Amazon Simple Queue Service

Developer Guide

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 (p. 82) and cost allocation tags (p. 80). It provides a generic web services API and it can be accessed by any programming language that the AWS SDK supports.

Amazon SQS supports both standard (p. 66) and FIFO queues (p. 69). For more information, see What Type of Queue Do I Need? (p. 2)

Topics
- What Are the Main Benefits of Amazon SQS? (p. 1)
- How Is Amazon SQS Different from Amazon MQ or Amazon SNS? (p. 1)
- What Type of Queue Do I Need? (p. 2)
- How Can I Get Started with Amazon SQS? (p. 2)
- We Want to Hear from You (p. 3)

What Are the Main Benefits of Amazon SQS?
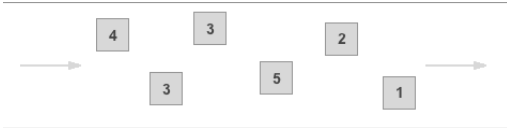
- **Security** – You control (p. 135) who can send messages to and receive messages from an Amazon SQS queue.

  Server-side encryption (SSE) (p. 161) lets you transmit sensitive data by protecting the contents of messages in queues using keys managed in AWS Key Management Service (AWS KMS).
- **Durability** – To ensure the safety of your messages, Amazon SQS stores them on multiple servers. Standard queues support at-least-once message delivery (p. 66), and FIFO queues support exactly-once message processing (p. 72).
- **Availability** – Amazon SQS uses redundant infrastructure (p. 64) to provide highly-concurrent access to messages and high availability for producing and consuming messages.
- **Scalability** – Amazon SQS can process each buffered request (p. 175) 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 (p. 87). You can store the contents of messages larger than 256 KB using Amazon Simple Storage Service (Amazon S3) (p. 88) or Amazon DynamoDB, with Amazon SQS holding a pointer to the Amazon S3 object, or you can split a large message into smaller messages.

How Is Amazon SQS Different from Amazon MQ or Amazon SNS?

Amazon SQS and Amazon SNS are queue and topic services that are highly scalable, simple to use, and don't require you to set up message brokers. We recommend these services for new applications that can benefit from nearly unlimited scalability and simple APIs.

Amazon MQ is a managed message broker service that provides compatibility with many popular message brokers. We recommend Amazon MQ for migrating applications from existing message brokers that rely on compatibility with APIs such as JMS or protocols such as AMQP, MQTT, OpenWire, and STOMP.

What Type of Queue Do I Need?

Standard Queue	FIFO Queue
Available in all regions. **Unlimited Throughput** – Standard queues support a nearly unlimited number of transactions per second (TPS) per action. **At-Least-Once Delivery** – A message is delivered at least once, but occasionally more than one copy of a message is delivered. **Best-Effort Ordering** – Occasionally, messages might be delivered in an order different from which they were sent.	Available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions. **High Throughput** – By default, FIFO queues support up to 3,000 messages per second with batching (p. 174). To request a limit increase, file a support request. FIFO queues support up to 300 messages per second (300 send, receive, or delete operations per second) without batching. **Exactly-Once Processing** – A message is delivered once and remains available until a consumer processes and deletes it. Duplicates aren't introduced into the queue. **First-In-First-Out Delivery** – The order in which messages are sent and received is strictly preserved.
Send data between applications when the throughput is important, for example: • Decouple live user requests from intensive background work: let users upload media while resizing or encoding it. • 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: • Ensure that user-entered commands are executed 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.

How Can I Get Started with Amazon SQS?

- To create your first queue with Amazon SQS and send, receive, and delete a message, see Getting Started with Amazon SQS (p. 8).
- To discover the functionality and architecture of Amazon SQS, see How Amazon SQS Works (p. 64).
- To find out the guidelines and caveats that will help you make the most of Amazon SQS, see Best Practices for Amazon SQS (p. 115).

- To learn about Amazon SQS actions, see the *Amazon Simple Queue Service API Reference*.
- To learn about Amazon SQS AWS CLI commands, see Amazon SQS in the *AWS CLI Command Reference*.

We Want to Hear from You

We welcome your feedback. To contact us, visit the Amazon SQS Discussion Forum.

New and Frequently Viewed Amazon SQS Topics

Latest update: October 8, 2018

Amazon Simple Queue Service Developer Guide

New Topics on Service Features	Most Frequently Viewed Topics
1. Creating CloudWatch Alarms for Amazon SQS Metrics (p. 126)	1. Amazon SQS Visibility Timeout (p. 85)
2. Configuring Messages Arriving in an Amazon SQS Queue to Trigger a Lambda Function (p. 43)	2. Amazon SQS Dead-Letter Queues (p. 82)
3. Configuring Visibility Timeout for an Amazon SQS Queue (p. 57)	3. Amazon SQS FIFO (First-In-First-Out) Queues (p. 69)
4. Configuring Long Polling for an Amazon SQS queue (p. 51)	4. Amazon SQS Long Polling (p. 81)
5. Configuring an Amazon SQS Dead-Letter Queue (p. 53)	5. Amazon SQS Limits (p. 121)
6. Sending a Message with Attributes to an Amazon SQS Queue (p. 30)	6. How Amazon SQS Works (p. 64)
7. Amazon SQS Message Attributes (p. 77)	7. Amazon SQS Message Attributes (p. 77)
8. Enable Compatibility between AWS Services and Queues with SSE (p. 166)	8. Amazon SQS Standard Queues (p. 66)
9. Working with JMS and Amazon SQS (p. 92)	9. Getting Started with Amazon SQS (p. 8)
10. Amazon SQS Cost Allocation Tags (p. 80)	10. Using Custom Policies with the Amazon SQS Access Policy Language (p. 149)

Amazon Simple Queue Service API Reference

New Topics on Service Features	Most Frequently Viewed Topics
1. `SendMessageBatch` (Message Timers)	1. `ReceiveMessage`
2. `SendMessage` (Message Timers)	2. `SendMessage`
3. `CreateQueue` (SSE, FIFO, Delay Queues)	3. `GetQueueAttributes`
4. `SetQueueAttributes` (Long Polling, Delay Queues, Dead Letter Queues)	4. `SendMessageBatch`
5. `ReceiveMessage` (FIFO, Long Polling)	5. `DeleteMessage`
6. `ListQueueTags`	6. `ChangeMessageVisibility`
7. `UntagQueue`	7. `CreateQueue`
8. `TagQueue`	8. `SetQueueAttributes`
9. `SetQueueAttributes` (SSE, FIFO)	9. `GetQueueUrl`
10. `GetQueueAttributes` (SSE, FIFO)	10. `AddPermission`

Setting Up Amazon SQS

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

Step 1: Create an AWS Account

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.

To set up a new account

1. Open https://aws.amazon.com/, and then choose **Create an AWS Account**.

    Note
    If you previously signed in to the AWS Management Console using AWS account root user credentials, choose **Sign in to a different account**. If you previously signed in to the console using IAM credentials, choose **Sign-in using root account credentials**. Then choose **Create a new AWS account**.

2. Follow the online instructions.

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

Step 2: Create an IAM User

To create an IAM user for yourself and add the user to an Administrators group

1. Use your AWS account email address and password to sign in as the *AWS account root user* to the IAM console at https://console.aws.amazon.com/iam/.

    Note
    We strongly recommend that you adhere to the best practice of using the `Administrator` IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.

2. In the navigation pane of the console, choose **Users**, and then choose **Add user**.
3. For **User name**, type `Administrator`.
4. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to create a new password the next time the user signs in.
5. Choose **Next: Permissions**.
6. On the **Set permissions** page, choose **Add user to group**.
7. Choose **Create group**.
8. In the **Create group** dialog box, for **Group name** type `Administrators`.

9. For **Filter policies**, select the check box for **AWS managed - job function**.
10. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.
11. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
12. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to specific AWS resources, go to Access Management and Example Policies.

Step 3: Get Your Access Key ID and Secret Access Key

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.

To get the access key ID and secret access key for an IAM user

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. We recommend that you use IAM access keys instead of AWS account root user access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

The only time that you can view or download the secret access keys is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see Permissions Required to Access IAM Resources in the *IAM User Guide*.

1. Open the IAM console.
2. In the navigation pane of the console, choose **Users**.
3. Choose your IAM user name (not the check box).
4. Choose the **Security credentials** tab and then choose **Create access key**.
5. To see the new access key, choose **Show**. Your credentials will look something like this:

    - Access key ID: AKIAIOSFODNN7EXAMPLE
    - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location.

    Keep the keys confidential in order to protect your AWS account, and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

Related topics

- What Is IAM? in the *IAM User Guide*
- AWS Security Credentials in *AWS General Reference*

Step 4: Get Ready to Use the Example Code

This guide shows how to work with Amazon SQS using the AWS Management Console and using Java. If you want to use the example code, you must install the Java Standard Edition Development Kit and make some configuration changes to the example code.

You can write code in other programming languages. For more information, see the documentation of the AWS SDKs.

> **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

Now that you're prepared for working with Amazon SQS, can get started (p. 8) with managing Amazon SQS queues and messages using the AWS Management Console. You can also try the more advanced Amazon SQS tutorials (p. 15).

Getting Started with Amazon SQS

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

Prerequisites

Before you begin, complete the steps in Setting Up Amazon SQS (p. 5).

Step 1: Create a Queue

The first and most common Amazon SQS task is creating queues. In this tutorial you'll learn how to create and configure a queue.

1. Sign in to the Amazon SQS console.
2. Choose **Create New Queue.**
3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

   Note
   The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions.

4. **Standard** is selected by default. Choose **FIFO**.
5. To create your queue with the default parameters, choose **Quick-Create Queue**.

   Your new queue is created and selected in the queue list.

   Note
   When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS..

   The **Queue Type** column helps you distinguish standard queues from FIFO queues at a glance. For a FIFO queue, the **Content-Based Deduplication** column displays whether you have enabled exactly-once processing (p. 72).

	Name	Queue Type ▾	Content-Based Deduplication ▾	Messages Available ▾	Messages in Flight ▾	Created	▾
☐	MyQueue	Standard Queue	N/A	0	0	2016-12-07 13:42:47 GMT-08:00	
☑	MyQueue.fifo	FIFO Queue	Disabled	0	0	2016-12-07 13:42:55 GMT-08:00	

   Your queue's **Name**, **URL**, and **ARN** are displayed on the **Details** tab.

