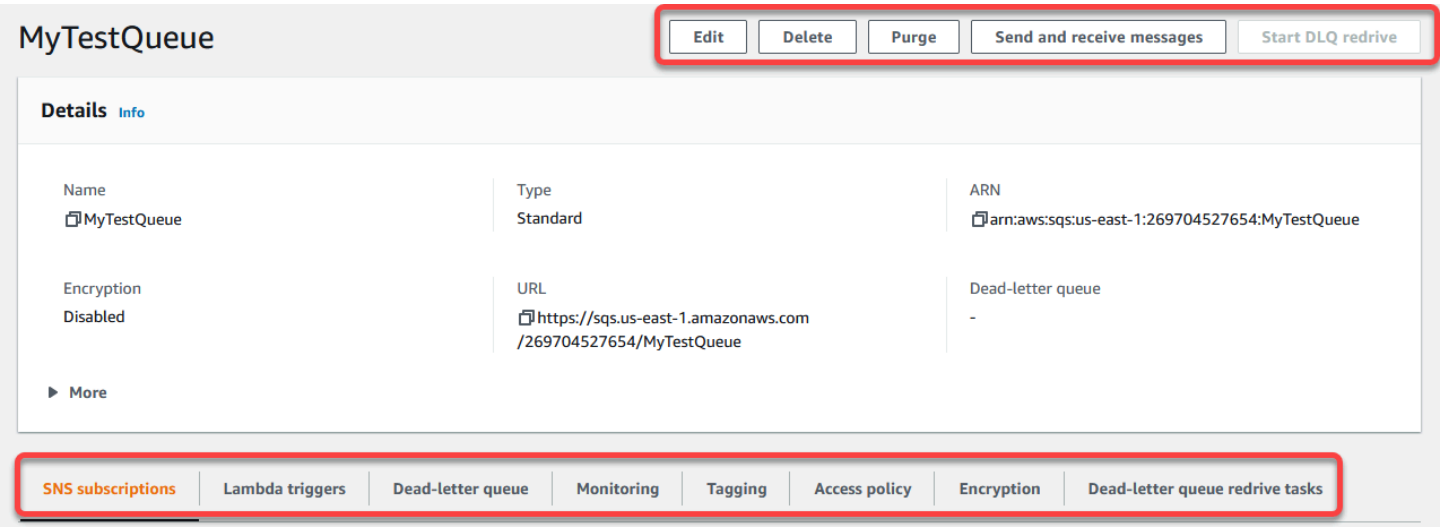


The entry for each queue shows the queue type and other information about the queue. The **Type** column helps you distinguish standard queues from First-In-First Out (FIFO) queues at a glance.

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.



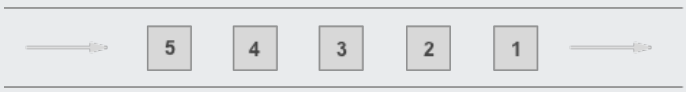
You can also choose the queue name, which open the **Details** page for the queue. The **Details** page includes the same actions as the **Queues** page. In addition, you can choose one of the tabs below the **Details** section to view additional configuration details and actions.



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 Getting started with Amazon SQS standard queues and Getting started with 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`).	**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`).
At-Least-Once Delivery – A message is delivered at least once, but occasionally more than one copy of a message is delivered.	**Exactly-Once Processing** – A message is delivered once and remains available until a
Best-Effort Ordering – Occasionally, messages are delivered in an order different from which they were sent.	

Standard queues	FIFO queues
	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. • 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.	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. • Prevent a student from enrolling in a course before registering for an account.

Creating an Amazon SQS standard queue and sending a message

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

Create a queue (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 (SSE) for a queue using SQS-managed encryption keys (console) and Configuring server-side encryption (SSE) for a queue (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 and Quotas related to messages.

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 (SSE) for a queue using SQS-managed encryption keys (console) and Configuring server-side encryption (SSE) for a queue (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 delivery logic.
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 delivery logic.
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.

Managing an Amazon SQS queue

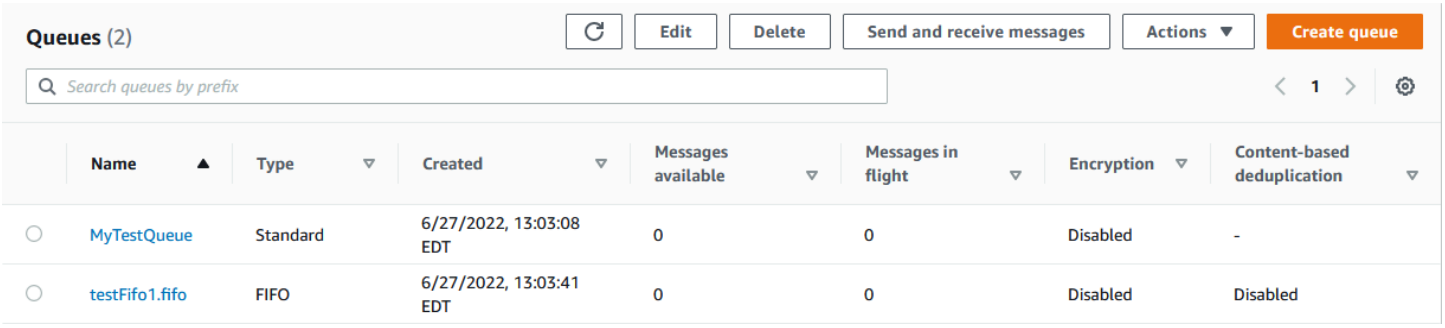
This section helps you become more familiar with Amazon SQS by showing you how to manage queues and messages using the Amazon SQS console.

Prerequisites

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

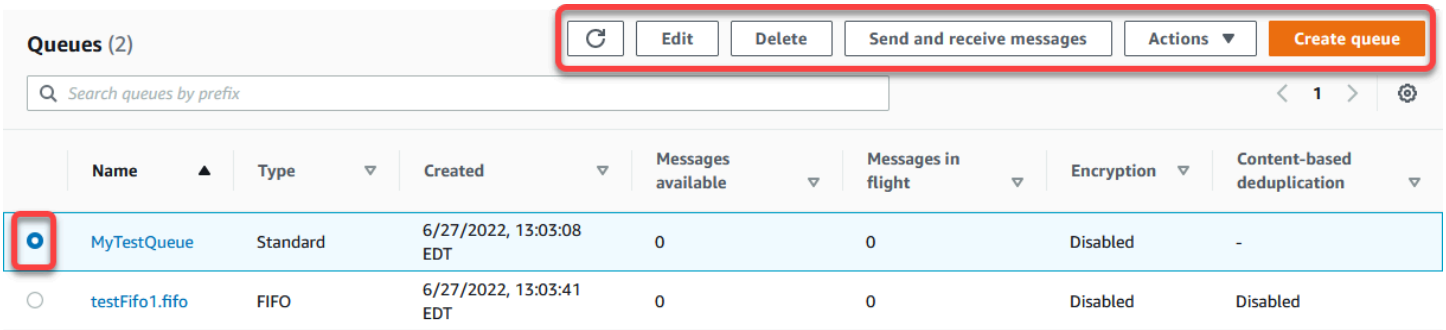
Understanding the Amazon SQS console

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.

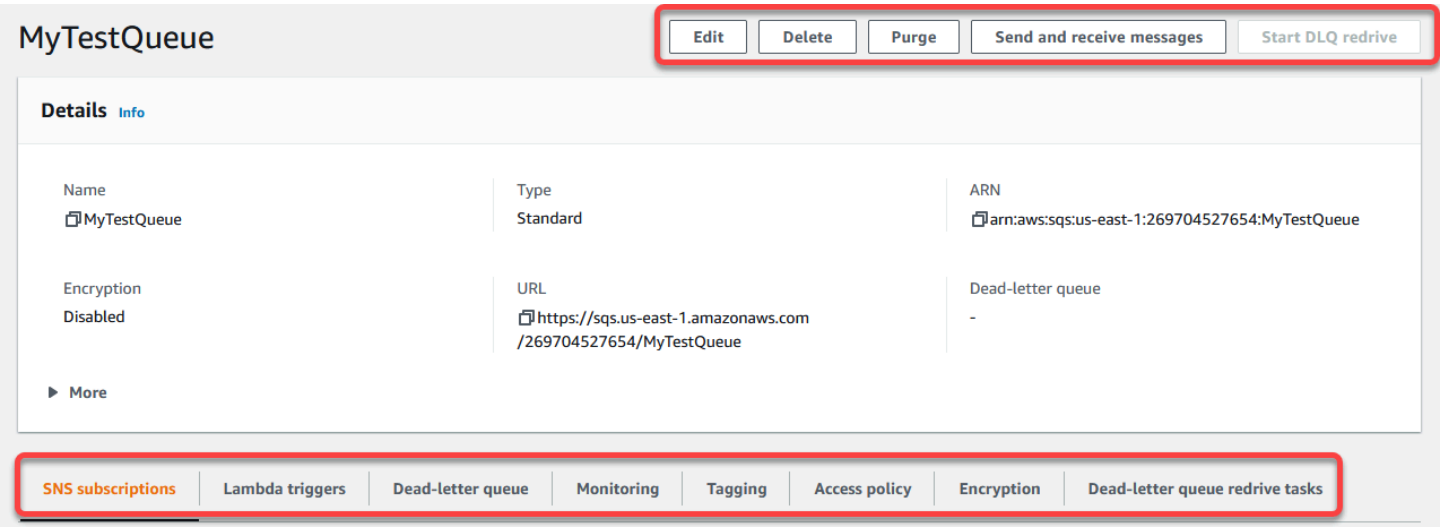


The entry for each queue shows the queue type and other information about the queue. The **Type** column helps you distinguish standard queues from First-In-First Out (FIFO) queues at a glance.

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.



Edit a queue (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.

Receive and delete a message (console)

After you send messages to a queue, you can receive and delete them. When you request messages from a queue, you cannot specify which messages to retrieve. Instead, you specify the maximum number of messages (up to 10) that you want to retrieve.

> **Note**
>
> Because Amazon SQS is a distributed system, a queue with very few messages might display an empty response to a receive request. In this case, rerun the request to get your message. Depending on your application's needs, you might have to use short or long polling to receive messages.

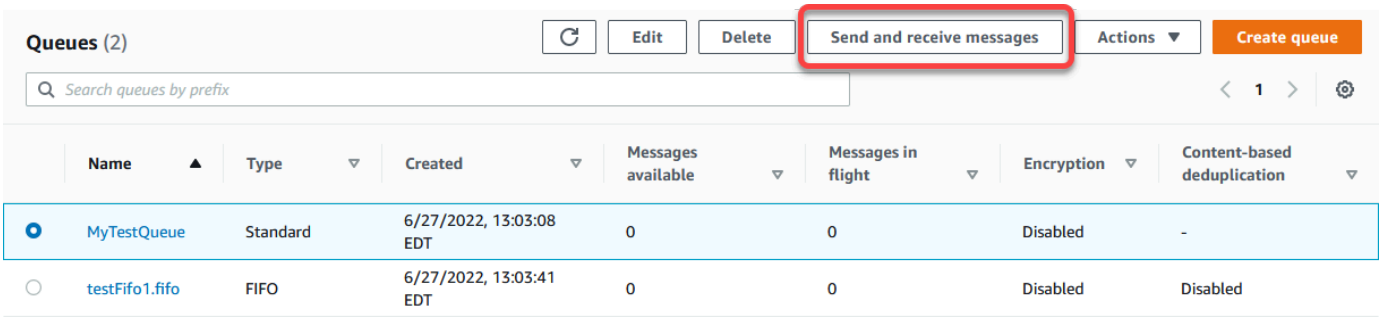
Amazon SQS doesn't automatically delete a message after retrieving it for you, in case you don't successfully receive the message (for example, if the consumers fail or you lose connectivity). To delete a message, you must send a separate request which acknowledges that you've successfully received and processed the message. Note that you must receive a message before you can delete it.

> **Note**
>
> After receiving messages from the Amazon SQS console, the console immediately sets the messages back to visible, so that the messages can be received again.

For more information on API options for recieving and deleting messages, see the Amazon SQS API Reference Guide.

To receive and delete a message (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**.



The console displays the **Send and receive messages** page.

5. 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.

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

Confirming that a 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.

Delete a 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 a 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 CLIAWS 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 Amazon SQS queue (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.

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 a queue to trigger an AWS Lambda function (console).
- 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 Best practices for Amazon SQS.
- 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 Sample Code and Libraries and the developer centers:
  - Java
  - JavaScript
  - PHP
  - Python

- Ruby
- Windows & .NET

- Learn about keeping an eye on costs and resources in the Automating and troubleshooting Amazon SQS queues section.
- Learn about protecting your data and access to it in the Security section.
- Learn more about Amazon SQS workflows and processes:

Getting started with Amazon SQS standard queues

Amazon SQS offers *standard* as the default queue type. Standard queues support a nearly unlimited number of API calls per second, per API action (`SendMessage`, `ReceiveMessage`, or `DeleteMessage`). Standard queues support at-least-once message delivery. However, occasionally (because of the highly distributed architecture that allows nearly unlimited throughput), more than one copy of a message might be delivered out of order. Standard queues provide best-effort ordering which ensures that messages are generally delivered in the same order as they're sent.

Amazon SQS redundantly stores a message in more than one availability zone (AZ) before a `SendMessage` is acknowledged. Because message copies are stored in multiple AZs, no single computer, network, or AZ failure can make messages inaccessible.

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

You can use standard message queues in many scenarios, as long as your application can process messages that arrive more than once and out of order, for example:

- **Decouple live user requests from intensive background work** – Let users upload media while resizing or encoding it.
- **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.

For quotas related to standard queues, see Quotas.

For best practices of working with standard queues, see Recommendations for Amazon SQS standard and FIFO queues.

Message ordering

A standard queue makes a best effort to preserve the order of messages, but more than one copy of a message might be delivered out of order. If your system requires that order be preserved, we recommend using a *FIFO (First-In-First-Out) queue* or adding sequencing information in each message so you can reorder the messages when they're received.

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 that unavailable server, 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.