   Name: MyQueue.fifo
   URL: https://sqs.us-west-2.amazonaws.com/   /MyQueue.fifo
   ARN: arn:aws:sqs:us-west-2:   :MyQueue.fifo

Step 2: Send a Message

After you create your queue, you can send a message to it. The following example shows sending a message to an existing queue.

1. From the queue list, select the queue that you've created.

2. From **Queue Actions**, select **Send a Message**.

The **Send a Message to** *QueueName* dialog box is displayed.

The following example shows the **Message Group ID** and **Message Deduplication ID** parameters specific to FIFO queues (content-based deduplication (p. 72) is disabled).

3. To send a message to a FIFO queue, type the **Message Body**, the **Message Group ID** `MyMessageGroupId1234567890`, and the **Message Deduplication ID** `MyMessageDeduplicationId1234567890`, and then choose **Send Message**. For more information, see FIFO Delivery Logic (p. 71).

   > **Note**
   > The message group ID is always required. However, if content-based deduplication is enabled, the message deduplication ID is optional.

Your message is sent and the **Send a Message to** *QueueName* dialog box is displayed, showing the attributes of the sent message.

The following example shows the **Sequence Number** attribute specific to FIFO queues.
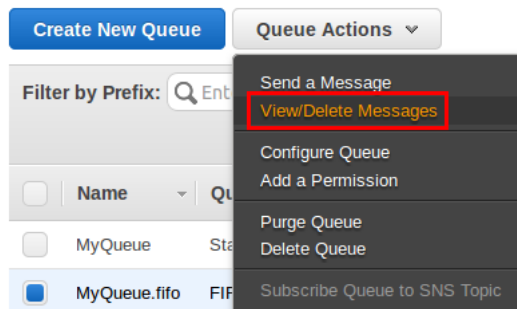
4. Choose **Close**.

Step 3: Receive and Delete Your Message

After you send a message into a queue, you can consume it (retrieve it from the queue). When you request a message from a queue, you can't specify which message to get. Instead, you specify the maximum number of messages (up to 10) that you want to get.

In this tutorial you'll learn how to receive and delete a message.

1. From the queue list, select the queue that you have created.
2. From **Queue Actions**, select **View/Delete Messages**.

The **View/Delete Messages in** *QueueName* dialog box is displayed.

> **Note**
> The first time you take this action, an information screen is displayed. To hide the screen, check the **Don't show this again** checkbox.

3. Choose **Start Polling for messages.**

Amazon SQS begins to poll the messages in the queue. The dialog box displays a message from the queue. A progress bar at the bottom of the dialog box displays the status of the message's visibility timeout.

The following example shows the **Message Group ID**, **Message Deduplication ID**, and **Sequence Number** columns specific to FIFO queues.

> **Note**
> When the progress bar is filled in, the visibility timeout (p. 85) expires and the message becomes visible to consumers.

4. *Before* the visibility timeout expires, select the message that you want to delete and then choose **Delete *1* Message**.

5. In the **Delete Messages** dialog box, confirm that the message you want to delete is checked and choose **Yes, Delete Checked Messages**.

The selected message is deleted.

6. Select **Close**.

Step 4: Delete Your Queue

If you don't use an Amazon SQS queue (and don't foresee using it in the near future), it is a best practice to delete it from Amazon SQS. In this tutorial you'll learn how to delete a queue.

1. From the queue list, select the queue that you have created.

2. From **Queue Actions**, select **Delete Queue**.
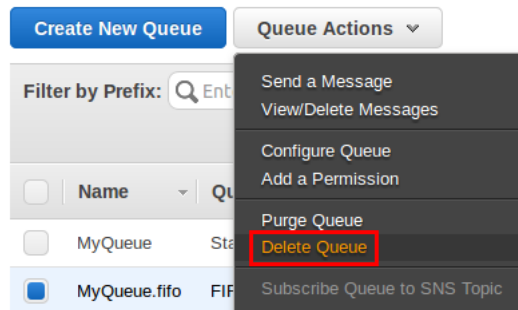
The **Delete Queues** dialog box is displayed.
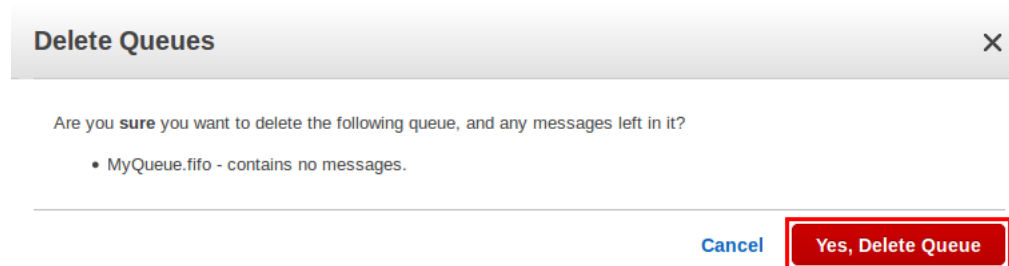
3. Choose **Yes, Delete Queue**.

   The queue is deleted.

Next Steps

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:

- Enable server-side encryption (SSE) for a new queue (p. 19) or configure SSE for an existing queue. (p. 48)
- Add permissions to a queue. (p. 23)
- Add, update, or remove tags for a queue. (p. 26)
- Configure long polling for a queue. (p. 51)
- Send a message with attributes. (p. 30)
- Send a message with a timer. (p. 35)
- Configure a dead-letter queue. (p. 53)
- Configure visibility timeout for a queue. (p. 57)
- Configure a delay queue. (p. 60)
- Subscribe a queue to an Amazon SNS topic. (p. 41)
- Configure messages arriving in a queue to trigger a Lambda function. (p. 43)
- Purge a queue. (p. 45)
- Learn more about Amazon SQS workflows and processes: Read How Queues Work (p. 64), Best Practices (p. 115), and Limits (p. 121). You can also explore the Amazon SQS Articles & Tutorials. If you ever have any questions, browse the Amazon SQS FAQs or participate in the Amazon SQS Developer Forums.
- Learn how to interact with Amazon SQS programmatically: Read Working with APIs (p. 169) 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 Amazon SQS Monitoring, Logging, and Automating (p. 124) section.
- Learn about protecting your data and access to it in the Security (p. 135) section.

Amazon SQS Tutorials

This guide shows how to work with Amazon SQS using the AWS Management Console and using Java. If you want to use the example code, you must install the Java Standard Edition Development Kit and make some configuration changes to the example code.

You can write code in other programming languages. For more information, see the documentation of the AWS SDKs.

> **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*.

Topics

Tutorials: Creating Amazon SQS Queues

The following tutorials show how to create Amazon SQS queues in various ways.

Topics

Tutorial: Creating an Amazon SQS Queue

The first and most common Amazon SQS task is creating queues. In this tutorial you'll learn how to create and configure a queue.

Topics

AWS Management Console

1. Sign in to the Amazon SQS console.

2. Choose **Create New Queue.**

3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

   Note
   The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions.

4. **Standard** is selected by default. Choose **FIFO**.

5. Create your queue.

   - To create your queue with the default parameters, choose **Quick-Create Queue**.
   - To configure your queue's parameters, choose **Configure Queue**. When you finish configuring the parameters, choose **Create Queue**. For more information about creating a queue with SSE, see Creating an Amazon SQS Queue with SSE (p. 19).

   The following example shows the **Content-Based Deduplication** parameter specific to FIFO queues.

   Your new queue is created and selected in the queue list.

   Note
   When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS.

   The **Queue Type** column helps you distinguish standard queues from FIFO queues at a glance. For a FIFO queue, the **Content-Based Deduplication** column displays whether you have enabled exactly-once processing (p. 72).

	Name	Queue Type ▾	Content-Based Deduplication ▾	Messages Available ▾	Messages in Flight ▾	Created
☐	MyQueue	Standard Queue	N/A	0	0	2016-12-07 13:42:47 GMT-08:00
☑	MyQueue.fifo	FIFO Queue	Disabled	0	0	2016-12-07 13:42:55 GMT-08:00

Your queue's **Name**, **URL**, and **ARN** are displayed on the **Details** tab.

Name: MyQueue.fifo

URL: https://sqs.us-west-2.amazonaws.com/   /MyQueue.fifo

ARN: arn:aws:sqs:us-west-2:   :MyQueue.fifo

AWS SDK for Java

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

To create a standard queue

1. Copy the example program (p. 67).

   The following section of the code creates the `MyQueue` queue:

   ```
   // Create a queue
   System.out.println("Creating a new SQS queue called MyQueue.\n");
   final CreateQueueRequest createQueueRequest = new CreateQueueRequest("MyQueue");
   final String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
   ```

2. Compile and run the example.

   The queue is created.

To create a FIFO queue

1. Copy the example program (p. 73).

   The following section of the code creates the `MyFifoQueue.fifo` queue:

   ```
   // Create a FIFO queue
   System.out.println("Creating a new Amazon SQS FIFO queue called " + "MyFifoQueue.fifo.
   \n");
   final Map<String, String> attributes = new HashMap<String, String>();

   // A FIFO queue must have the FifoQueue attribute set to True
   attributes.put("FifoQueue", "true");

   // If the user doesn't provide a MessageDeduplicationId, generate a
    MessageDeduplicationId based on the content.
   attributes.put("ContentBasedDeduplication", "true");

   // The FIFO queue name must end with the .fifo suffix
   final CreateQueueRequest createQueueRequest = new
    CreateQueueRequest("MyFifoQueue.fifo")
           .withAttributes(attributes);
   final String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();
   ```

2. Compile and run the example.

   The queue is created.

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*.

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 (p. 72).

    > **Note**
    > The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions.

```json
{
    "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 **Select Template** page, choose **Upload a template to Amazon S3**, 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.

	Stack Name	Created Time	Status	Description
☑	MyQueue	2017-02-20 11:39:47 UTC-0800	CREATE_COMPLETE	

Filter: Active ▾ By Stack Name — Showing 1 stack

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.

MyQueue

Stack name:	MyQueue
Stack ID:	arn:aws:cloudformation:us-east-2: :stack/MyQueue/
Status:	CREATE_COMPLETE
Status reason:	
IAM Role:	
Description:	

▼ Outputs

Key	Value	Description	Export Name
QueueURL	https://sqs.us-east-2.amazonaws.com/ /MyQueue-	URL of the queue	
QueueARN	arn:aws:sqs:us-east-2: :MyQueue-	ARN of the queue	

Tutorial: Creating an Amazon SQS Queue with Server-Side Encryption (SSE)

Server-side encryption (SSE) for Amazon SQS is available in the AWS GovCloud (US-West) Region, in addition to all commercial regions where Amazon SQS is available, except for the China Regions. You can enable SSE for a queue to protect its data. For more information about using SSE, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161).

Important
All requests to queues with SSE enabled must use HTTPS and Signature Version 4.

In this tutorial you'll learn how to create an Amazon SQS queue with SSE enabled. Although the example uses a FIFO queue, SSE works with both standard and FIFO queues.

Topics
- AWS Management Console (p. 19)
- AWS SDK for Java (p. 21)

AWS Management Console

1. Sign in to the Amazon SQS console.

2. Choose **Create New Queue.**

3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

   Note
   The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions.

4. **Standard** is selected by default. Choose **FIFO**.

5. Choose **Configure Queue**, and then choose **Use SSE**.

6. Specify the customer master key (CMK) ID. For more information, see Key Terms (p. 163).

   For each CMK type, the **Description**, **Account**, and **Key ARN** of the CMK are displayed.

   Important
   If you aren't the owner of the CMK, or if you log in with an account that doesn't have the `kms:ListAliases` and `kms:DescribeKey` permissions, you won't be able to view information about the CMK on the Amazon SQS console.
   Ask the owner of the CMK to grant you these permissions. For more information, see the AWS KMS API Permissions: Actions and Resources Reference in the *AWS Key Management Service Developer Guide*.

   - The AWS managed CMK for Amazon SQS is selected by default.

     Note
     Keep the following in mind:

     - If you don't specify a custom CMK, Amazon SQS uses the AWS managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the *AWS Key Management Service Developer Guide*.

     - The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS managed CMK 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 CMK for Amazon SQS.

   - To use a custom CMK from your AWS account, select it from the list.

     Note
     For instructions on creating custom CMKs, see Creating Keys in the *AWS Key Management Service Developer Guide*.

   - To use a custom CMK ARN from your AWS account or from another AWS account, select **Enter an existing CMK ARN** from the list and type or copy the CMK.

7. (Optional) For **Data key reuse period**, specify a value between 1 minute and 24 hours. The default is 5 minutes. For more information, see How Does the Data Key Reuse Period Work? (p. 163).

Data Key Reuse Period ⓘ [5] [minutes ▾] This value must be between 1 minute and 24 hours.

8. Choose **Create Queue**.

Your new queue is created with SSE. The encryption status, alias of the CMK, **Description**, **Account**, **Key ARN**, and the **Data Key Reuse Period** are displayed on the **Encryption** tab.

Server-side encryption (SSE) is enabled. SSE lets you protect the contents of messages in Amazon SQS queues using keys managed in the AWS Key Management Service (AWS KMS). Learn more.

To modify the SSE parameters, choose **Queue Actions, Configure Queue**.

AWS KMS Customer Master Key (CMK) ⓘ	alias/aws/sqs
Description	Default master key that protects my SQS messages when no other key is defined
Account	
Key ARN	
Data Key Reuse Period ⓘ	5 minutes

AWS SDK for Java

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

Before you can use SSE, you must configure AWS KMS key policies to allow encryption of queues and encryption and decryption of messages. You must also ensure that the key policies of the customer master key (CMK) allow the necessary permissions. For more information, see What AWS KMS Permissions Do I Need to Use SSE for Amazon SQS? (p. 165)

1. Obtain the customer master key (CMK) ID. For more information, see Key Terms (p. 163).

    Note
    Keep the following in mind:

    - If you don't specify a custom CMK, Amazon SQS uses the AWS managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the *AWS Key Management Service Developer Guide*.
    - The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS managed CMK 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 CMK for Amazon SQS.

2. To enable server-side encryption, specify the CMK ID by setting the `KmsMasterKeyId` attribute of the `CreateQueue` or `SetQueueAttributes` action.

    The following code example creates a new queue with SSE using the AWS managed CMK for Amazon SQS:

```
final AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
final CreateQueueRequest createRequest = new CreateQueueRequest("MyQueue");
final Map<String, String> attributes = new HashMap<String, String>();

// Enable server-side encryption by specifying the alias ARN of the
// AWS managed CMK for Amazon SQS.
final String kmsMasterKeyAlias = "arn:aws:kms:us-east-2:123456789012:alias/aws/sqs";
attributes.put("KmsMasterKeyId", kmsMasterKeyAlias);

// (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse
attributes.put("KmsDataKeyReusePeriodSeconds", "60");
```

```
final CreateQueueResult createResult = client.createQueue(createRequest);
```

The following code example creates a new queue with SSE using a custom CMK:

```
final AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
final CreateQueueRequest createRequest = new CreateQueueRequest("MyQueue");
final Map<String, String> attributes = new HashMap<String, String>();

// Enable server-side encryption by specifying the alias ARN of the custom CMK.
final String kmsMasterKeyAlias = "arn:aws:kms:us-east-2:123456789012:alias/MyAlias";
attributes.put("KmsMasterKeyId", kmsMasterKeyAlias);

// (Optional) Specify 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.
attributes.put("KmsDataKeyReusePeriodSeconds", "864000");

final CreateQueueResult createResult = client.createQueue(createRequest);
```

3. (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse a data key (p. 163) to encrypt or decrypt messages before calling AWS KMS again. Set the `KmsDataKeyReusePeriodSeconds` attribute of the `CreateQueue` or `SetQueueAttributes` action. Possible values may be between 60 seconds (1 minute) and 86,400 seconds (24 hours). If you don't specify a value, the default value of 300 seconds (5 minutes) is used.

The first code example above sets the data key reuse time period to 60 seconds (1 minute). The second code example sets it to 86,400 seconds (24 hours). The following code example sets the data key reuse period to 60 seconds (1 minute):

```
// (Optional) Specify 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.
attributes.put("KmsDataKeyReusePeriodSeconds", "60");
```

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

To retrieve the CMK ID or the data key reuse period for a particular queue, use the `KmsMasterKeyId` and `KmsDataKeyReusePeriodSeconds` attributes of the `GetQueueAttributes` action.

For information about switching a queue to a different CMK with the same alias, see Updating an Alias in the *AWS Key Management Service Developer Guide*.

Tutorial: Listing All Amazon SQS Queues in a Region

When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS. In this tutorial you'll learn how to confirm your queue's existence by listing all queues in the current region.

Topics

- AWS Management Console (p. 23)
- AWS SDK for Java (p. 23)

AWS Management Console

1. Sign in to the Amazon SQS console.

2. Your queues in the current region are listed.

   The **Queue Type** column helps you distinguish standard queues from FIFO queues at a glance. For a FIFO queue, the **Content-Based Deduplication** column displays whether you have enabled exactly-once processing (p. 72).

	Name	Queue Type	Content-Based Deduplication	Messages Available	Messages in Flight	Created	
☐	MyQueue	Standard Queue	N/A	0	0	2016-12-07 13:42:47 GMT-08:00	
☑	MyQueue.fifo	FIFO Queue	Disabled	0	0	2016-12-07 13:42:55 GMT-08:00	

   Your queue's **Name**, **URL**, and **ARN** are displayed on the **Details** tab.

   Name: MyQueue.fifo

   URL: https://sqs.us-west-2.amazonaws.com/▮▮▮▮▮▮/MyQueue.fifo

   ARN: arn:aws:sqs:us-west-2:▮▮▮▮▮▮:MyQueue.fifo

AWS SDK for Java

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

> **Note**
> This action is identical for standard and FIFO queues.

1. Copy the standard queue example program (p. 67) or the FIFO queue example program (p. 73).

   The following section of the code list all queues in the current region:

   ```
   // List queues
   System.out.println("Listing all queues in your account.\n");
   for (final String queueUrl : sqs.listQueues().getQueueUrls()) {
       System.out.println("  QueueUrl: " + queueUrl);
   }
   System.out.println();
   ```

2. Compile and run the example.

   All queues in the current region created using API version 2012-11-05 are listed. The response include the following items:

   - The unique *queue URL*.

   - The *request ID* that Amazon SQS assigned to your request.

Tutorial: Adding Permissions to an Amazon SQS Queue

You can specify to whom you allow (or explicitly deny) the ability to interact with your queue in specific ways by adding permissions to a queue. The following example shows how to add the permission for anyone to get a queue's URL.

Note
An Amazon SQS policy can have a maximum of 7 actions.

AWS Management Console

1. Sign in to the Amazon SQS console.

2. From the queue list, select a queue.
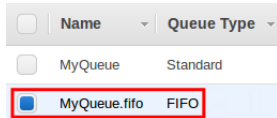
3. From **Queue Actions**, select **Add a Permission**.
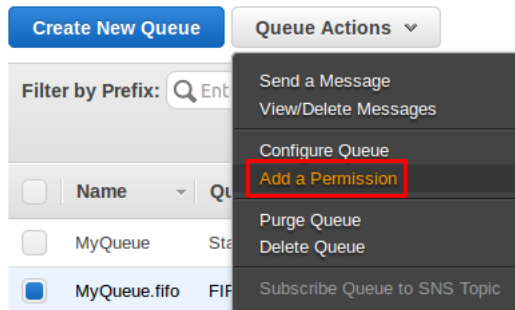
The **Add a Permission** dialog box is displayed.

4. In this example, you allow anyone to get the queue's URL:

❶ Ensure that next to **Effect**, **Allow** is selected.

❷ Next to **Principal**, check the **Everybody** box.

❸ From the **Actions** drop-down list, select **GetQueueUrl** box.

❹ Choose **Add Permission**.

The permission is added to the queue.

Your queues's policy **Effect**, **Principals**, **Actions**, and **Conditions** are displayed on your queue's **Permissions** tab.

Tutorial: Adding, Updating, and Removing Cost Allocation Tags for an Amazon SQS Queue

You can add cost allocation tags to your Amazon SQS queues to help organize and identify them. The following example show how to add, update, and remove tags for a queue. For a more information, see Amazon SQS Cost Allocation Tags (p. 80).

Topics

AWS Management Console

The following steps assume that you already created an Amazon SQS queue (p. 15).

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.
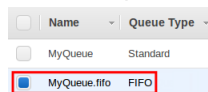
3. Choose the **Tags** tab.

   The tags added to the queue are listed.

4. Choose **Add/Edit Tags**.
5. Modify queue tags:

   - To add a tag, choose **Add New Tag**, enter a **Key** and **Value**, and then choose **Apply Changes**.
   - To update a tag, change its **Key** and **Value** and then choose **Apply Changes**.
   - To remove a tag, choose ✖ next to a key-value pair and then choose **Apply Changes**.

   The queue tag changes are applied.

AWS SDK for Java

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

To add, update, and remove tags from a queue

1. Copy the example program for a standard queue (p. 67) or a FIFO queue (p. 73).
2. To list the tags added to a queue, add the following code which uses the `ListQueueTags` action:

```
final ListQueueTagsRequest listQueueTagsRequest = new ListQueueTagsRequest(queueUrl);
final ListQueueTagsResult listQueueTagsResult = SQSClientFactory.newSQSClient()
        .listQueueTags(listQueueTagsRequest);
System.out.println(String.format("ListQueueTags: \tTags for queue %s are %s.\n",
        QUEUE_NAME, listQueueTagsResult.getTags()));
```

3. To add or update the values of the queue's tags using the tag's key, add the following code which uses the `TagQueue` action:

```
final Map<String, String> addedTags = new HashMap<>();
addedTags.put("Team", "Development");
addedTags.put("Priority", "Beta");
addedTags.put("Accounting ID", "456def");
final TagQueueRequest tagQueueRequest = new TagQueueRequest(queueUrl, addedTags);

System.out.println(String.format("TagQueue: \t\tAdd tags %s to queue %s.\n", addedTags,
 QUEUE_NAME));
SQSClientFactory.newSQSClient().tagQueue(tagQueueRequest);
```

4. To remove a tag from the queue using the tag's key, add the following code which uses the `UntagQueue` action:

```
final List<String> tagKeys = Arrays.asList("Accounting ID");
final UntagQueueRequest untagQueueRequest = new UntagQueueRequest(queueUrl, tagKeys);
System.out.println(String.format("UntagQueue: \tRemove tags %s from queue %s.\n",
 tagKeys, QUEUE_NAME));
SQSClientFactory.newSQSClient().untagQueue(untagQueueRequest);
```

5. Compile and run your program.

    The existing tags are listed, three are updated, and one tag is removed from the queue.

Tutorials: Sending Messages to Amazon SQS Queues

The following tutorials show how to send messages to Amazon SQS queues in various ways.

Topics

Tutorial: Sending a Message to an Amazon SQS Queue

After you create your queue, you can send a message to it. The following example shows sending a message to an existing queue.

Topics

AWS Management Console

1. Sign in to the Amazon SQS console.

2. From the queue list, select a queue.

Name	Queue Type
MyQueue	Standard
MyQueue.fifo	FIFO

3. From **Queue Actions**, select **Send a Message**.

   The **Send a Message to _QueueName_** dialog box is displayed.

   The following example shows the **Message Group ID** and **Message Deduplication ID** parameters specific to FIFO queues (content-based deduplication (p. 72) is disabled).

4. To send a message to a FIFO queue, type the **Message Body**, the **Message Group ID** `MyMessageGroupId1234567890`, and the **Message Deduplication ID** `MyMessageDeduplicationId1234567890`, and then choose **Send Message**. For more information, see FIFO Delivery Logic (p. 71).

Note
The message group ID is always required. However, if content-based deduplication is enabled, the message deduplication ID is optional.

Message Group ID ⓘ	MyMessageGroupId1234567890
Message Deduplication ID ⓘ	MyMessageDeduplicationId1234567890

Cancel   **Send Message**

Your message is sent and the **Send a Message to** *QueueName* dialog box is displayed, showing the attributes of the sent message.

The following example shows the **Sequence Number** attribute specific to FIFO queues.

Send a Message to MyQueue.fifo   ✕

Your message has been sent and is ready to be received.

Note: It may take up to 60 seconds for the *Messages Available* column to update.

Sent Message Attributes:

Message Identifier: 78fa2441-04c9-4873-9390-
MD5 of Body: 6a1559560f67c5e7a7d5d8
Sequence Number: 188259190627

5. Choose **Close**.

AWS SDK for Java

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

To send a message to a standard queue

1. Copy the example program (p. 67).

   The following section of the code sends the `This is my message text.` message to your queue:

   ```
   // Send a message
   System.out.println("Sending a message to MyQueue.\n");
   sqs.sendMessage(new SendMessageRequest(myQueueUrl, "This is my message text."));
   ```

2. Compile and run the example.

   The message is sent to the queue. The response includes the following items:

   - The message ID (p. 77) Amazon SQS assigns to the message.
   - An MD5 digest of the message body, used to confirm that Amazon SQS received the message correctly (for more information, see RFC1321).
   - The *request ID* that Amazon SQS assigned to your request.

To send a message to a FIFO queue

1. Copy the example program (p. 73).

The following section of the code sends the `This is my message text.` message to your queue:

```
// Send a message
System.out.println("Sending a message to MyFifoQueue.fifo.\n");
final SendMessageRequest sendMessageRequest = new SendMessageRequest(myQueueUrl, "This
 is my message text.");

// When you send messages to a FIFO queue, you must provide a non-empty MessageGroupId.
sendMessageRequest.setMessageGroupId("messageGroup1");

// Uncomment the following to provide the MessageDeduplicationId
//sendMessageRequest.setMessageDeduplicationId("1");
final SendMessageResult sendMessageResult = sqs.sendMessage(sendMessageRequest);
final String sequenceNumber = sendMessageResult.getSequenceNumber();
final String messageId = sendMessageResult.getMessageId();
System.out.println("SendMessage succeed with messageId " + messageId + ", sequence
 number " + sequenceNumber
        + "\n");
```

2.   Compile and run the example.

The message is sent to your queue.

Tutorial: Sending a Message with Attributes to an Amazon SQS Queue

You can include structured metadata (such as timestamps, geospatial data, signatures, and identifiers) with messages using *message attributes*. In this tutorial you'll learn how to send a message with attributes to an existing queue. For more information, see Amazon SQS Message Attributes (p. 77).
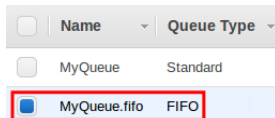
For a more detailed explanation of sending messages to standard and FIFO queues, see Sending a Message to an Amazon SQS Queue (p. 27).

Topics
- AWS Management Console (p. 30)
- AWS SDK for Java (p. 33)

AWS Management Console

1.   Sign in to the Amazon SQS console.
2.   From the queue list, select a queue.

3.   From **Queue Actions**, select **Send a Message**.

The **Send a Message to** *QueueName* dialog box is displayed.

The following example shows the **Message Group ID** and **Message Deduplication ID** parameters specific to FIFO queues (content-based deduplication (p. 72) is disabled).

4. To send a message to a FIFO queue, type the **Message Body**, the **Message Group ID** `MyMessageGroupId1234567890`, and the **Message Deduplication ID** `MyMessageDeduplicationId1234567890`, and then choose **Message Attributes**.

5. Define the message attribute parameters. For more information, see Message Attribute Components (p. 78) and Message Attribute Data Types (p. 78).

   a. For the message attribute **Name** type `MyMessageAttribute`.

   b. For the message attribute data **Type**, select **Number** and type `byte` for the optional custom type.

   c. For the message attribute **Value**, type `24`.

   Choose **Add Attribute**.

   The attribute is added to the message as **Number.byte**.

   You can modify the value before sending the message. To delete the attribute, choose ✖.

6. When you finish adding attributes to the message, choose **Send Message**.

   Your message is sent and the **Send a Message to** *QueueName* dialog box is displayed, showing the attributes of the sent message.

   The following example shows the **MD5 of Message Attributes** specific to your custom message attribute and the **Sequence Number** attribute specific to FIFO queues.

Send a Message to MyQueue.fifo   ✕

Your message has been sent and is ready to be received.

Note: It may take up to 60 seconds for the *Messages Available* column to update.

Sent Message Attributes:

Message Identifier:	dea890ef-d9ba-43e1-9e86-badcb6a5ad92
MD5 of Body:	6a1559560f67c5e7a7d5d838bf0272ee
MD5 of Message Attributes:	c7e128fff5069b1f485828551d66084a
Sequence Number:	18836602484391663616
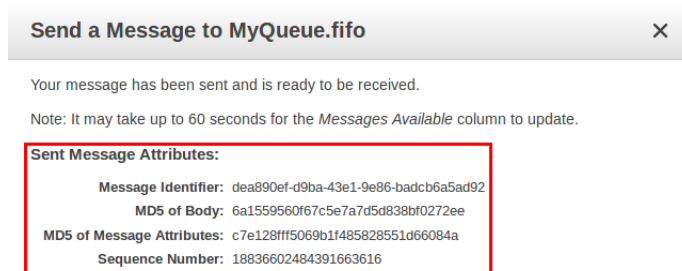
7. Choose **Close**.

AWS SDK for Java

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

To send a message with attributes to a queue

1. Copy the standard queue example program (p. 67) or the FIFO queue example program (p. 73).
2. To define an attribute for a message, add the following code which uses the `MessageAttributeValue` data type. For more information, see Message Attribute Components (p. 78) and Message Attribute Data Types (p. 78).

   > **Note**
   > The AWS SDK for Java automatically calculates the message body and message attribute checksums and compares them with the data which Amazon SQS returns. For more information, see the *AWS SDK for Java Developer Guide* and Calculating the MD5 Message Digest for Message Attributes (p. 79) for other programming languages.

   String

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

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

   Number

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

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

   Binary

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

   ```
   final Map<String, MessageAttributeValue> messageAttributes = new HashMap<>();
   messageAttributes.put("ByteArray", new MessageAttributeValue()
           .withDataType("Binary")
   ```

```
        .withBinaryValue(ByteBuffer.wrap(new byte[10])));
```

String (Custom)

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

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

Number (Custom)

This example defines the custom attribute `Number.AccountId` named `AccountId` with the value `0023456`.

```
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` action 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])));
```

3. Replace the section of the code that sends the message with the following:

```
// 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 FIFO queue, make sure that the `sendMessage` method executes *after* you provide the message group ID.
> If you use the `SendMessageBatch` action instead of `SendMessage`, you must specify message attributes for each individual message in the batch.

4. Compile and run the example.

The message is sent to the queue. The response includes the following items:

- The message ID (p. 77) Amazon SQS assigns to the message.
- An MD5 digest of the message body, used to confirm that Amazon SQS received the message correctly (for more information, see RFC1321).
- An MD5 digest of the message attributes, used to confirm that Amazon SQS received the message attributes correctly.