Topics

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=
```

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.

Quota	Description
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.
	The tag `Key` is required, but the tag `Value` is optional.
	The tag `Key` and tag `Value` are case-sensitive.

Quota	Description
	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.

Getting started with 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.

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

- E-commerce order management system where order is critical
- Integrating with a third-party systems where events need to be processed in order
- Processing user-entered inputs in the order entered
- Communications and networking – Sending and receiving data and information in the same order
- Computer systems – Making sure that user-entered commands are run in the right order
- Educational institutes – Preventing a student from enrolling in a course before registering for an account
- 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. For information on throughput quotas, see the section called “Quotas related to messages”.

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 (console). For Java examples, see Amazon SQS Java SDK examples.

For best practices of working with FIFO queues, see Additional recommendations for Amazon SQS FIFO queues and Recommendations for Amazon SQS standard and FIFO queues.

FIFO delivery logic

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.

Message ordering

The FIFO queue improves upon and complements the standard queue. The most important features of this queue type 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.

In addition, 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.

Exactly-once processing

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

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

> **Note**
>
> You can't convert an existing standard queue into a FIFO queue. To make the move, you must either create a new FIFO queue for your application or delete your existing standard queue and recreate it as a FIFO queue.

To make sure that your application correctly works with a FIFO queue, use the following checklist:

- Use the **recommended** high throughput mode for FIFO to achieve increased throughput. To learn more about messaging quotas, see Quotas related to messages.
- FIFO queues don't support per-message delays, only per-queue delays. If your application sets the same value of the `DelaySeconds` parameter on each message, you must modify your

application to remove the per-message delay and set `DelaySeconds` on the entire queue instead.

- Message group is a unique FIFO feature that enables customers to process messages in parallel while maintaining their respective ordering. Customers organize messages into message groups by specifying a message group ID. Message groups are often based on a business dimension for a given workload. To better scale with FIFO queues, use a more granular business dimension for message ID. The more message group IDs you distribute messages to, the greater number of messages FIFO makes available for consumption.
- Before sending messages to a FIFO queue, confirm the following:
  - If your application can send messages with identical message bodies, you can modify your application to provide a unique message deduplication ID for each sent message.
  - If your application sends messages with unique message bodies, you can enable content-based deduplication.
- You don't have to make any code changes to your consumer. However, if it takes a long time to process messages and your visibility timeout is set to a high value, consider adding a receive request attempt ID to each `ReceiveMessage` action. This allows you to retry receive attempts in case of networking failures and prevents queues from pausing due to failed receive attempts.

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

High throughput for FIFO queues

High throughput for FIFO queues supports a higher number of requests per API, per second. To increase the number of requests in high throughput for FIFO queues, you can increase the number of message groups you use. For more information on high throughput message quotas, see Amazon SQS service quotas in the *Amazon Web Services General Reference*. For information on per-queue quotas with high throughput for FIFO quotas, see Quotas related to messages and Partitions and data distribution for high throughput for SQS FIFO queues.

Topics

- Partitions and data distribution for high throughput for SQS FIFO queues
- Enable high throughput for FIFO queues

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:

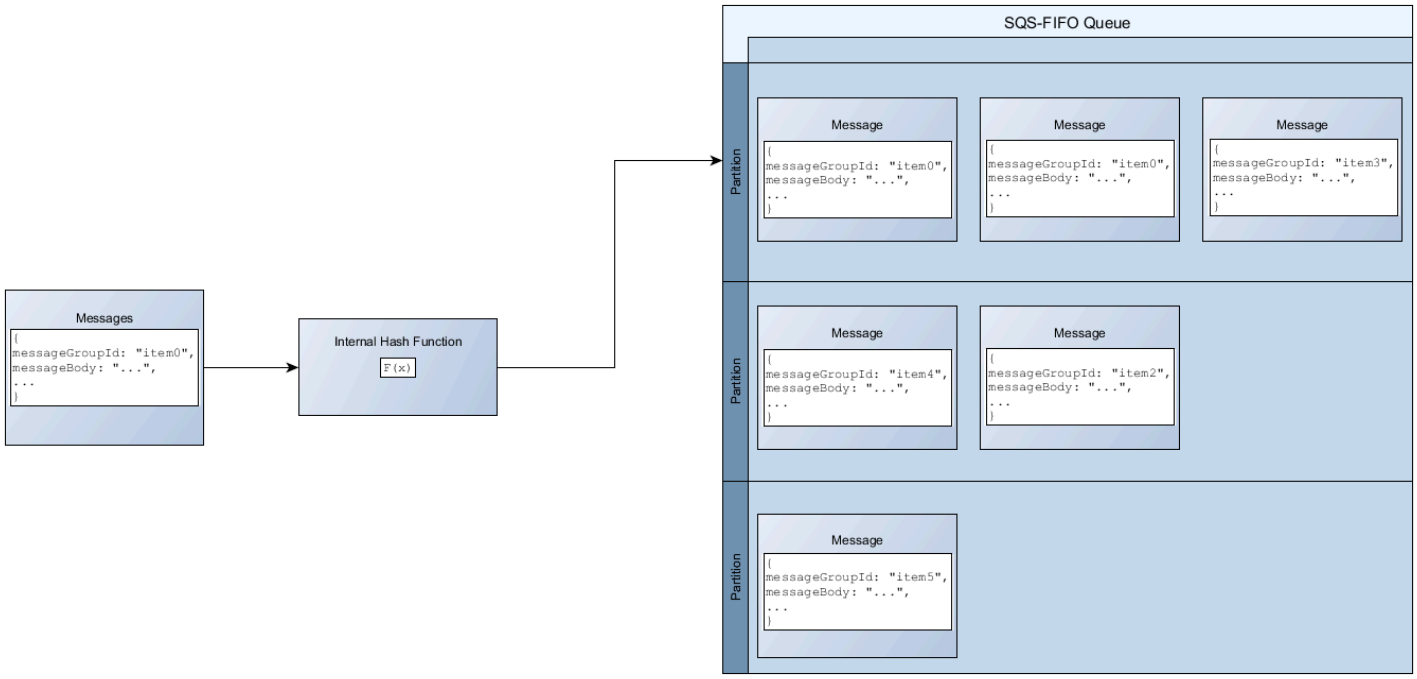
- If the current request rate approaches or exceeds what the existing partitions can support, additional partitions are allocated until the queue reaches the regional quota. For information on quotas, see Quotas related to messages.
- If the current partitions have low utilization, the number of partitions 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*.

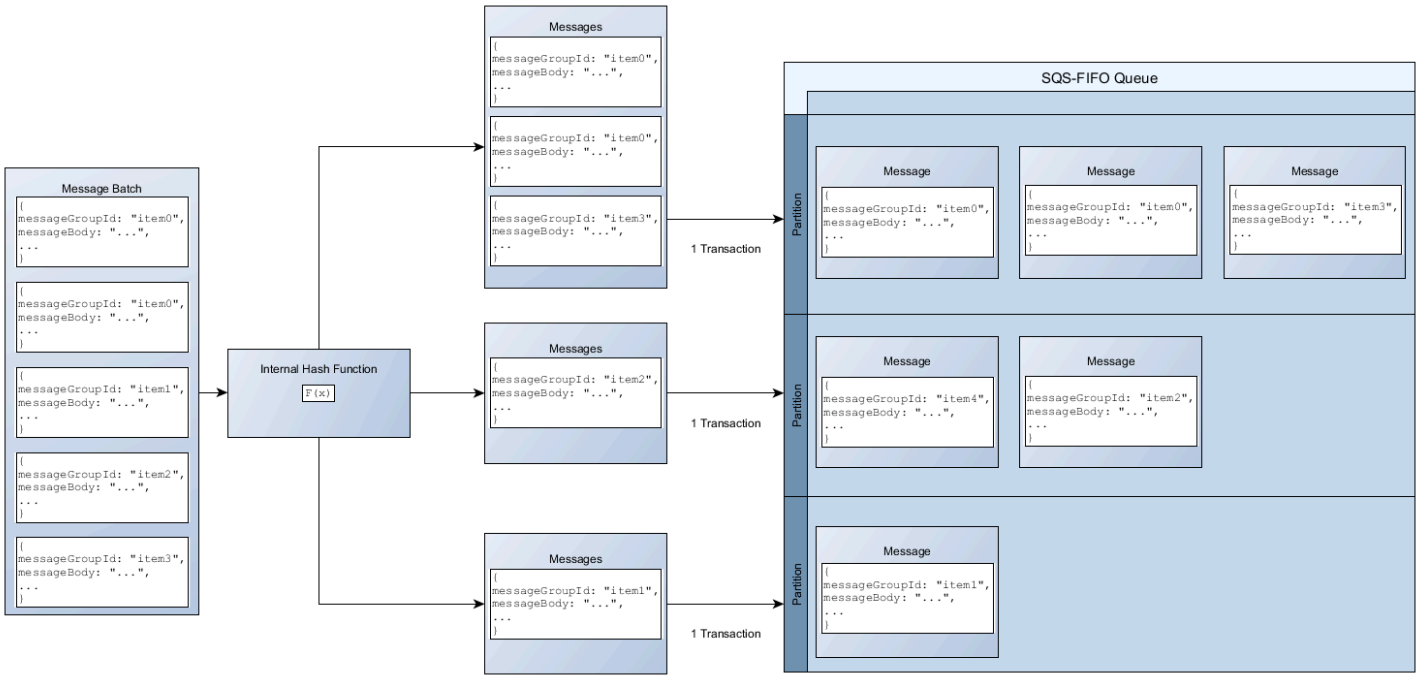
When using batch APIs, each message is routed based on the process described in Distributing data by message group IDs. Messages that are routed to the same partition are grouped and processed in a single transaction.

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.

Enable high throughput for FIFO queues

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 Quotas related to messages.

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*.

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.

Compatibility

Clients

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

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.

Amazon SQS queue and message identifiers

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

Topics

Identifiers for Amazon SQS FIFO 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 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 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.

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.
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.

Quota	Description
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 quotas

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

Topics

Quotas related to messages

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.
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.

Quota	Description
	`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`).
	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. To request a quota increase, submit a support request.

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. • 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 synchrono us 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 Quotas related to policies.

Quotas related to policies

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

Amazon SQS provides the following features and capabilities.

Topics

Message metadata

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 (console).

> **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.

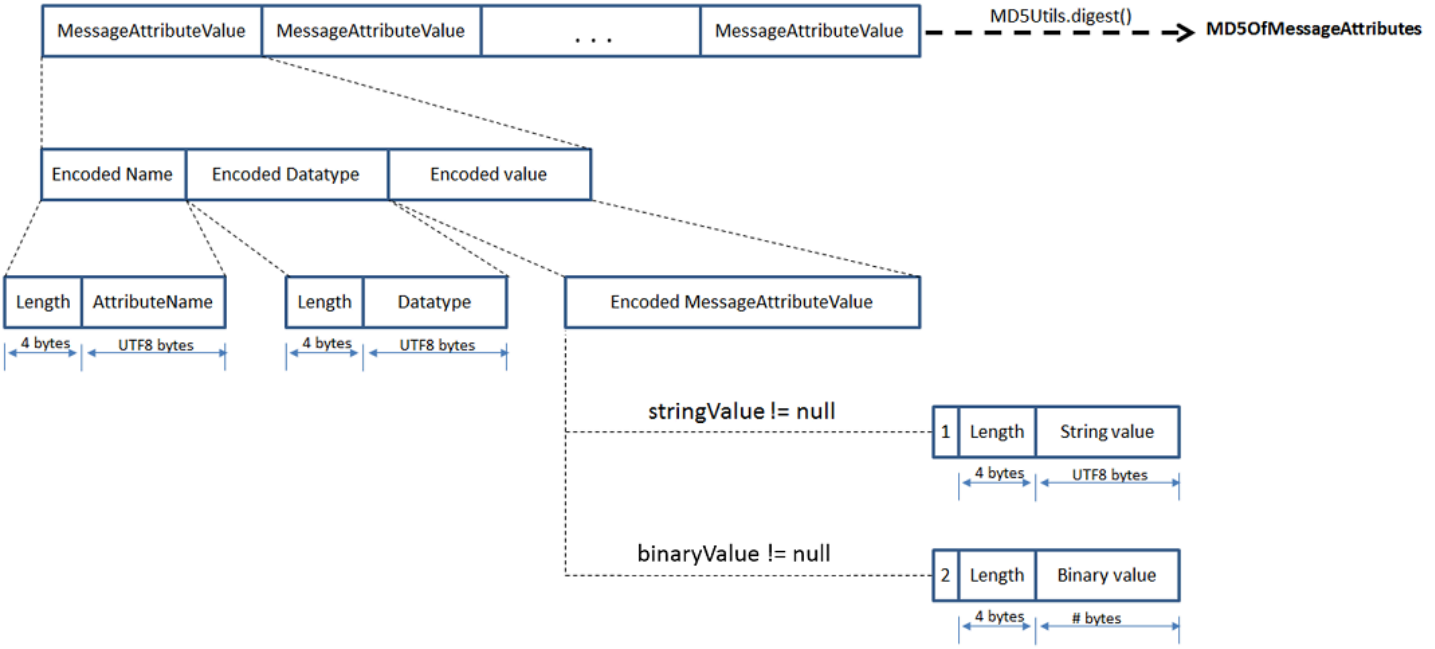
> **Note**
>
> Always include custom data type suffixes in the MD5 message-digest calculation.

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 standard queues, 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.
> For FIFO queues, the result is exact.

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.	`ApproximateNumberOfMessages Visible`

Task	Attribute name
Get the approximate 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.	`ApproximateNumberOfMessagesDelayed`
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.	`ApproximateNumberOfMessagesNotVisible`

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 Quotas.

Amazon SQS short and long polling

Amazon SQS provides short polling and long polling to receive messages from a queue. By default, queues use short polling.

With *short polling*, the `ReceiveMessage` request queries only a subset of the servers (based on a weighted random distribution) to find messages that are available to include in the response. Amazon SQS sends the response right away, even if the query found no messages.

With *long polling*, the `ReceiveMessage` request queries all of the servers for messages. Amazon SQS sends a response after it collects at least one available message, up to the maximum number of messages specified in the request. Amazon SQS sends an empty response only if the polling wait time expires.

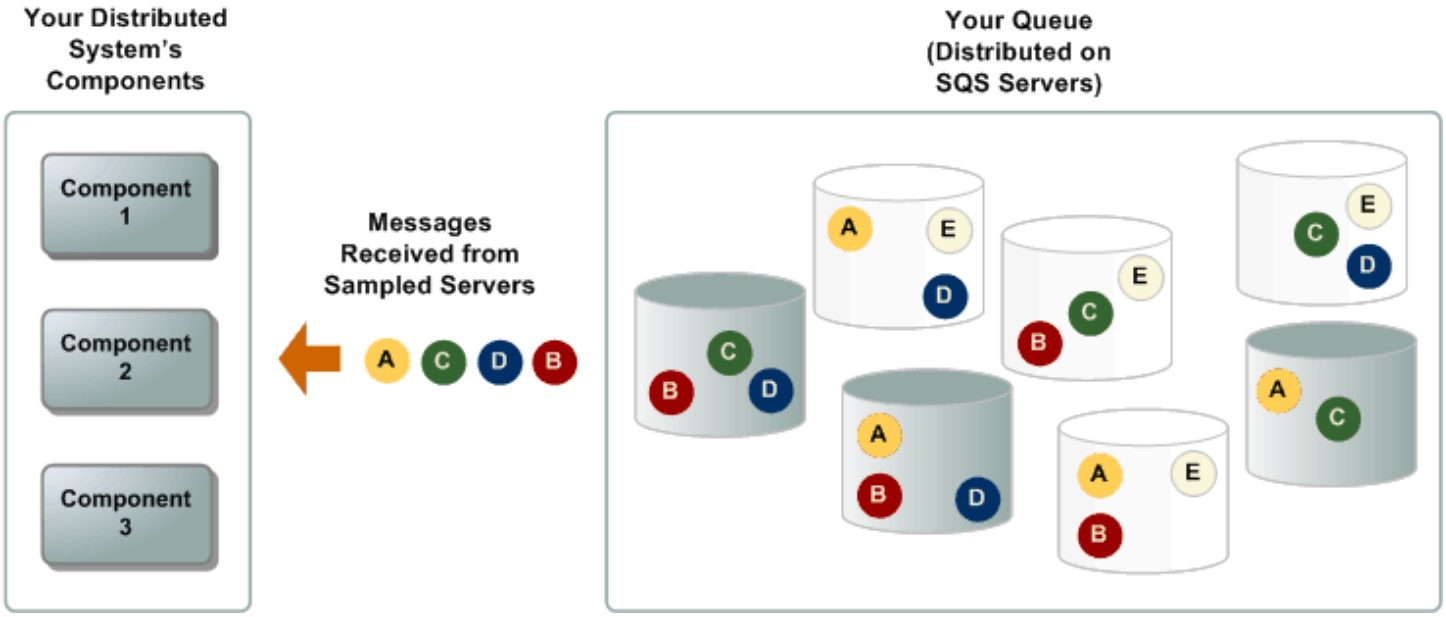
The following sections explain the details of short polling and long polling.

Topics

Consuming messages using short polling

When you consume messages from a queue 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 (console). For best practices, see Setting up long polling.

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 a 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 dead-letter queues

Amazon SQS supports *dead-letter queues* (DLQ), which other queues (*source queues*) can target for messages that can't be processed (consumed) successfully. Dead-letter queues are useful for debugging your application or messaging system because they let you isolate unconsumed messages to determine why their processing didn't succeed. For information about configuring a dead-letter queue using the Amazon SQS console, see Configuring a dead-letter queue (console). Once you have debugged the consumer application or the consumer application is available to consume the message, you can use the **dead-letter queue redrive capability** to move the messages back to the source queue.