- The *request ID* that Amazon SQS assigned to your request.

Tutorial: Sending a Message with a Timer to an Amazon SQS Queue

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) invisibility period for a message is 0 seconds. The maximum is 15 minutes. In this tutorial you'll learn how to send a message with a timer to an existing queue. For more information, see Amazon SQS Message Timers (p. 88).

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

For a more detailed explanation of sending messages to standard and FIFO queues, see Sending a Message to an Amazon SQS Queue (p. 27).

Topics

- AWS Management Console (p. 35)
- AWS SDK for Java (p. 36)

AWS Management Console

1. Sign in to the Amazon SQS console.

2. From the queue list, select a queue.
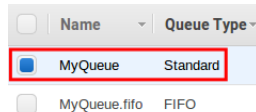
3. From **Queue Actions**, select **Send a Message**.
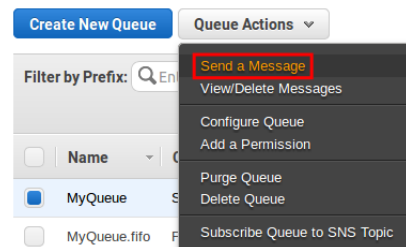
The **Send a Message to *QueueName*** dialog box is displayed.

4. To send a message to a standard queue, type the **Message Body**, choose **Delay delivery of this message by** and type a value, for example 60 seconds.

5. Choose **Send Message**.

   Your message is sent and the **Send a Message to *QueueName*** dialog box is displayed, showing the attributes of the sent message.

6. Choose **Close**.

AWS SDK for Java

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

To send a message with a timer to a queue

1. Copy the standard queue example program (p. 67).

2. Change the `main` method signature to the following:

```
public static void main(String[] args) throws InterruptedException
```

3. Replace the section of the code that sends the message with the following:

```
// Send a message with a 5-second timer.
System.out.println("Sending a message with a 5-second timer to MyQueue.\n");
```

```
SendMessageRequest request = new SendMessageRequest(myQueueUrl, "This is my message
 text.");
request.setDelaySeconds(5);
sqs.sendMessage(request);

// Wait for 10 seconds.
System.out.println("Waiting for 10 seconds.");
Thread.sleep(10000L);
```

4. Compile and run the example.

   The message is sent to the queue. The response includes the following items:

   - The message ID (p. 77) Amazon SQS assigns to the message.
   - An MD5 digest of the message body, used to confirm that Amazon SQS received the message correctly (for more information, see RFC1321).
   - The *request ID* that Amazon SQS assigned to your request.

Tutorial: Receiving and Deleting a Message from an Amazon SQS Queue

After you send a message into a queue, you can consume it from the queue. When you request a message from a queue, you can't specify which message to get. Instead, you specify the maximum number of messages (up to 10) that you want to get.

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, you can rerun the request to get your message. Depending on your application's needs, you might have to use short or long polling (p. 81) to receive messages.

Amazon SQS doesn't automatically delete a message after receiving it for you, in case you don't successfully receive the message (for example, the consumers can fail or lose connectivity). To delete a message, you must send a separate request which acknowledges that you no longer need the message because you've successfully received and processed it.

In this tutorial you'll learn how to receive and delete a message.

Topics
- AWS Management Console (p. 37)
- AWS SDK for Java (p. 39)

AWS Management Console

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.
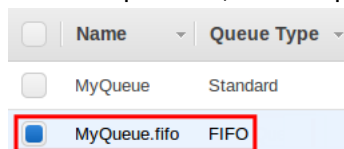
3. From **Queue Actions**, select **View/Delete Messages**.
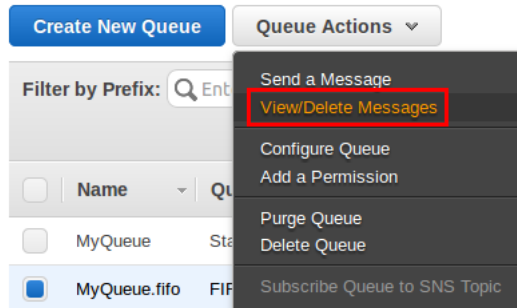
The **View/Delete Messages in _QueueName_** dialog box is displayed.

> **Note**
> The first time you take this action, an information screen is displayed. To hide the screen,
> check the **Don't show this again** checkbox.

4. Choose **Start Polling for messages.**
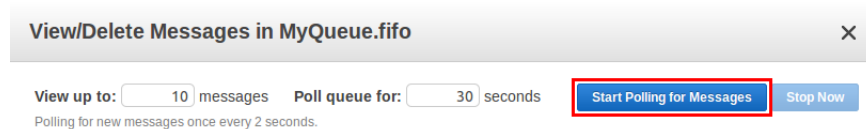
Amazon SQS begins to poll the messages in the queue. The dialog box displays a message from the
queue. A progress bar at the bottom of the dialog box displays the status of the message's visibility
timeout.
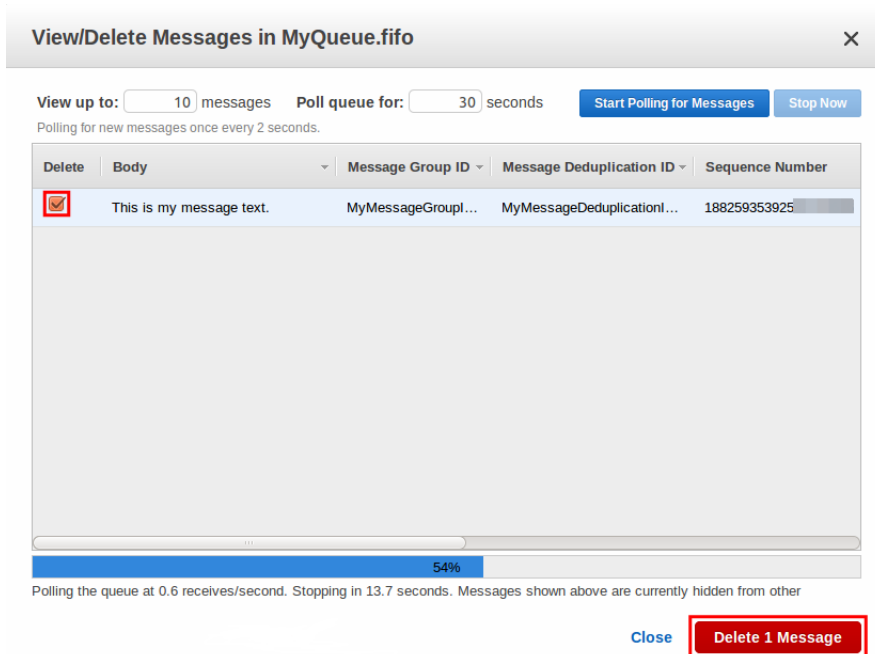
The following example shows the **Message Group ID**, **Message Deduplication ID**, and **Sequence
Number** columns specific to FIFO queues.
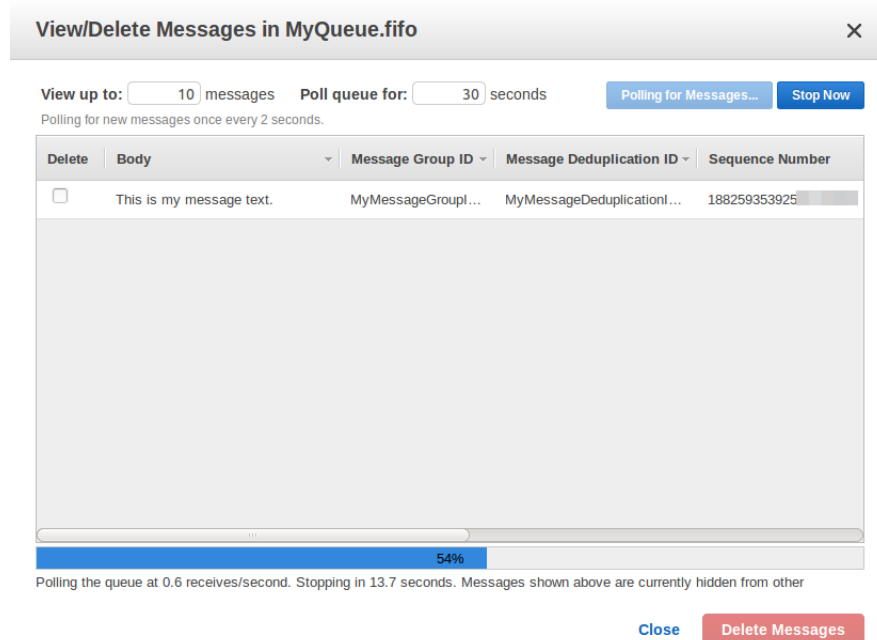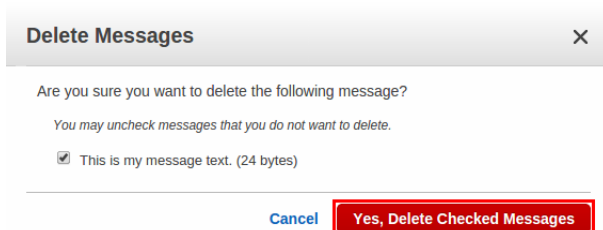
> **Note**
> When the progress bar is filled in, the visibility timeout (p. 85) expires and the message
> becomes visible to consumers.

5. *Before* the visibility timeout expires, select the message that you want to delete and then choose **Delete 1 Message**.

6. In the **Delete Messages** dialog box, confirm that the message you want to delete is checked and choose **Yes, Delete Checked Messages**.

The selected message is deleted.

7. Select **Close**.

AWS SDK for Java

To specify the message to delete, provide the receipt handle (p. 77) that Amazon SQS returned when you received the message. You can delete only one message per action. To delete an entire queue, you must use the `DeleteQueue` action. (You can delete an entire queue even if the queue has messages in it.)

Note
If you don't have the receipt handle for the message, you can call the `ReceiveMessage` action to receive the message again. Each time you receive the message, you get a different receipt handle. Use the latest receipt handle when using the `DeleteMessage` action. Otherwise, your message might not be deleted from the queue.

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

To receive and delete a message from a standard queue

1. Copy the example program (p. 67).

   The following section of the code receives a message from your queue:

```
// Receive messages
System.out.println("Receiving messages from MyQueue.\n");
final ReceiveMessageRequest receiveMessageRequest = new
 ReceiveMessageRequest(myQueueUrl);
final List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
for (final Message message : messages) {
    System.out.println("Message");
    System.out.println("  MessageId:     " + message.getMessageId());
    System.out.println("  ReceiptHandle: " + message.getReceiptHandle());
    System.out.println("  MD5OfBody:     " + message.getMD5OfBody());
    System.out.println("  Body:          " + message.getBody());
    for (final Entry<String, String> entry : message.getAttributes().entrySet()) {
        System.out.println("Attribute");
        System.out.println("  Name:  " + entry.getKey());
        System.out.println("  Value: " + entry.getValue());
    }
}
System.out.println();
```

   The following section of the code deletes the message:

```
// Delete the message
System.out.println("Deleting a message.\n");
final String messageReceiptHandle = messages.get(0).getReceiptHandle();
sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl, messageReceiptHandle));
```

2. Compile and run the example.

   The queue is polled and returns 0 or more messages. The example prints the following items:

   - The message ID (p. 77) that you received when you sent the message to the queue.
   - The receipt handle (p. 77) that you later use to delete the message.
   - An MD5 digest of the message body (for more information, see RFC1321).
   - The message body.
   - The *request ID* that Amazon SQS assigned to your request

   If no messages are received in this particular call, the response includes only the request ID.

   The message is deleted. The response includes the request ID that Amazon SQS assigned to your request.

To receive and delete a message from a FIFO queue

1. Copy the example program (p. 73).

   The following section of the code receives a message from your queue:

```
// Receive messages
System.out.println("Receiving messages from MyFifoQueue.fifo.\n");
final ReceiveMessageRequest receiveMessageRequest = new
 ReceiveMessageRequest(myQueueUrl);
```

```
// Uncomment the following to provide the ReceiveRequestDeduplicationId
//receiveMessageRequest.setReceiveRequestAttemptId("1");
final List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
for (final Message message : messages) {
    System.out.println("Message");
    System.out.println("  MessageId:     " + message.getMessageId());
    System.out.println("  ReceiptHandle: " + message.getReceiptHandle());
    System.out.println("  MD5OfBody:     " + message.getMD5OfBody());
    System.out.println("  Body:          " + message.getBody());
    for (final Entry<String, String> entry : message.getAttributes().entrySet()) {
        System.out.println("Attribute");
        System.out.println("  Name:  " + entry.getKey());
        System.out.println("  Value: " + entry.getValue());
    }
}
System.out.println();
```

The following section of the code deletes the message:

```
// Delete the message
System.out.println("Deleting the message.\n");
final String messageReceiptHandle = messages.get(0).getReceiptHandle();
sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl, messageReceiptHandle));
```

2. Compile and run the example.

    The message is received and deleted.

Tutorial: Subscribing an Amazon SQS Queue to an Amazon SNS Topic

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

> **Important**
> Amazon SNS isn't currently compatible with FIFO queues.
> For information about using Amazon SNS with encrypted Amazon SQS queues, see Enable Compatibility between AWS Services and Queues with SSE (p. 166).
> When you subscribe an Amazon SQS queue to an Amazon SNS topic, Amazon SNS uses HTTPS to forward messages to Amazon SQS.

In this tutorial you'll learn how to subscribe an existing Amazon SQS queue to an existing Amazon SNS topic.

AWS Management Console

1. Sign in to the Amazon SQS console.

2. From the list of queues, choose the queue (or queues) to which you want to subscribe an Amazon SNS topic.
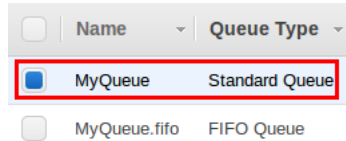
3. From **Queue Actions**, select **Subscribe Queue to SNS Topic** (or **Subscribe Queues to SNS Topic**).
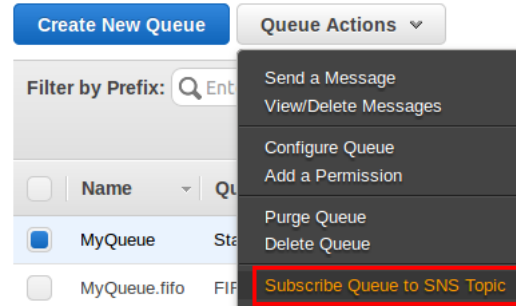
The **Subscribe to a Topic** dialog box is displayed.

4. From the **Choose a Topic** drop-down list, select an Amazon SNS topic to which you want to subscribe your queue (or queues), select the **Topic Region** (optional), and then choose **Subscribe**.
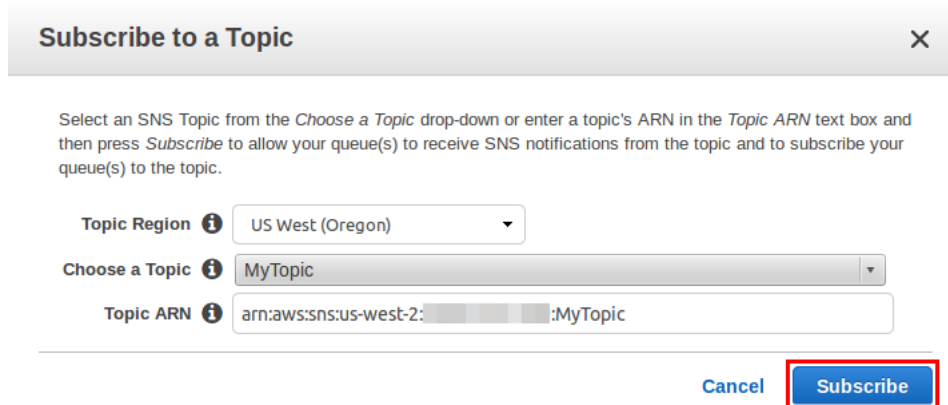
Note
Typing a different **Topic ARN** is useful when you want to subscribe a queue to an Amazon SNS topic from an AWS account other than the one you used to create your Amazon SQS queue.
This is also useful if the Amazon SNS topic isn't listed in the **Choose a Topic** drop-down list.

The **Topic Subscription Result** dialog box is displayed.

5. Review the list of Amazon SQS queues that are subscribed to the Amazon SNS topic and choose **OK**.
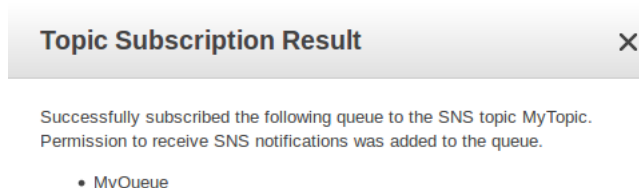
The queue is subscribed to the topic.

> **Note**
> If your Amazon SQS queue and Amazon SNS topic are in different AWS accounts, the owner
> of the topic must first confirm the subscription. For more information, see Confirm the
> Subscription in the *Amazon Simple Notification Service Developer Guide*.
> To list your subscriptions, unsubscribe from topics, and delete topics, use the Amazon SNS
> console. For more information, see Clean Up.

To verify the result of the subscription, you can publish to the topic and then view the message that
the topic sends to the queue. For more information, see Sending Amazon SNS Messages to Amazon
SQS Queues in the *Amazon Simple Notification Service Developer Guide*.

Tutorial: Configuring Messages Arriving in an Amazon SQS Queue to Trigger an AWS Lambda Function

In this tutorial you'll learn how to configure an existing Amazon SQS queue to trigger an AWS Lambda
function when new messages arrive in a queue.

Lambda functions let you run code without provisioning or managing a server. For example, you can
configure a Lambda function to process messages from one queue while another queue acts as a *dead-
letter queue* for messages that your Lambda function can't process successfully. When you resolve the
issue, you can *redrive* the messages from the dead-letter queue through the Lambda function. For more
information see Amazon SQS Dead-Letter Queues (p. 82) and also What is AWS Lambda? and Using
AWS Lambda with Amazon SQS in the *AWS Lambda Developer Guide*.

> **Note**
> Your queue and Lambda function must be in the same AWS Region.
> FIFO queues don't support Lambda function triggers.
> You can associate only one queue with one or more Lambda functions.
> You can't associate an encrypted queue (p. 161) that uses an AWS managed Customer Master
> Key for Amazon SQS with a Lambda function in a different AWS account.

Prerequisites

To configure Lambda function triggers using the console, you must ensure the following:

* Your Amazon SQS role must include the following permissions:
  * `lambda:CreateEventSourceMapping`
  * `lambda:ListEventSourceMappings`
  * `lambda:ListFunction`
* Your Lambda role must include the following permissions:
  * `sqs:ChangeMessageVisibility`
  * `sqs:DeleteMessage`
  * `sqs:GetQueueAttributes`
  * `sqs:ReceiveMessage`

For more information, see Overview of Managing Access Permissions to Your Amazon Simple Queue
Service Resource (p. 137).

AWS Management Console

1. Sign in to the Amazon SQS console.

2. From the list of queues, choose the queue which you want to trigger a Lambda function.
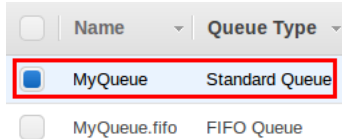
3. From **Queue Actions**, select **Configure Trigger for Lambda Function**.
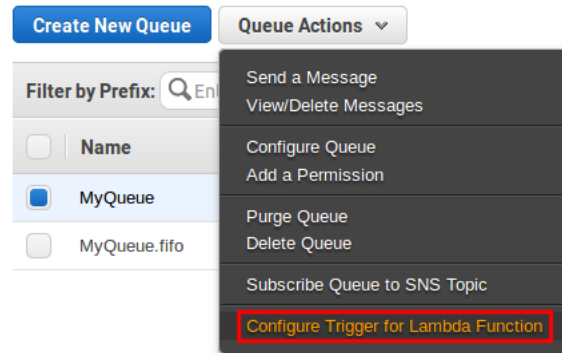
4. In the **Configure Incoming Messages to Trigger a Lambda Function** dialog box, do one of the following:

   - To use an existing Lambda function, **Select a Lambda Function** from the list.
   - To create a new Lambda function in the AWS Lambda console, choose **Create New**. For more information, see Create a Simple Lambda Function in the *AWS Lambda Developer Guide*.

5. Choose **Save**.

6. In the **Lambda Function Trigger Configuration Result** dialog box, review the Lambda function that will be triggered by your Amazon SQS queue and choose **OK**.