> **Important**
>
> Amazon SQS does *not* create the dead-letter queue automatically. You must first create the queue before using it as a dead-letter queue.

Topics

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

How do dead-letter queues work?

Sometimes, messages can't be processed because of a variety of possible issues, such as erroneous conditions within the producer or consumer application or an unexpected state change that causes an issue with your application code. For example, if a user places a web order with a particular product ID, but the product ID is deleted, the web store's code fails and displays an error, and the message with the order request is sent to a dead-letter queue.

Occasionally, producers and consumers might fail to interpret aspects of the protocol that they use to communicate, causing message corruption or loss. Also, the consumer's hardware errors might corrupt message payload.

The *redrive policy* specifies the *source queue*, the *dead-letter queue*, and the conditions under which Amazon SQS moves messages from the former to the latter if the consumer of the source queue fails to process a message a specified number of times. The `maxReceiveCount` is the number of times a consumer tries receiving a message from a queue without deleting it before being moved to the dead-letter queue. Setting the `maxReceiveCount` to a low value such as 1 would result in any failure to receive a message to cause the message to be moved to the dead-letter queue. Such failures include network errors and client dependency errors. 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. This policy applies to a potential dead-letter queue. You can choose whether to allow all source queues, allow specific source queues, or deny all source queues. The default is to allow 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.

To specify a dead-letter queue, you can use the console or the AWS SDKs. You must do this for each queue that sends messages to a dead-letter queue. Multiple queues of the same type can target a single dead-letter queue. For more information, see Configuring a dead-letter queue (console) and the `RedrivePolicy` and `RedriveAllowPolicy` attributes of the `CreateQueue` or `SetQueueAttributes` action.

Important

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.
You must use the same AWS account to create the dead-letter queue and the other queues that send messages to the dead-letter queue. Also, dead-letter queues must reside in the same region as the other queues that use the dead-letter queue. For example, if you create a queue in the US East (Ohio) region and you want to use a dead-letter queue with that queue, the second queue must also be in the US East (Ohio) region.
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.

What are the benefits of dead-letter queues?

The main task of a dead-letter queue is to handle the lifecycle of unconsumed messages. A dead-letter queue lets you set aside and isolate messages that can't be processed correctly to determine why their processing didn't succeed. Setting up a dead-letter queue allows you to do the following:

- Configure an alarm for any messages moved to a dead-letter queue.
- Examine logs for exceptions that might have caused messages to be moved to a dead-letter queue.
- Analyze the contents of messages moved to a dead-letter queue to diagnose software or the producer's or consumer's hardware issues.

- Determine whether you have given your consumer sufficient time to process messages.

How do different queue types handle message failure?

Standard queues

Standard queues keep processing messages until the expiration of the retention period. This continuous processing of messages minimizes the chances of having your queue blocked by messages that can't be processed. Continuous message processing also provides faster recovery for your queue.

In a system that processes thousands of messages, having a large number of messages that the consumer repeatedly fails to acknowledge and delete might increase costs and place extra load on the hardware. Instead of trying to process failing messages until they expire, it is better to move them to a dead-letter queue after a few processing attempts.

> **Note**
>
> Standard queues 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 valid messages can slow down. Thus, to maintain the efficiency of your queue, make sure that your application correctly handles message processing.

FIFO queues

FIFO queues provide exactly-once processing by consuming messages in sequence from a message group. Thus, 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.

> **Note**
>
> FIFO queues allow a lower number of in flight messages. Thus, to keep your FIFO queue from getting blocked by a message, make sure that your application correctly handles message processing.
> 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.

When should I use a dead-letter queue?

✔ Do use dead-letter queues with standard queues. You should always take advantage of dead-letter queues when your applications don't depend on ordering. Dead-letter queues can help you troubleshoot incorrect message transmission operations.

Note

Even when you use dead-letter queues, you should continue to monitor your queues and retry sending messages that fail for transient reasons.

✔ Do use dead-letter queues to decrease the number of messages and to reduce the possibility of exposing your system to *poison-pill messages* (messages that can be received but can't be processed).

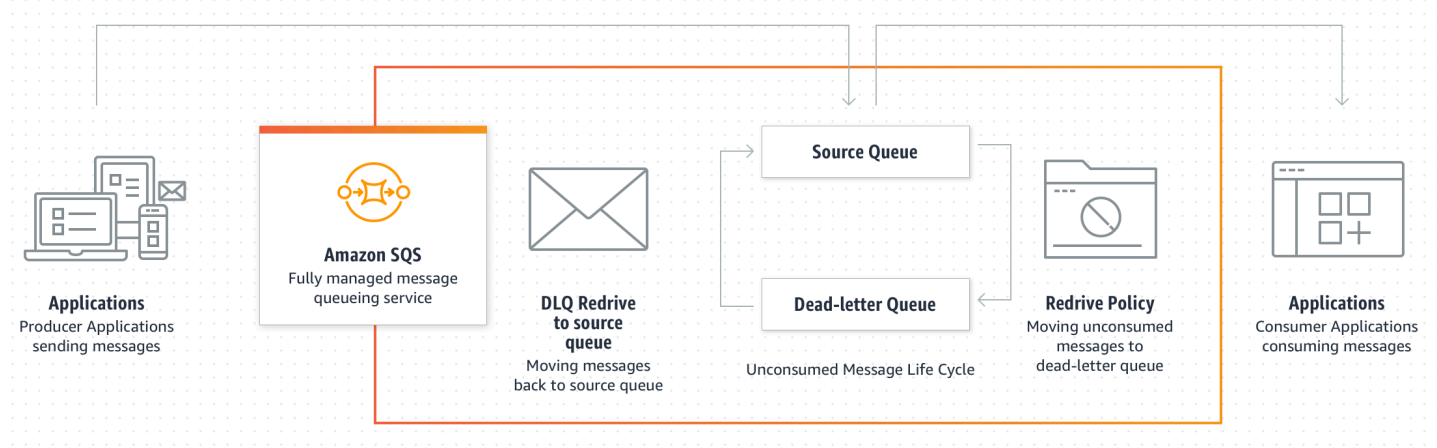
✘ Don't use a dead-letter queue with standard queues when you want to be able to keep retrying the transmission of a message indefinitely. For example, don't use a dead-letter queue if your program must wait for a dependent process to become active or available.

✘ Don't use a dead-letter queue with a FIFO queue if you don't want to break the exact order of messages or operations. For example, don't use a dead-letter queue with instructions in an Edit Decision List (EDL) for a video editing suite, where changing the order of edits changes the context of subsequent edits.

Moving messages out of a dead-letter queue

You can use *dead-letter queue redrive* to manage the lifecycle of unconsumed messages. After you have investigated the attributes and related metadata available for unconsumed messages in a standard or FIFO dead-letter queue, you can redrive the messages back to their source queues. Dead-letter queue redrive reduces API call billing by batching the messages while moving them.

The redrive task uses Amazon SQS's `SendMessageBatch`, `ReceiveMessage`, and `DeleteMessageBatch` APIs on behalf of the user to redrive the messages. Therefore, all redriven messages are considered new messages with a new `messageid`, `enqueueTime`, and retention period. The pricing of dead-letter queue redrive uses the number of API calls invoked and bills based on the Amazon SQS pricing.



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. For instructions about configuring a dead-letter queue redrive, see Configuring a dead-letter queue redrive.

Note

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.

When messages are redriven from a FIFO dead-letter queue, the message groupID and deduplicationID remain the same, and the message receives a new messageID.

Amazon SQS 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 inteweave with the new messages from the producer.

Troubleshooting dead-letter queues

In some cases, Amazon SQS dead-letter queues might not always behave as expected. This section gives an overview of common issues and shows how to resolve them.

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. Thus, 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. Thus, it is possible for the values of `NumberOfMessagesSent` and `NumberOfMessagesReceived` to be different.

For information about creating and configuring a dead-letter queue redrive

Note that the 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.

In case of insufficient permissions, the dead-letter queue redrive to the source queue does not initiate the message redrive and can fail the task. You can view the status of your message redrive task to remediate the issues and try again.

Topics

Configuring a dead-letter queue (console)

A *dead-letter queue* is a queue that one or more source queues can use for messages that are not consumed successfully. For more information, see Amazon SQS dead-letter queues.

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 (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. Choose 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**.

Configuring a dead-letter queue redrive

You can configure a *dead-letter queue redrive* to move standard unconsumed messages out of an existing dead-letter queue back to their source queues. For more information about dead letter queue redrive, see Moving messages out of a dead-letter queue.

Configuring a dead-letter queue redrive for an existing standard queue (API)

You can configure a dead-letter queue redrive using the following 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 (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.

  > **Note**
  >
  > The custom destination queue must match the type of the dead-letter queue. For example, if the dead-letter queue is a FIFO queue, then the custom destination queue must be a FIFO queue as well.

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.
   > The redrive task resets the retention period. A new `messageID` and `enqueueTime` are assigned to redriven messages.

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. • 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"
      ],
      "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: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"
      ],
      "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 (DLQ) 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

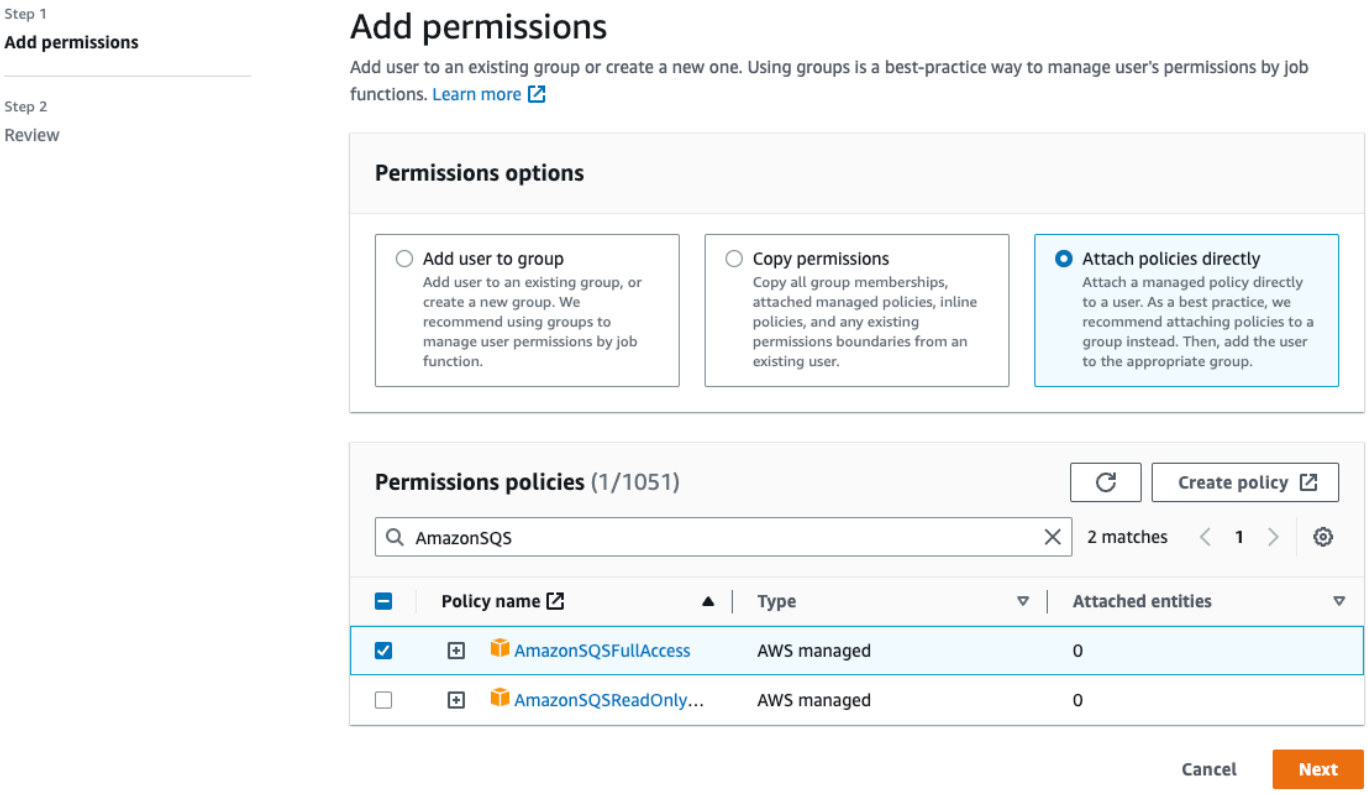
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage",
        "sqs:GetQueueAttributes",
```

```
        "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

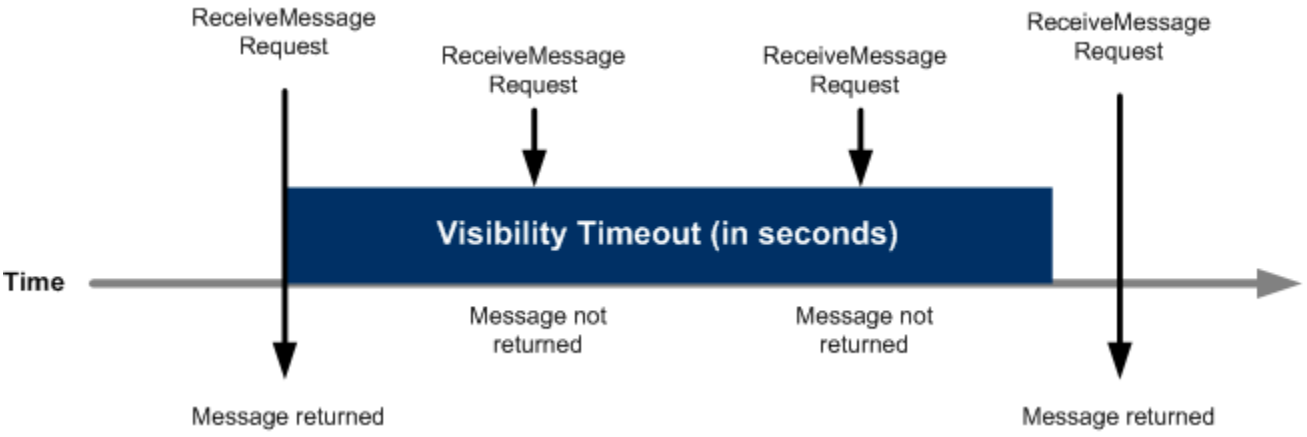
If you are using customer managed policies (CMPs), you can use AWS CloudTrail and IAM to identify the policies impacted by the queue permissions update.

> **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`.

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 (console).

Note

For standard queues, the visibility timeout isn't a guarantee against receiving a message twice. For more information, see 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 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.

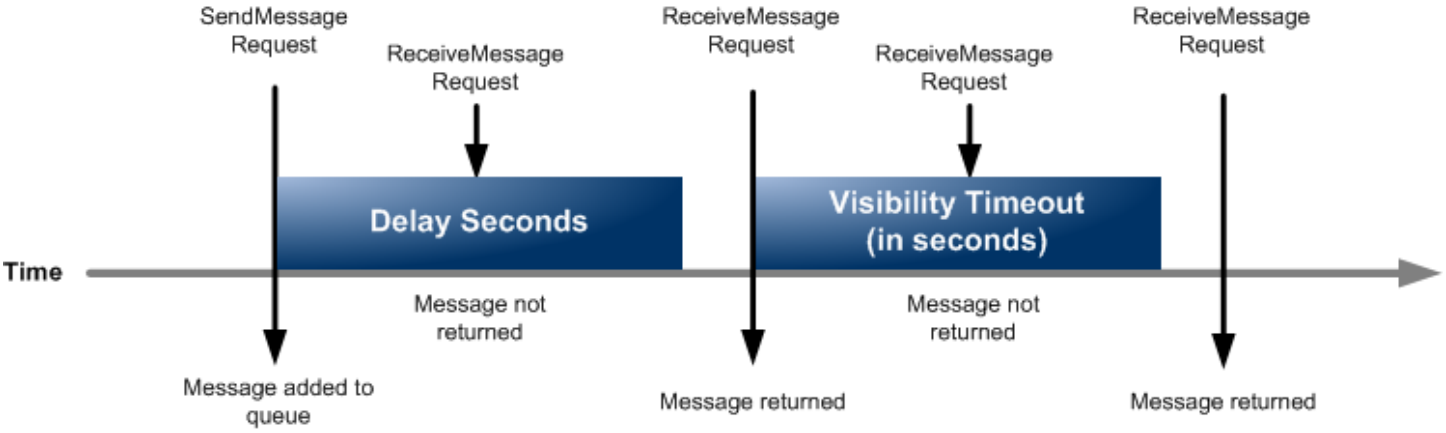
Amazon SQS delay queues

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 (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 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 Quotas related to messages.

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.

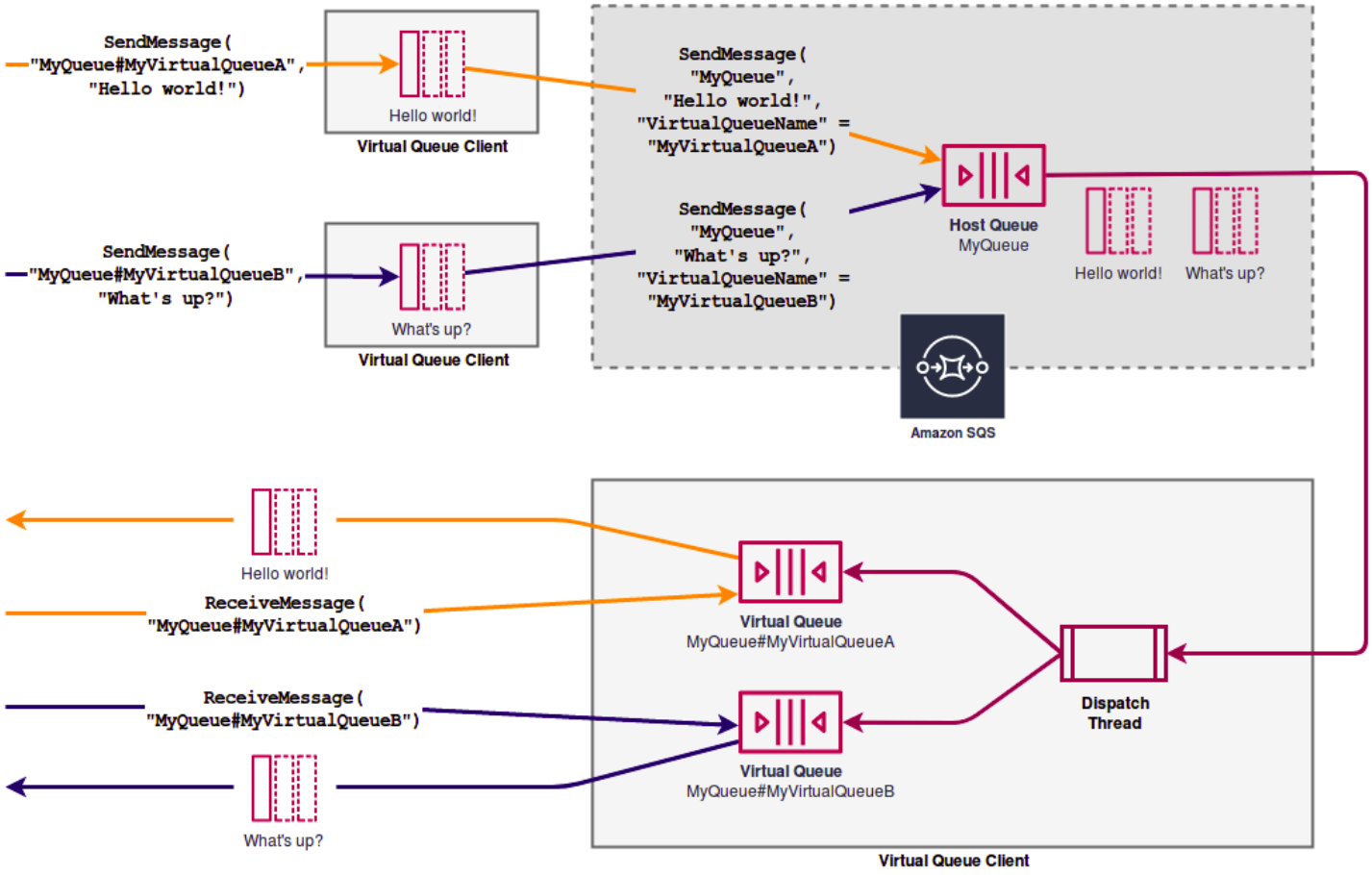
The `AmazonSQSTemporaryQueuesClient` class turns all queues it creates into temporary queues automatically. It also creates host queues with the same queue attributes automatically, on demand. These queues' names share a common, configurable prefix (by default, `__RequesterClientQueues__`) that identifies 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 connects 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 the 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 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 S3 bucket
- Delete the message object from an 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 1.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 2010-2022 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 2010-2022 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 (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 (ABAC) 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.

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

- ABAC condition keys for Amazon SQS

- Tagging for access control
- Creating IAM users and Amazon SQS queues
- Testing attribute-based access control

ABAC condition keys for Amazon SQS

You can use the following condition keys to control function actions:

ABAC condition key	Description	Policy type	Amazon SQS operations
aws:ResourceTag	Tag (key and value) on the Amazon SQS queue matches the tag (key and value) or tag pattern in the policy	IAM policy only	Amazon SQS queue resource operations
aws:RequestTag	Tag (key and value) on the Amazon SQS queue resource operations matches the tag (key and value) or tag pattern in the policy	Queue policy and IAM policies	TagQueue, UntagQueue, CreateQueue
aws:TagKeys	Tag keys in the request match the tag keys in the policy	Queue policy and IAM policies	TagQueue, UntagQueue, CreateQueue

Tagging for access control

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

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 (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 Getting started with 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 and Quotas related to messages.
- **Redrive allow policy**: defines which source queues can use this queue as the dead-letter queue. For more information, see Amazon SQS dead-letter queues.

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 and Quotas related to messages.

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 access policy (console)

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 (SSE) for a queue using SQS-managed encryption keys (console)

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 (SSE) for a queue (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.

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 an Amazon SQS queue (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 an Amazon SQS queue to an Amazon SNS topic (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 a queue to trigger an AWS Lambda function (console)

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.

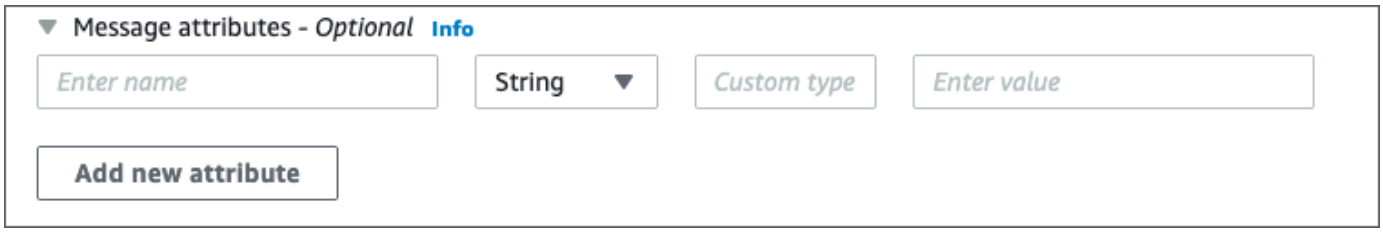
   On the **Details** page, the **Lambda triggers** tab displays the Lambda function and its status. It takes approximately 1 minute for the Lambda function to become associated with your queue.
7. To verify the results of the configuration, send a message to your queue and then view the triggered Lambda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

Sending a message with attributes (console)

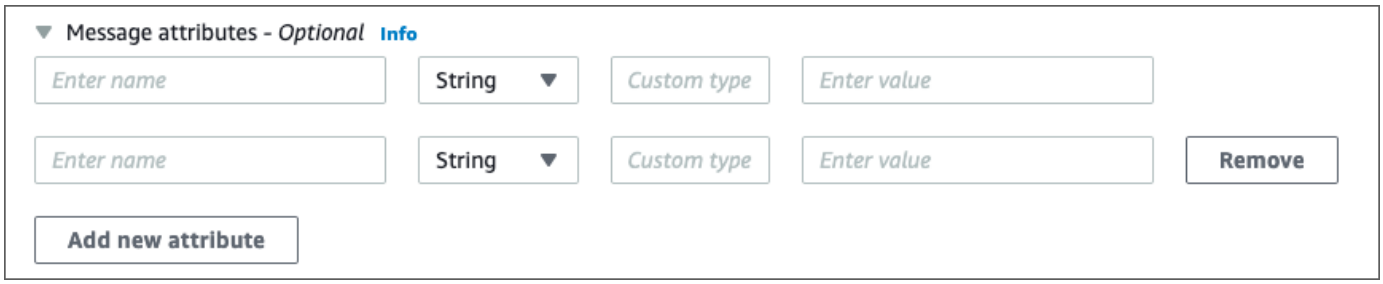
For standard and FIFO queues, you can include structured metadata (such as timestamps, geospatial data, signatures, and identifiers) with messages. For more information, see Amazon SQS message attributes.

To send a message with attributes to a queue (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.

Best practices for Amazon SQS

These best practices can help you make the most of Amazon SQS.

Topics

Recommendations for Amazon SQS standard and FIFO queues

The following best practices can help you reduce costs and process messages efficiently using Amazon SQS.

Topics

Working with Amazon SQS messages

The following guidelines can help you process messages efficiently using Amazon SQS.

Topics

Processing messages in a timely manner

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.

Handling request errors

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).

Setting up 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 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.

Capturing problematic messages

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

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.

Avoiding inconsistent message processing

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.

Implementing request-response systems

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

- Don't create reply queues *per message*. Instead, create reply queues on startup, *per producer*, and use a correlation ID message attribute to map replies to requests.
- Don't let your producers share reply queues. This can cause a producer to receive 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).

Reducing Amazon SQS costs

The following best practices can help you reduce costs and take advantage of additional potential cost reduction and near-instantaneous response.

Batching message actions

To reduce costs, batch your message actions:

- To send, receive, and delete messages, and to change the message visibility timeout for multiple messages with a single action, use the Amazon SQS batch API actions.
- To combine client-side buffering with request batching, use long polling together with the buffered asynchronous client included with the AWS SDK for Java.

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

Using the appropriate polling mode

- 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.

Moving from an Amazon SQS Standard queue to a FIFO queue

If you're not setting the `DelaySeconds` parameter on each message, you can move to a FIFO queue by providing a message group ID for every sent message.

For more information, see Moving from a standard queue to a FIFO queue.

Additional recommendations for Amazon SQS FIFO queues

The following best practices can help you use the message deduplication ID and message group ID optimally. For more information, see the `SendMessage` and `SendMessageBatch` actions in the *Amazon Simple Queue Service API Reference*.

Topics

- Using the Amazon SQS message deduplication ID
- Using the Amazon SQS message group ID
- Using the Amazon SQS receive request attempt ID

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

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

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.
- 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

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

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

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

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

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

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 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 (SSE)

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.

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 a 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

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 `230.00000000000000001`.