Note
It takes approximately 1 minute for the Lambda function to become associated with your queue.

The Lambda function and its status are displayed on the **Lambda Triggers** tab.

Lambda Function ARN	Status	
arn:aws:lambda:us-east-1::function:my-lambda-function	Enabled	✖

- To verify the results of the configuration, you can send a message to your queue (p. 27) and then view the triggered Lambda function in the Lambda console.

- To delete the association between a Lambda function and your queue, choose ✖ next to a Lambda function ARN.

Tutorial: Purging Messages from an Amazon SQS Queue

If you don't want to delete an Amazon SQS queue but need to delete all the messages from it, you can purge the queue. In this tutorial you'll learn how to purge a queue.

Important
When you purge a queue, you can't retrieve any messages deleted from it.
The message deletion process takes up to 60 seconds. We recommend waiting for 60 seconds regardless of your queue's size.

AWS Management Console

1. Sign in to the Amazon SQS console.

2. From the queue list, select a queue.

	Name	▾	Queue Type	▾
☐	MyQueue		Standard	
☑	MyQueue.fifo		FIFO	
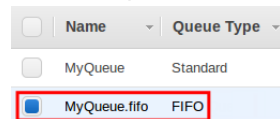
3. From **Queue Actions**, select **Purge Queue**.

Create New Queue **Queue Actions** ⌄

Filter by Prefix: 🔍 Ent

- Send a Message
- View/Delete Messages
- Configure Queue
- Add a Permission
- Purge Queue
- Delete Queue
- Subscribe Queue to SNS Topic

	Name	▾	Qu
☐	MyQueue		Sta
☑	MyQueue.fifo		FIF
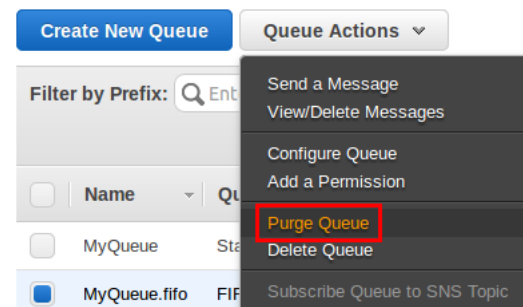
The **Purge Queues** dialog box is displayed.
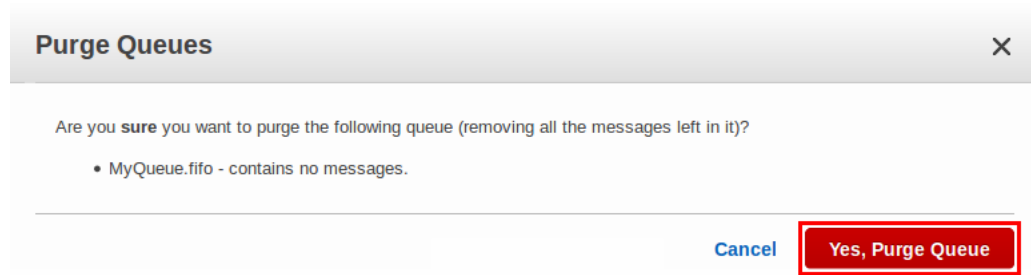
4. Choose **Yes, Purge Queue**.

   All messages are purged from the queue.

   The **Purge Queues** confirmation dialog box is displayed.
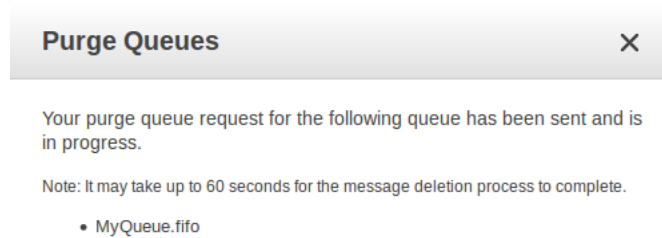
5. Choose **OK**.

Tutorial: Deleting an Amazon SQS Queue

If you don't use an Amazon SQS queue (and don't foresee using it in the near future), it is a best practice to delete it from Amazon SQS. In this tutorial you'll learn how to delete a queue.

> **Note**
> You can delete a queue even when it isn't empty. If you want to delete the messages in a queue but not the queue itself, you can purge the queue (p. 45).
> By default, a queue retains a message for four days after it is sent. You can configure a queue to retain messages for up to 14 days.

Topics

AWS Management Console

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.
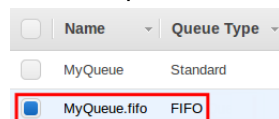

3. From **Queue Actions**, select **Delete Queue**.
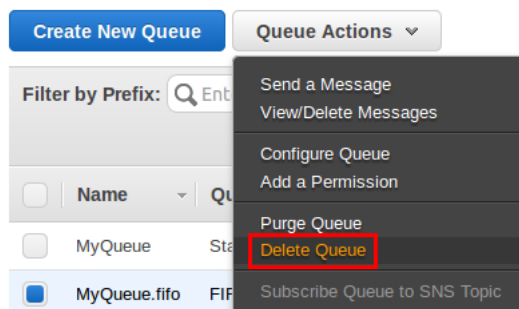
The **Delete Queues** dialog box is displayed.
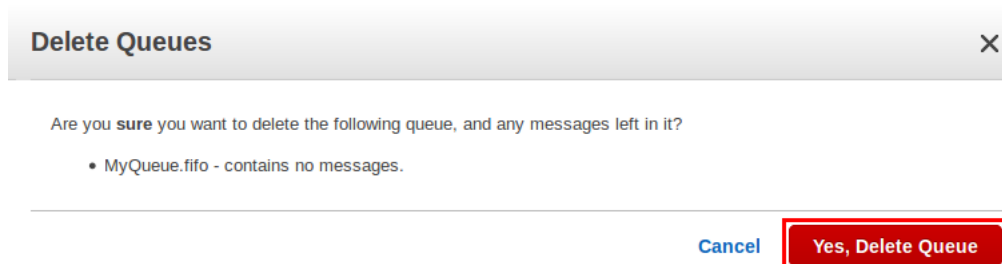
4. Choose **Yes, Delete Queue**.

    The queue is deleted.

AWS SDK for Java

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

Note
This action is identical for standard and FIFO queues.

1. Copy the standard queue example program (p. 67) or the FIFO queue example program (p. 73).

    The following section of the code deletes the queue:

    ```
    // Delete the queue
    System.out.println("Deleting the test queue.\n");
    sqs.deleteQueue(new DeleteQueueRequest(myQueueUrl));
    ```

2. Compile and run the example.

    The queue is deleted.

Tutorials: Configuring Amazon SQS Queues

The following tutorials show how to configure Amazon SQS queues in various ways.

Topics

*   Tutorial: Configuring Server-Side Encryption (SSE) for an Existing Amazon SQS Queue (p. 48)
*   Tutorial: Configuring Long Polling for an Amazon SQS Queue (p. 51)

-
-
-

Tutorial: Configuring Server-Side Encryption (SSE) for an Existing Amazon SQS Queue

Server-side encryption (SSE) for Amazon SQS is available in the AWS GovCloud (US-West) Region, in addition to all commercial regions where Amazon SQS is available, except for the China Regions. You can enable SSE for a queue to protect its data. For more information about using SSE, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161).

Important
All requests to queues with SSE enabled must use HTTPS and Signature Version 4.
When you disable SSE, messages remain encrypted. You must receive and decrypt a message to view its contents.

In this tutorial you'll learn how to enable, disable, and configure SSE for an existing Amazon SQS queue.

Topics
- AWS Management Console (p. 48)
- AWS SDK for Java (p. 49)

AWS Management Console

1. Sign in to the Amazon SQS console.
2. From the queue list, select a queue.
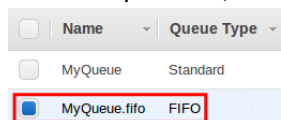
3. From **Queue Actions**, select **Configure Queue**.
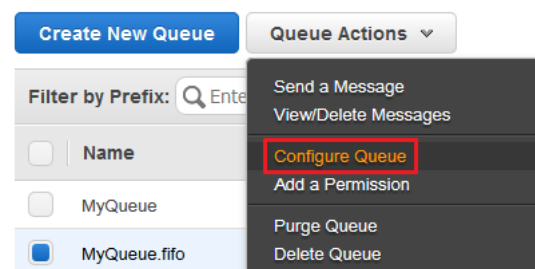
   The **Configure** *QueueName* dialog box is displayed.
4. To enable or disable SSE, use the **Use SSE** check box.
5. Specify the customer master key (CMK) ID. For more information, see Key Terms (p. 163).

   For each CMK type, the **Description**, **Account**, and **Key ARN** of the CMK are displayed.

   Important
   If you aren't the owner of the CMK, or if you log in with an account that doesn't have the `kms:ListAliases` and `kms:DescribeKey` permissions, you won't be able to view information about the CMK on the Amazon SQS console.

Ask the owner of the CMK to grant you these permissions. For more information, see the AWS KMS API Permissions: Actions and Resources Reference in the *AWS Key Management Service Developer Guide*.

- To use the AWS managed CMK for Amazon SQS, select it from the list.

AWS KMS Customer Master Key (CMK) ⓘ	(Default) aws/sqs ▾
Description	Default master key that protects my SQS messages when no other key is defined
Account	
Key ARN	arn:aws:kms:us-east-1:

  Note
  Keep the following in mind:

  - If you don't specify a custom CMK, Amazon SQS uses the AWS managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the *AWS Key Management Service Developer Guide*.

  - The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS managed CMK 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 CMK for Amazon SQS.

- To use a custom CMK from your AWS account, select it from the list.

AWS KMS Customer Master Key (CMK) ⓘ	demo-key ▾
Description	A key for demonstrating the functionality of SSE in Amazon SQS.
Account	
Key ARN	arn:aws:kms:us-east-1:

  Note
  For instructions on creating custom CMKs, see Creating Keys in the *AWS Key Management Service Developer Guide*.

- To use a custom CMK ARN from your AWS account or from another AWS account, select **Enter an existing CMK ARN** from the list and type or copy the CMK.

AWS KMS Customer Master Key (CMK) ⓘ	Enter an existing CMK ARN ▾
Enter a CMK ARN ⓘ	arn:aws:kms:us-east-1:

6. (Optional) For **Data key reuse period**, specify a value between 1 minute and 24 hours. The default is 5 minutes. For more information, see How Does the Data Key Reuse Period Work? (p. 163).