```
final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
messageAttributes.put("AccurateWeight", new MessageAttributeValue()
.withDataType("Number")
.withStringValue("230.00000000000000001"));
```

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.

Working with JMS and 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 Supported JMS 1.1 implementations and the Amazon SQS FAQs.

Topics

Prerequisites

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 example 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 Getting started with the Amazon SQS Java Messaging Library.

Getting started with 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

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.

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 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 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 example 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 Getting started with 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 Getting started with the Amazon SQS Java Messaging Library example. For a complete example of an asynchronous consumer, see `AsyncMessageReceiver.java` in Working Java example 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 Getting started with 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 Getting started with 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 example 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 Amazon SQS Java Message Service (JMS) Client 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:

```
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 example 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.)

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 2010-2022 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 2010-2022 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 2010-2022 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 2010-2022 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 2010-2022 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 2010-2022 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 2010-2022 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 2010-2022 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 2010-2022 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() );
        }
    }
}
```

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 (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.

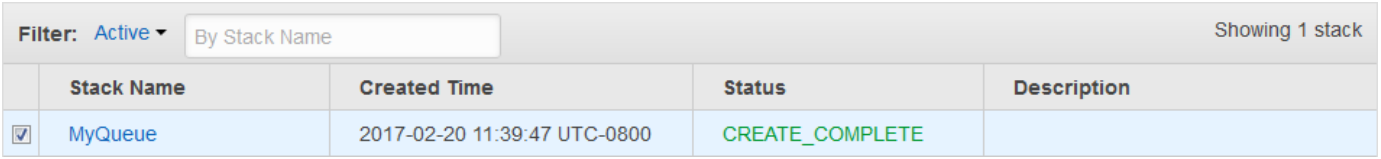
   > **Note**
   >
   > The name of a FIFO queue must end with the `.fifo` suffix.

```
{
   "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.
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 Receive and delete a message (console).

For information about deleting your resources, see the following:

- Deleting a VPC Endpoint in the *Amazon VPC User Guide*
- Delete a queue
- Terminate Your Instance in the *Amazon EC2 User Guide for Linux Instances*
- 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 for Linux Instances*

Automating and troubleshooting Amazon SQS queues

This section provides information about automating and troubleshooting Amazon SQS queues.

Topics

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*.

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

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

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

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 (SSE)
- 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*.

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 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.

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`
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 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 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 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). Thus, 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

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
- Creating an Amazon VPC endpoint policy for Amazon SQS

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 How IAM roles differ from resource-based policies 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 (IAM policies and Amazon SQS 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

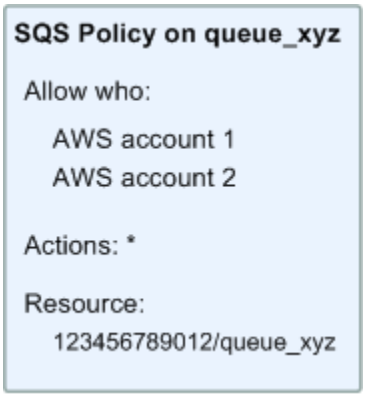
    If you want to grant the permission to assume the role to an AWS service, the principal in the trust policy can also be an AWS service principal.

For more information about using IAM to delegate permissions, see Access Management in the *IAM User Guide*.

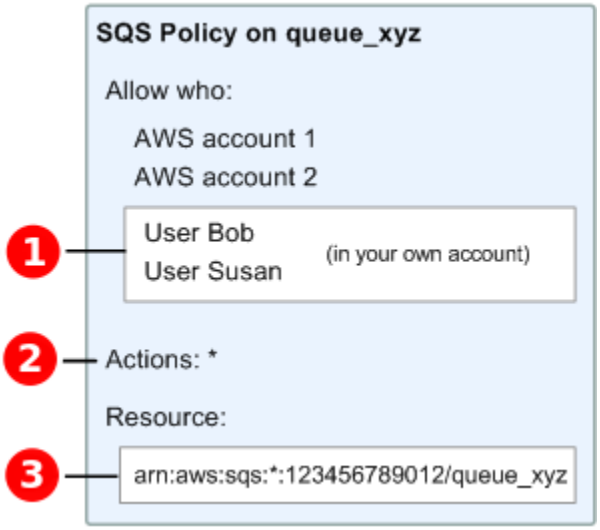
While Amazon SQS works with IAM policies, it has its own policy infrastructure. You can use an Amazon SQS policy with a queue to specify which AWS Accounts have access to the queue. You can specify the type of access and conditions (for example, a condition that grants permissions to use `SendMessage`, `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 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 How IAM roles differ from resource-based policies 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 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 How IAM roles differ from resource-based policies 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.

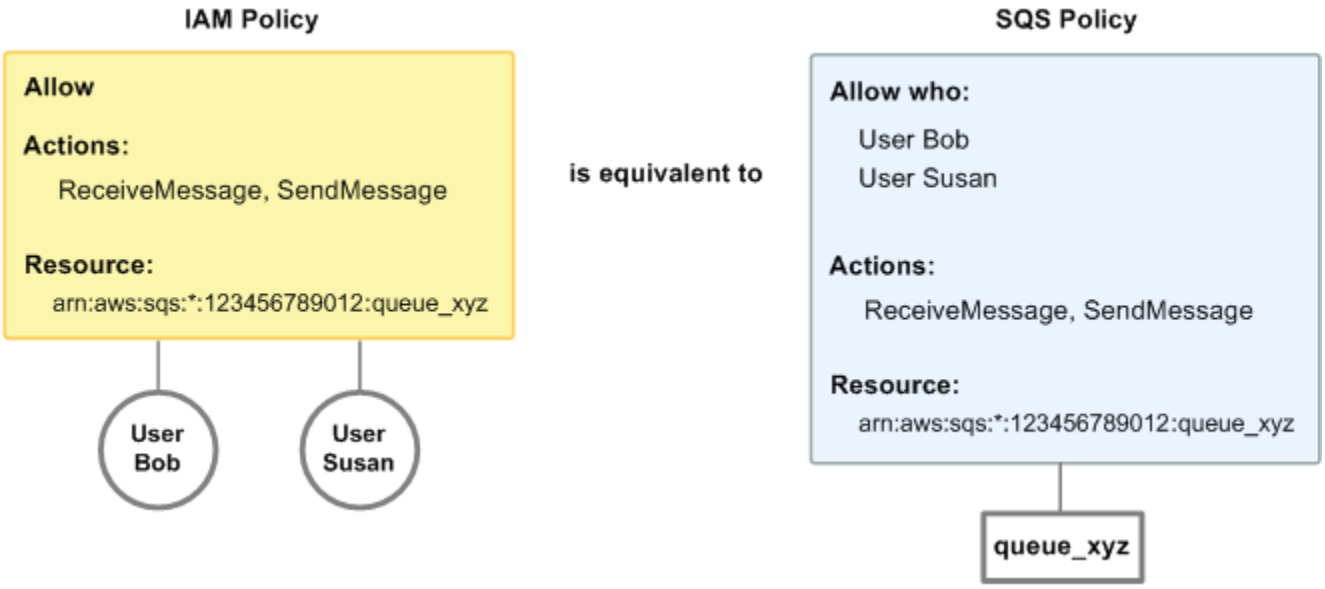
With the exception of `ListQueues`, all Amazon SQS actions support resource-level permissions. For more information, see Amazon SQS API permissions: Actions and resource reference.

Topics

Using Amazon SQS and IAM policies

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.

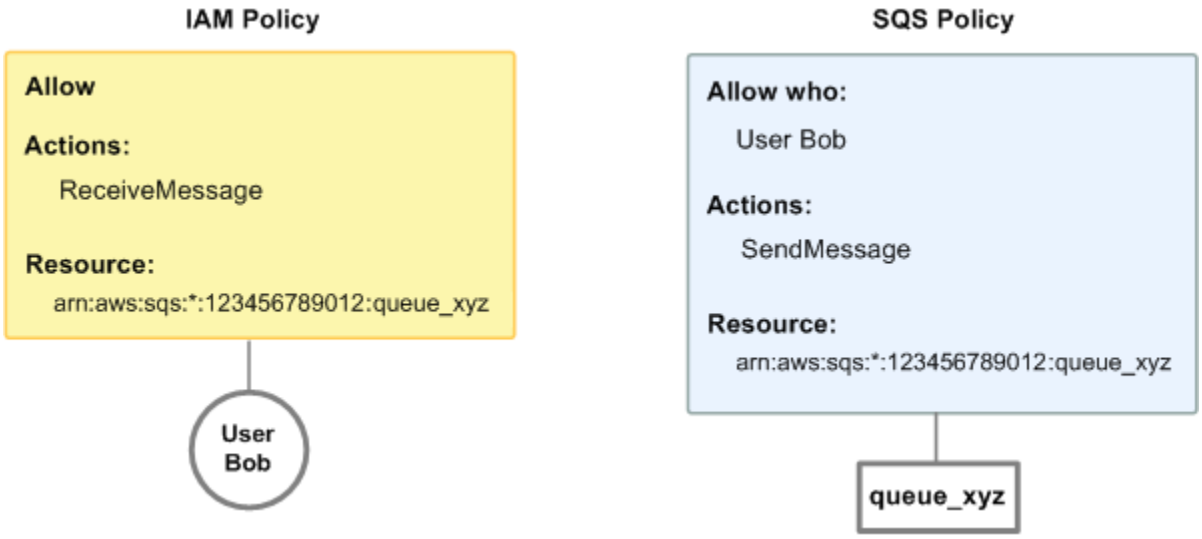


> **Note**
>
> This example shows simple policies without conditions. You can specify a particular condition in either policy and get the same result.

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.

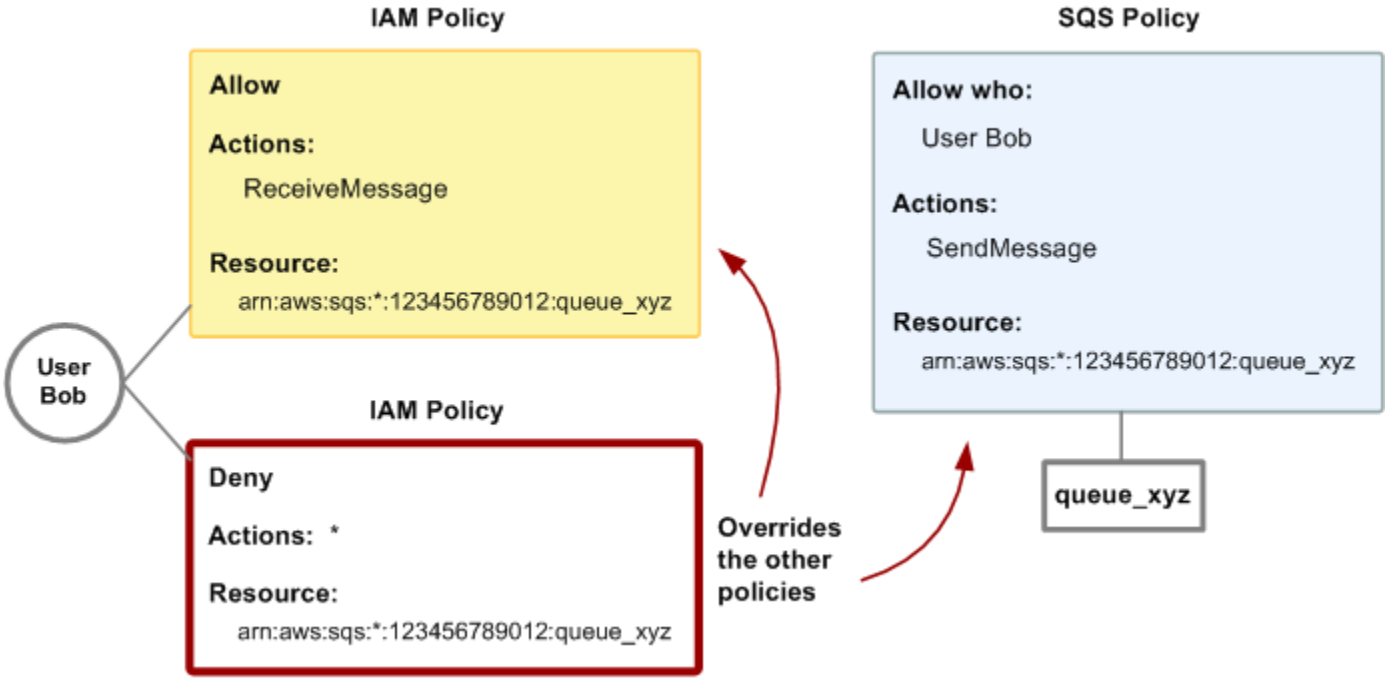
It is up to you how you use both of the systems together to manage your permissions. The following examples show how the two policy systems work together.

- 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 for Linux Instances*.

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.

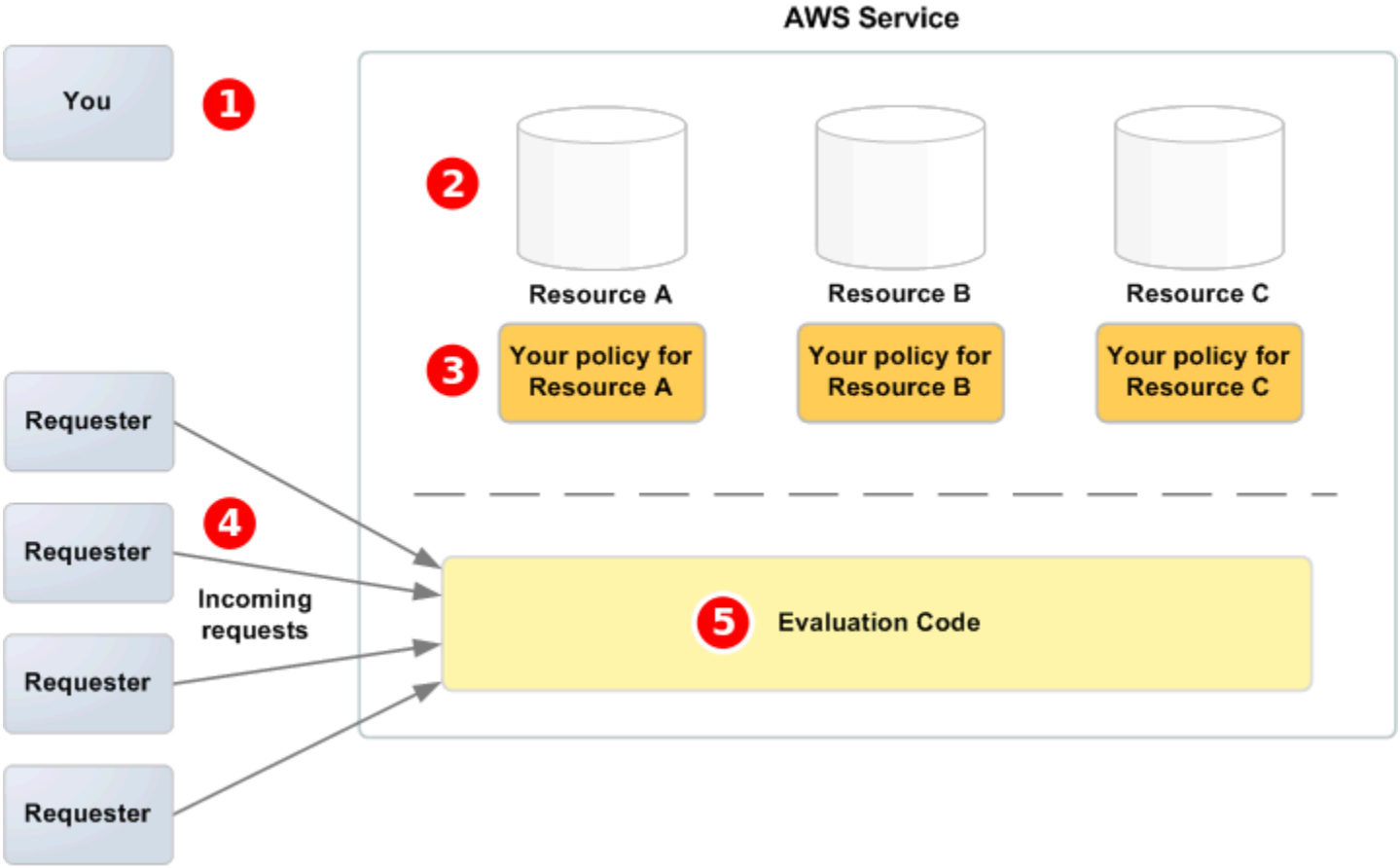
Topics

- Amazon SQS access control architecture
- Amazon SQS access control process workflow
- Amazon SQS Access Policy Language key concepts
- Amazon SQS Access Policy Language evaluation logic

- Relationships between explicit and default denials in the Amazon SQS Access Policy Language
- Limitations of Custom Policies
- Custom Amazon SQS Access Policy Language examples

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

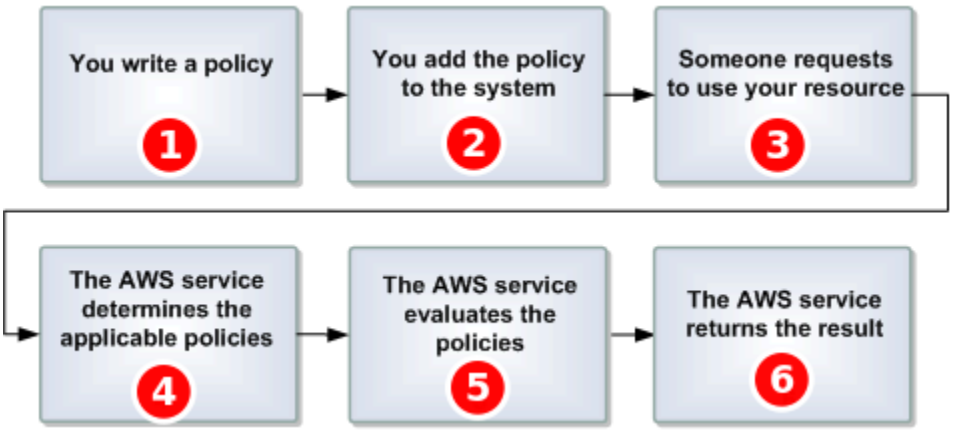
Requesters and their incoming requests to the AWS service.

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

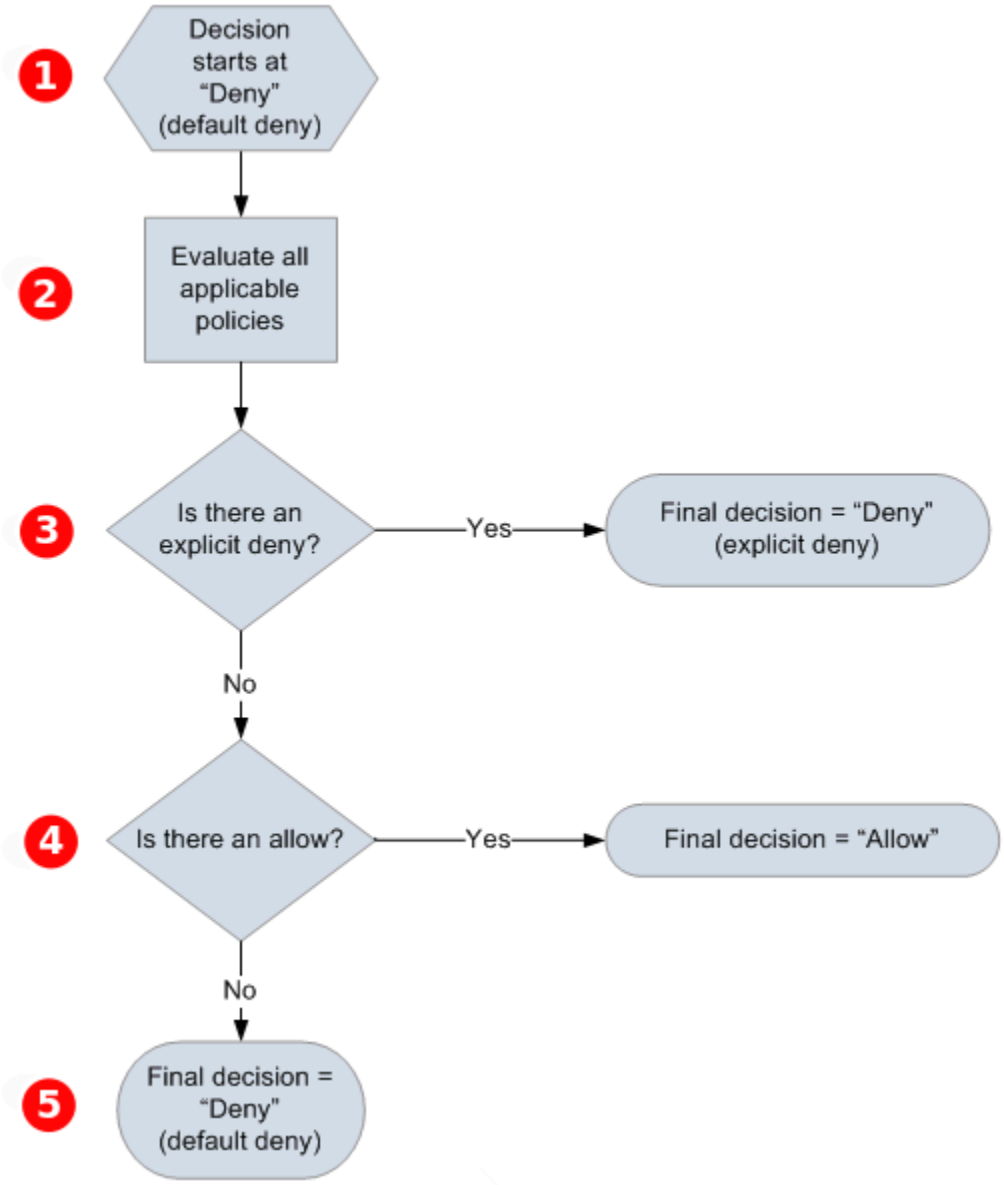
The user who sends a request for access to a **Resource**.

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).

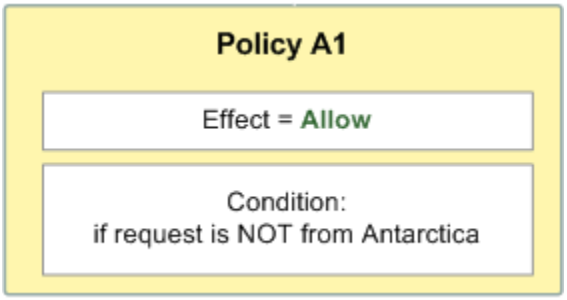
5

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**).

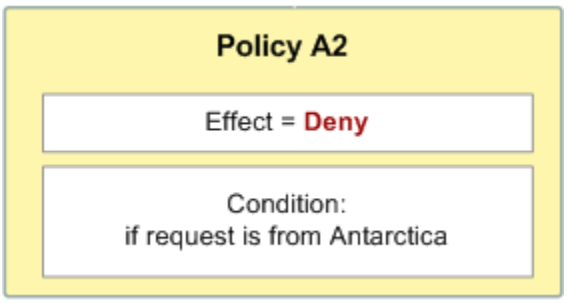
Relationships between explicit and default denials in the Amazon SQS Access Policy Language

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.

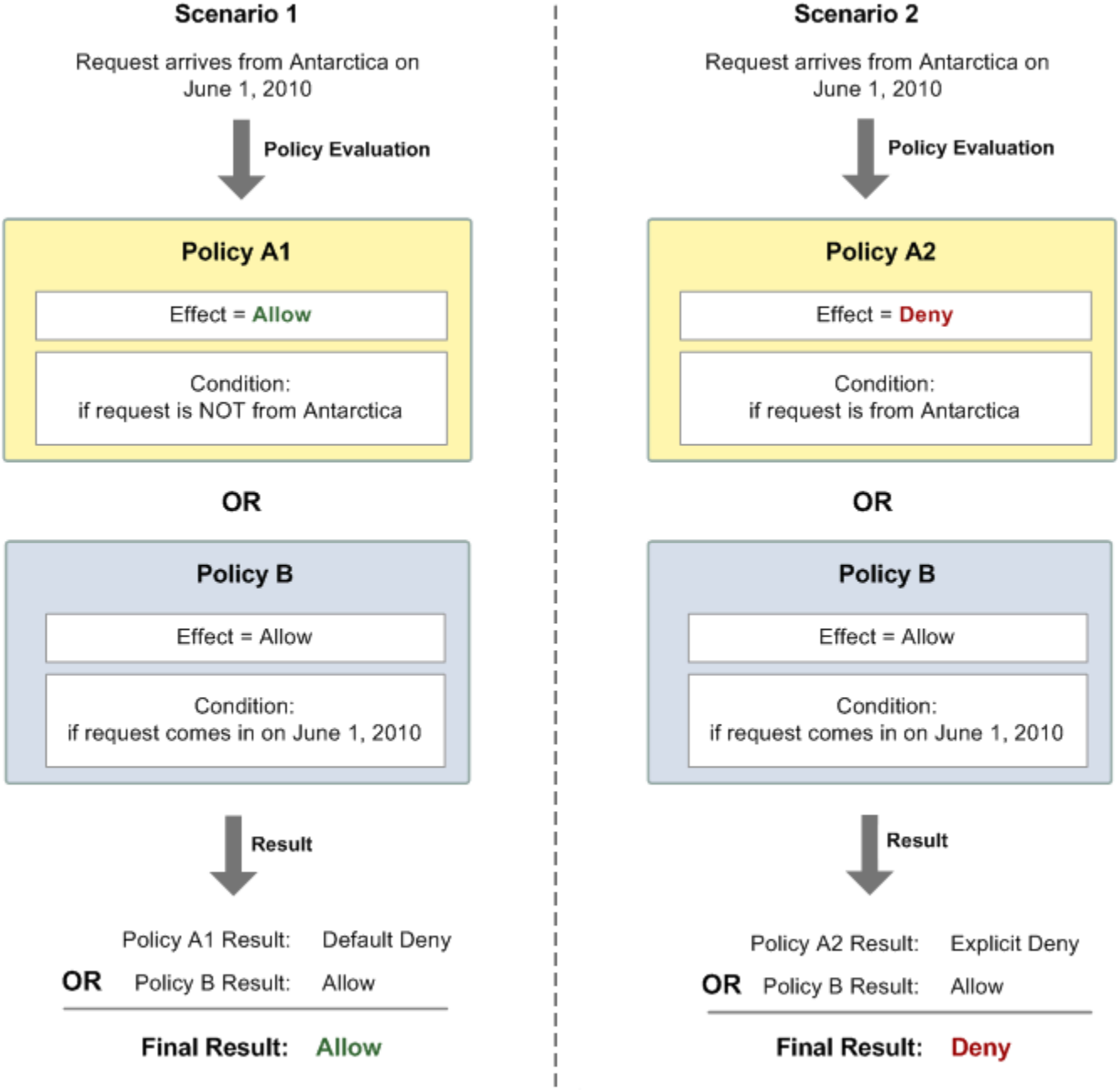


If a user sends a request from the U.S., the condition is met (the request isn't from Antarctica), and the request results in an **allow**. However, if a user sends a request from Antarctica, the condition isn't met and the request defaults to a **default-deny**. You can change the result to an **explicit-deny** 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 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
```

Managing access to your encrypted Amazon SQS queue using least privilege Amazon SQS policy and AWS KMS key policy

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.

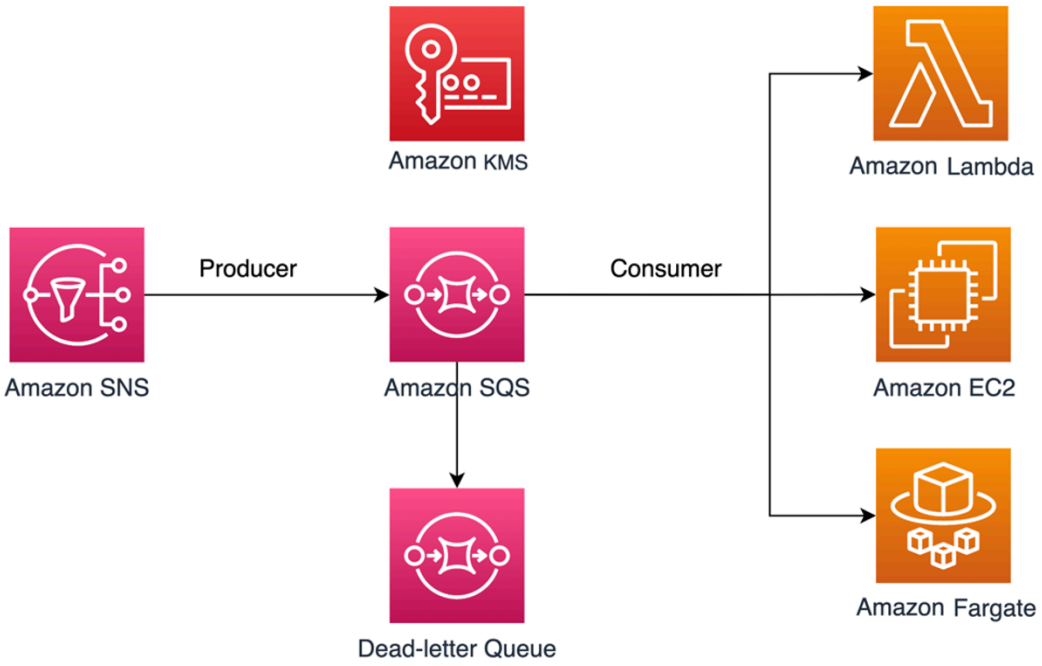
This guide also describes how resource-based policies address the confused-deputy problem by using the `aws:SourceArn`, `aws:SourceAccount`, and `aws:PrincipalOrgID` global IAM condition context keys.

Topics

- Overview
- Least privilege key policy for Amazon SQS
- Amazon SQS policy statements for the dead-letter queue
- Prevent the cross-service confused deputy problem
- Use IAM Access Analyzer to review cross-account access

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 Amazon SQS queues.

Topics

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.

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"
```

```
  }
```

SendMessageBatch

The following example shows a CloudTrail data event for SendMessageBatch.

```
    {
  "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": "2023-11-07T23:59:05Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "SendMessageBatch",
  "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": {
    "delaySeconds": 0,
    "entries": [
      {
        "id": "0",
        "messageBody": "HIDDEN_DUE_TO_SECURITY_REASONS",
```

```
        "messageAttributes": [
          {
            "name": "HIDDEN_DUE_TO_SECURITY_REASONS"
          }
        ],
        "messageDeduplicationId": "MsgDedupIdSdk1027092b6-b6f6-41af-a084-e72d572a6d4b",
        "messageGroupId": "MsgGroupIdSdk12"
      }
    ],
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue"
  },
  "responseElements": {
    "successful": [
      {
        "id": "0",
        "messageId": "9048ab28-e38d-46da-b9fe-f70b3873f888",
        "mD5OfMessageBody": "0f1a575a56eb5cf5072a8dedc585d2dd",
        "mD5OfMessageAttributes": "6e1d6d5d774a05efe9df5eb000639db7",
        "sequenceNumber": "18881790869375471872",
        "mD5OfMessageSystemAttributes": "6f540b6e375dcda1aad2d4aaff28ebf8"
      }
    ]
  },
  "requestID": "b5a386a4-2d4a-5de3-9910-db60fcc368ee",
  "eventID": "20f5ecbe-2b0b-4d0b-a6f7-365bc94c4ca5",
  "readOnly": false,
  "resources": [
    {
      "accountId": "123456789012",
      "ARN": "arn:aws:sqs:ap-southeast-4:123456789012:MyQueue",
      "type": "AWS::SQS::Queue"
    }
  ],
  "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"
  }
}
```

DeleteMessage

The following example shows a CloudTrail data event for `DeleteMessage`.

```
    {
  "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": "2023-11-07T23:58:42Z",
  "eventSource": "sqs.amazonaws.com",
  "eventName": "DeleteMessage",
  "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": {
    "receiptHandle": "AQEBfgYUKTy3dyOAewC4wI3lQZEB3oUDuv8M8FWhObnr3lqRsFBiZ57mmxO1/
dWfdlvGgDW7sRSry6HHxWrNfHItQMUHtWX3a/vEjJ6sWC/5Mf36I/B2HBLCT2zGO/
IZTywxFmUT4HUudWKcGpuZb/Kcl3Fom6hYU8PxxzPxLOKPtFwrVU+G2Spvf/
Tbuyj27h5+AkNxfaAhu/dnvXnAJcDJErGsJTjSS1i6iRzFq+jg6K5Fw6T578QJZcx/
ZLaCyohmj2HaOOktwhbqQc4j+2gKSfxrACgXCu6De5bCtwgtGdhMEh4DtVIQh88qGUCaofQ3t/
eRBIvIFJIa61JCVNWSBqOtELEIfxaHpSvo0c1IEecKDt1IJ08Cij3euLFMIzmUot24IViZt8ntKVAZ6KBLlLedrVlxOhNVc
WG0jfbqz3iBS1T1AD1zJKT7ICIA+edgaYJpOZw4=",
    "queueUrl": "https://sqs.ap-southeast-4.amazonaws.com/123456789012/MyQueue"
  },
  "responseElements": null,
  "requestID": "fbd23ff4-a107-536d-8fcb-623070754bc0",
  "eventID": "9951fed7-365f-4046-bc71-e5bf065a9b47",
  "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"
  }
}
```

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/
hpb4RbXHPZ9tLBVleECbChQIE5NtaDuoZhZPOkTy0eN46EyRR4jXDzE3AlkbPlX1mA9f2fUuTrXx8aeCoCA3I3woNg3fXXA
hlLS94tjAZqV2krc4BaC2pYgjyHWcW019HwIV8T/bjNMIeZoQwOM5V
+o9vHPfewz5QGr5SKpDo7uE7Umyk5n5CJZvcn1efp/
```

```
mrwtaCIb9M7cCQUYcZm2ZmZDnIO9XpGTAi3m2dQ0M83pnNh0nvDfpkHpoa+hX1TrUmxCupCWHJwA8HFJ1O/
CCJsodMNFthLBA9S57dkBZCsw41G8jAmgQ0MkvZ0UL5mg0OFQQd1Yrw0zvthjCgiwdzn0yXoMzxIZMBxkY14E4nVVZ7N5XE
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"
  }
}
```

ChangeMessageVisibility

The following example shows a CloudTrail data event for `ChangeMessageVisibility`.

```
    {
  "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": "ChangeMessageVisibility",
  "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",
    "receiptHandle": "AQEBy+2qnmQQVxcXrEwN7t6dXkjGAllr5DuSpGlvHx9s/vwbwp+RIr3dD6vRvlsU/
lteIulKHBs6DEIR7KL+J3mACfB+RRpRlWPlguiCdLKNKSVpdhkSBDDvkRHfycTHjuszGIebGdl+tYYjPrlz
+DSePmpty0EdhqtorW1xAc0Xf0GZbt0FtkbRFK3ql5lETIHgthBCABoxuOCNvMElz9rYQ9m5OY30Z5YOZvQ/
coPHYl+9HhNV/A6Fs+/d6mVx9v6TomTh5L03wXqtjA8b0gkGftclQh/tJBAxqY/S8YG9OKtY4NDP0SQBtYF/
vCCsCq9+5fiUfiYyvtdHSlwP9AyRotenCGrUKaRFiRhxDm1D6up0UaBs2d8wgHdKFf/5mENTdeqrXQdZfwkFazW1a8ifWJm
+HzhfA9EJcdgWSS72WCMaerydsCxaX+E08B2ubL6oiafMYW4gK0GIRxYZ0+eeXKWy4TxkReW3j7k=",
    "visibilityTimeout": 1272
  },
  "responseElements": null,
  "requestID": "6fbefbde-55d9-5640-98d1-a61a84457f14",
  "eventID": "72275c61-bfc0-4606-934b-a6b7397aef20",
  "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/
abV3szqVWXOLZR4PwX8zZkWVQGNCNnY2q2lGCG586F8QwvrOFYoXNwB8ymd1t77e1PDPknq1Io3JFuzkEsndkkETy4fV1Qq
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

- 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

- Creating CloudWatch alarms for Amazon SQS metrics
- Available CloudWatch metrics for Amazon SQS

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

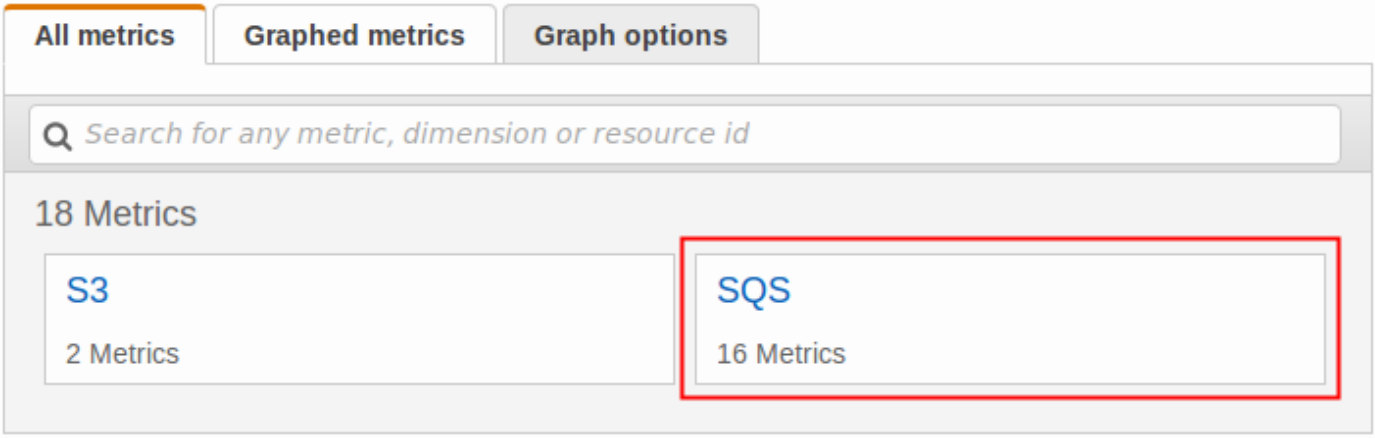
1. Sign in to the Amazon SQS console.
2. In the list of queues, choose (check) the boxes for the queues that you want to access metrics for. You can show metrics for up to 10 queues.
3. Choose the **Monitoring** tab.

   Various graphs are displayed in the **SQS metrics** section.
4. To understand what a particular graph represents, hover over

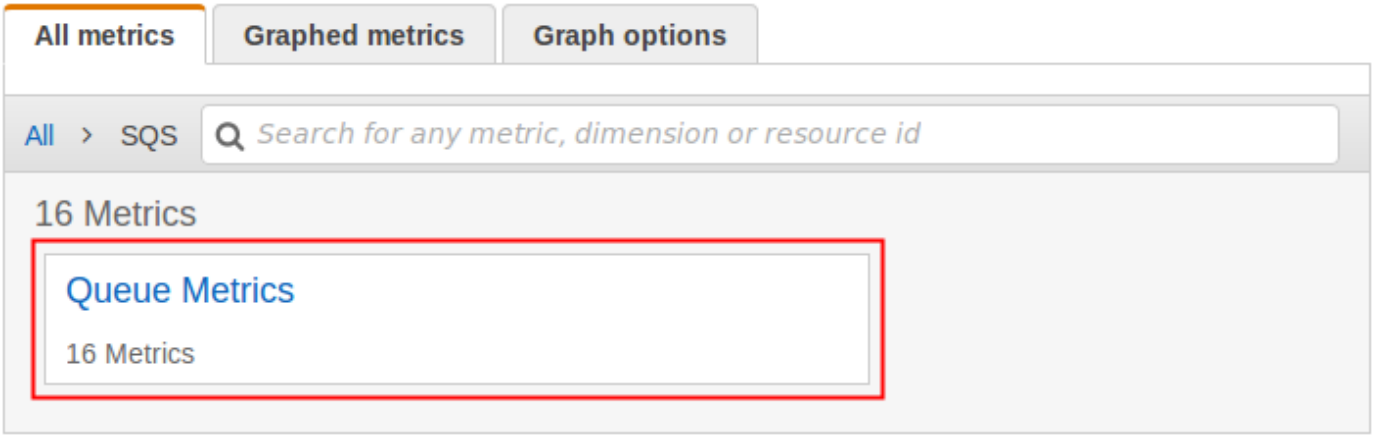
   next to the desired graph, or see Available CloudWatch metrics for Amazon SQS.
5. To change the time range for all of the graphs at the same time, for **Time Range**, choose the desired time range (for example, **Last Hour**).
6. To view additional statistics for an individual graph, choose the graph.
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

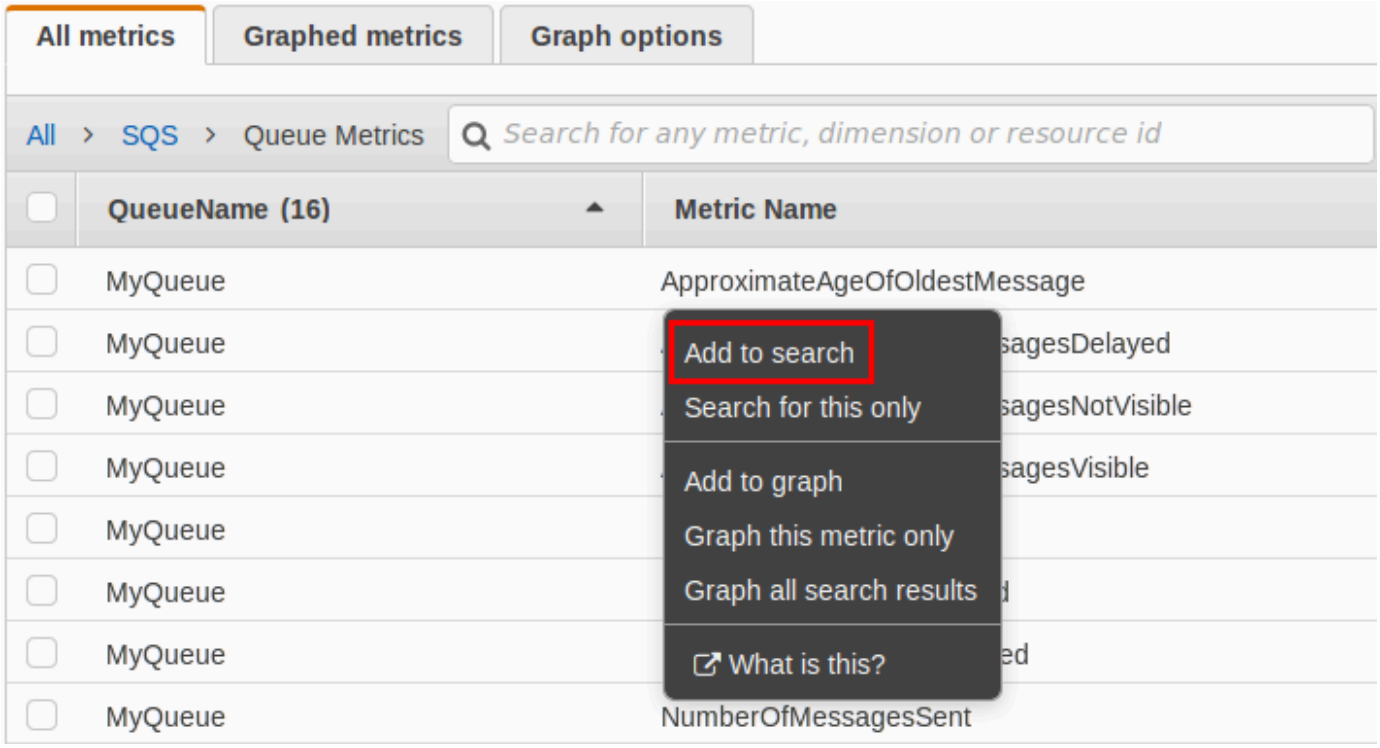
1. Sign in to the CloudWatch console.
2. On the navigation panel, choose **Metrics**.
3. Select the **SQS** metric namespace.



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 standard queues, 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.
> For FIFO queues, the result is exact.

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. •

Metric	Description
	Because a single poison-pill message (received multiple 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)
`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)

Metric	Description
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)
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.

Metric	Description
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
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 message is processed and deleted but

Metric	Description
	you call the DeleteMessage action again using the same receipt handle, a success status is returned, incre asing the value of the NumberOfM essagesDeleted 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)
NumberOfMessagesSent [1]	The number of messages added to a 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
`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.
- 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

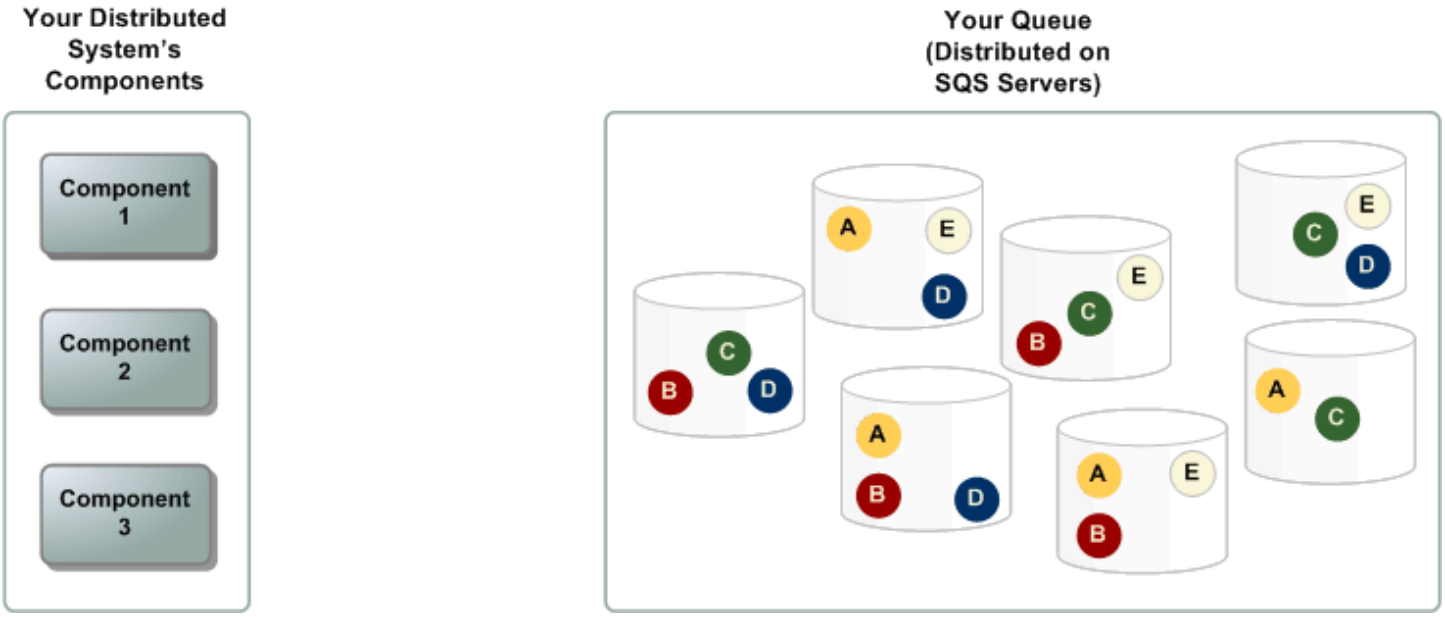
low-latency, high throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single 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.

> **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.

Preventative best practices

The following are preventative security best practices for Amazon SQS.

Topics

- Make sure that queues aren't publicly accessible
- Implement least-privilege access
- Use IAM roles for applications and AWS services which require Amazon SQS access
- 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 and 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.

Working with Amazon SQS APIs

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 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
- Making Query API requests with AWS query protocol
- Authenticating requests
- Amazon SQS batch actions

Making query API requests using AWS JSON protocol

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

- Constructing an endpoint
- Making a POST request
- Interpreting Amazon SQS JSON API responses
- 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 infomation. 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.

Making Query API requests with AWS query protocol

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

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

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);
    }
}
```

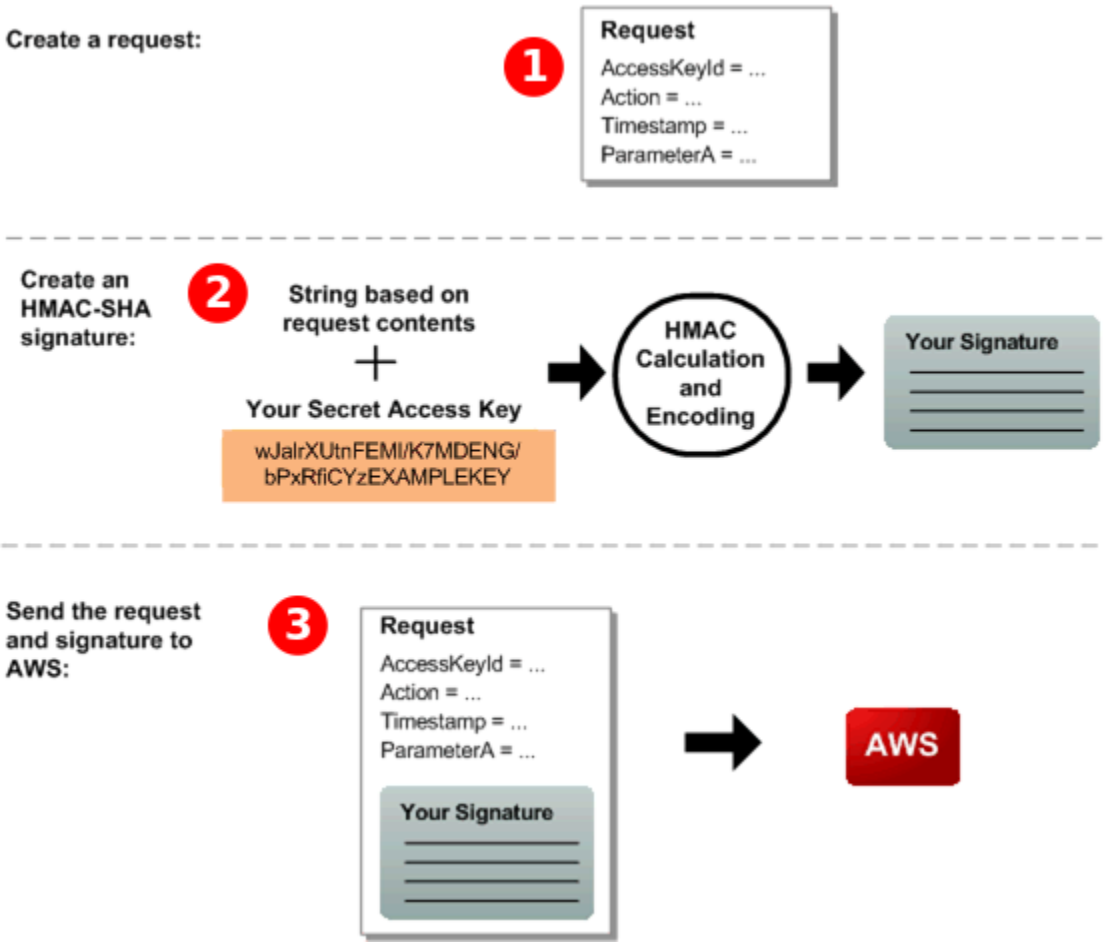
- The **timestamp (or expiration)** of the request. The timestamp that you use in the request must be a `dateTime` object, with the complete date, including hours, minutes, and seconds. For example: `2007-01-31T23:59:59Z` Although this isn't required, we recommend providing the object using the Coordinated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

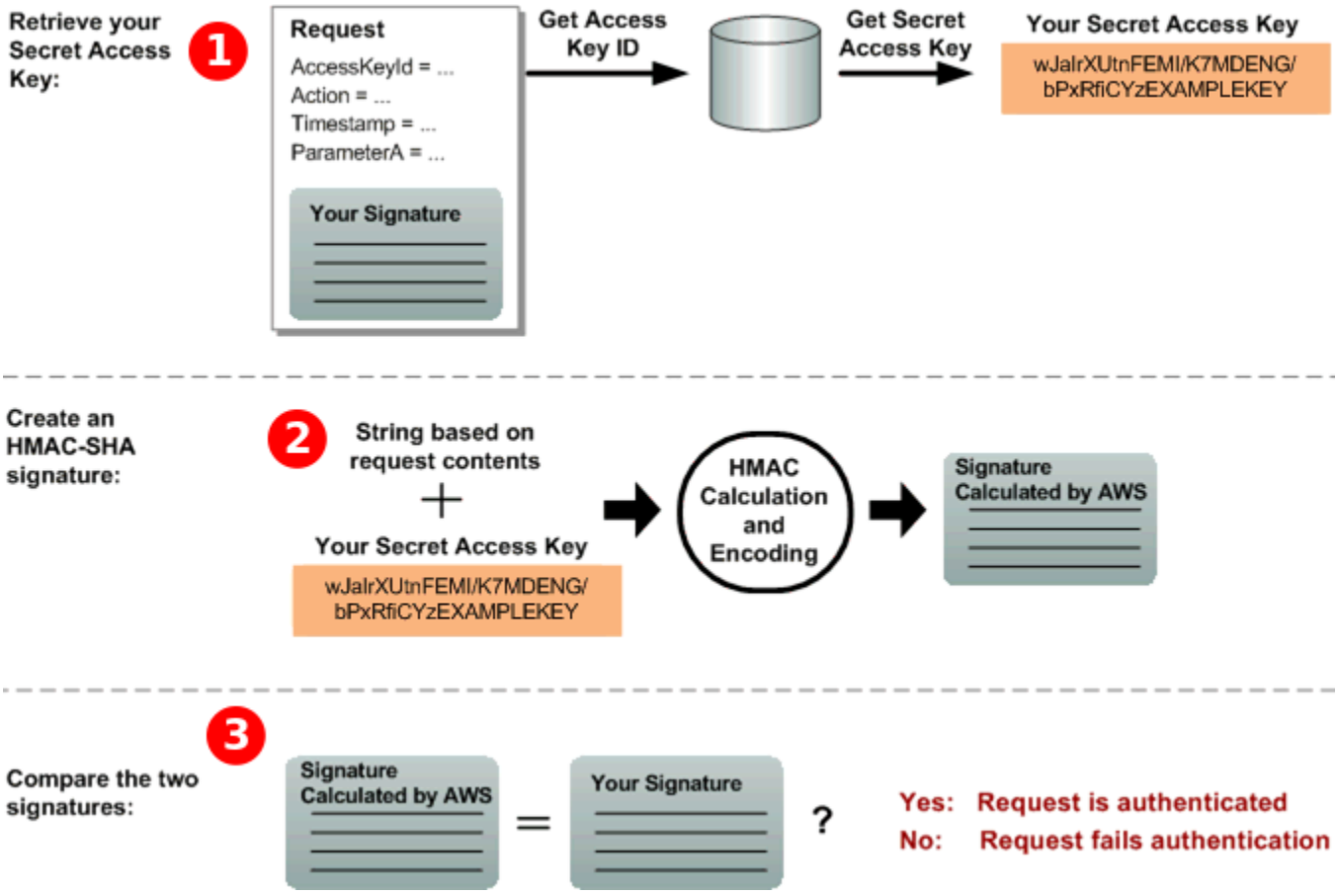
The following is the process you must follow to authenticate AWS requests using an HMAC-SHA request signature.



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

To reduce costs or manipulate up to 10 messages with a single action, you can use the following actions:

- SendMessageBatch
- DeleteMessageBatch
- ChangeMessageVisibilityBatch

You can take advantage of batch functionality using the Query API, or an AWS SDK that supports the Amazon SQS batch actions.

> **Note**
>
> The total size of all messages that you send in a single SendMessageBatch call can't exceed 262,144 bytes (256 KB).
> You can't set permissions for SendMessageBatch, DeleteMessageBatch, or ChangeMessageVisibilityBatch explicitly. Setting permissions for SendMessage, DeleteMessage, or ChangeMessageVisibility sets permissions for the corresponding batch versions of the actions.
> The Amazon SQS console doesn't support batch actions.

Topics

Enabling client-side buffering and request batching

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.

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

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.

Parameter	Default value	Description
`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. 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.

Parameter	Default value	Description
`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.
`maxBatchSizeBytes`	256 KB	The maximum size of a message batch, in bytes, that the client attempts to send to Amazon SQS. **Note** 256 KB 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 prefetching 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 prefetching 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

Amazon SQS queues can deliver very high throughput. For information on throughput quotas, see Quotas related to messages.

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 2010-2022 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.

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
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. This action is associated with the `ListMessageMoveTasks` API operation.	June 7, 2023
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,	October 20, 2022

	plus FIFO high throughput document optimization.	
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-effective, 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

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.