Data Key Reuse Period ⓘ	5	minutes ▾	This value must be between 1 minute and 24 hours.

7. Choose **Save Changes**.

   Your changes are applied to the queue.

AWS SDK for Java

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

Before you can use SSE, you must configure AWS KMS key policies to allow encryption of queues and encryption and decryption of messages. You must also ensure that the key policies of the customer master key (CMK) allow the necessary permissions. For more information, see What AWS KMS Permissions Do I Need to Use SSE for Amazon SQS? (p. 165)

1. Obtain the customer master key (CMK) ID. For more information, see Key Terms (p. 163).

   Note
   Keep the following in mind:

   - If you don't specify a custom CMK, Amazon SQS uses the AWS managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the *AWS Key Management Service Developer Guide*.
   - The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS managed CMK 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 CMK for Amazon SQS.

2. To enable server-side encryption, specify the CMK ID by setting the `KmsMasterKeyId` attribute of the `CreateQueue` or `SetQueueAttributes` action.

   The following code example enables SSE for an existing queue using the AWS managed CMK for Amazon SQS:

   ```
   final SetQueueAttributesRequest setAttributesRequest = new SetQueueAttributesRequest();
   setAttributesRequest.setQueueUrl(queueUrl);

   // Enable server-side encryption by specifying the alias ARN of the
   // AWS managed CMK for Amazon SQS.
   final String kmsMasterKeyAlias = "arn:aws:kms:us-east-2:123456789012:alias/aws/sqs";
   attributes.put("KmsMasterKeyId", kmsMasterKeyAlias);

   final SetQueueAttributesResult setAttributesResult =
    client.setQueueAttributes(setAttributesRequest);
   ```

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

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

3. (Optional) Specify the length of time, in seconds, for which Amazon SQS can reuse a data key (p. 163) to encrypt or decrypt messages before calling AWS KMS. Set the `KmsDataKeyReusePeriodSeconds` attribute of the `CreateQueue` or `SetQueueAttributes` action. Possible values may be between 60 seconds (1 minute) and 86,400 seconds (24 hours). If you don't specify a value, the default value of 300 seconds (5 minutes) is used.

   The following code example sets the data key reuse period to 60 seconds (1 minute):

   ```
   // (Optional) Specify 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.
   attributes.put("KmsDataKeyReusePeriodSeconds", "60");
   ```

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

To retrieve the CMK ID or the data key reuse period for a particular queue, use the `KmsMasterKeyId` and `KmsDataKeyReusePeriodSeconds` attributes of the `GetQueueAttributes` action.

For information about switching a queue to a different CMK with the same alias, see Updating an Alias in the *AWS Key Management Service Developer Guide*.

Tutorial: Configuring Long Polling for an Amazon SQS Queue

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). In this tutorial you'll learn how to configure long polling for an Amazon SQS queue. For more information, see Amazon SQS Long Polling (p. 81).

Topics
- AWS Management Console (p. 51)
- AWS SDK for Java (p. 51)

AWS Management Console

1. Sign in to the Amazon SQS console.
2. Choose **Create New Queue.**
3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

    Note
    The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions.
4. **Standard** is selected by default. Choose **FIFO**.
5. Choose **Configure Queue**.
6. For **Receive Message Wait Time**, type a number between `1` and `20`.

    | Receive Message Wait Time ⓘ | 5 | seconds | Value must be between 0 and 20 seconds. |

    Note
    Setting the value to `0` configures *short polling*. For more information, see Differences Between Long and Short Polling (p. 82).
7. Choose **Create Queue**.

    Your new queue is configured to use long polling, created, and selected in the queue list.

    Note
    When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS.

AWS SDK for Java

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

To configure long polling for a queue

Prerequisites

Add the `aws-java-sdk-sqs.jar` package to your Java class path. The following example shows this dependency in a Maven project `pom.xml` file.

```
<dependencies>
```

```
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-sqs</artifactId>
            <version><replaceable>LATEST</replaceable></version>
        </dependency>
</dependencies>
```

SQSLongPollingExample.java

The following example Java code creates a standard queue and configures long polling for it.

```java
/*
 * Copyright 2010-2018 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.AmazonServiceException;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.QueueAttributeName;

import java.util.Scanner;

public class SQSLongPollingExample {
    public static void main(String[] args) {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the ReceiveMessage wait time (1-20 seconds): ");
        final String receiveMessageWaitTime = input.nextLine();

        /*
         * 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 AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

        try {
            // Create a queue with long polling.
            final CreateQueueRequest createQueueRequest = new CreateQueueRequest()
                    .withQueueName(queueName)
                    .addAttributesEntry(QueueAttributeName.ReceiveMessageWaitTimeSeconds
                            .toString(), receiveMessageWaitTime);
            sqs.createQueue(createQueueRequest);

            System.out.println("Created queue " + queueName + " with " +
                    "ReceiveMessage wait time set to " + receiveMessageWaitTime +
```

```
                            " seconds.");

        } catch (final AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means " +
                    "your request made it to Amazon SQS, but was " +
                    "rejected with an error response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (final AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means " +
                    "the client encountered a serious internal problem while " +
                    "trying to communicate with Amazon SQS, such as not " +
                    "being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

To configure long polling for a queue and send, receive, and delete a message

1. Copy the example program for a or a .

2. Retrieve the queue's URL:

```
final String queueUrl = sqs.getQueueUrl(queueName).getQueueUrl();
```

3. Use the `SetQueueAttributesRequest` action to configure long polling for an existing queue:

```
final SetQueueAttributesRequest setQueueAttributesRequest = new
 SetQueueAttributesRequest()
        .withQueueUrl(queueUrl)
        .addAttributesEntry("ReceiveMessageWaitTimeSeconds", "20");
sqs.setQueueAttributes(setQueueAttributesRequest);
```

4. Use the `ReceiveMessageRequest` action to configure the long polling for a message receipt:

```
final ReceiveMessageRequest receive_request = new ReceiveMessageRequest()
        .withQueueUrl(queueUrl)
        .withWaitTimeSeconds(20);
sqs.receiveMessage(receive_request);
```

5. Compile and run your program.

   The long-polling for your queue is configured and the message is sent, received, and deleted.

Tutorial: Configuring an Amazon SQS Dead-Letter Queue

A dead-letter queue is a queue that *other* (source) queues can target for messages that can't be processed (consumed) successfully. In this tutorial you'll learn how to create an Amazon SQS source queue and to configure a second queue as a dead-letter queue for it. For more information, see .

Important
When you designate a queue to be a source queue, a dead-letter queue is *not* created automatically. You must first create a normal standard or FIFO queue before designating it

a dead-letter queue. This tutorial assumes you already have a normal FIFO queue named
`MyDeadLetterQueue.fifo`.

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.

Topics

AWS Management Console

1. Sign in to the Amazon SQS console.
2. Choose **Create New Queue.**
3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

    > **Note**
    > The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions.

4. **Standard** is selected by default. Choose **FIFO**.
5. Choose **Configure Queue**.
6. In this example, you enable the redrive policy for your new queue, set the `MyDeadLetterQueue.fifo` queue as the dead-letter queue, and set the number of maximum receives to `50`.

❶ To configure the dead-letter queue, choose **Use Redrive Policy**.

❷ Enter the name of the existing **Dead Letter Queue** to which you want sources queues to send messages.

❸ 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.

> **Note**
> The **Maximum Receives** setting applies only to individual messages.

❹ Choose **Create Queue**.

Your new queue is configured to use a dead-letter queue, created, and selected in the queue list.

> **Note**
> When you create a queue, it can take a short time for the queue to propagate throughout
> Amazon SQS.

Your queue's **Maximum Receives** and **Dead Letter Queue** ARN are displayed on the **Redrive Policy**
tab.

Maximum Receives 50

Dead Letter Queue arn:aws:sqs:us-east-2:███████████:MyDeadLetterQueue.fifo

AWS SDK for Java

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

To configure a dead-letter queue

Prerequisites

Add the `aws-java-sdk-sqs.jar` package to your Java class path. The following example shows this
dependency in a Maven project `pom.xml` file.

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version><replaceable>LATEST</replaceable></version>
    </dependency>
</dependencies>
```

SQSDeadLetterQueueExample.java

The following example Java code creates two standard queues and configures one queue to act as a
source queue for the other—a dead-letter queue.

```
/*
 * Copyright 2010-2018 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.AmazonServiceException;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.GetQueueAttributesRequest;
import com.amazonaws.services.sqs.model.GetQueueAttributesResult;
import com.amazonaws.services.sqs.model.QueueAttributeName;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;
```

```
import java.util.Scanner;

public class SQSDeadLetterQueueExample {
    public static void main(String[] args) {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the source queue name: ");
        final String sourceQueueName = input.nextLine();

        System.out.print("Enter the dead-letter queue name: ");
        final String deadLetterQueueName = input.nextLine();

        /*
         * 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 AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

        try {
            // Create a source queue.
            sqs.createQueue(sourceQueueName);

            // Create a dead-letter queue.
            sqs.createQueue(deadLetterQueueName);

            // Get the dead-letter queue ARN.
            final String deadLetterQueueUrl = sqs.getQueueUrl(deadLetterQueueName)
                    .getQueueUrl();
            final GetQueueAttributesResult deadLetterQueueAttributes =
 sqs.getQueueAttributes(
                    new GetQueueAttributesRequest(deadLetterQueueUrl)
                            .withAttributeNames("QueueArn"));
            final String deadLetterQueueArn =
 deadLetterQueueAttributes.getAttributes().get("QueueArn");

            // Set the dead letter queue for the source queue using the redrive policy.
            final String sourceQueueUrl = sqs.getQueueUrl(sourceQueueName)
                    .getQueueUrl();
            final SetQueueAttributesRequest request = new SetQueueAttributesRequest()
                    .withQueueUrl(sourceQueueUrl)
                    .addAttributesEntry(QueueAttributeName.RedrivePolicy.toString(),
                            "{\"maxReceiveCount\":\"5\", \"deadLetterTargetArn\":\""
                                    + deadLetterQueueArn + "\"}");
            sqs.setQueueAttributes(request);

            System.out.println("Set queue " + sourceQueueName + " as source queue " +
                    "for dead-letter queue " + deadLetterQueueName + ".");

        } catch (final AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means " +
                    "your request made it to Amazon SQS, but was " +
                    "rejected with an error response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (final AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means " +
                    "the client encountered a serious internal problem while " +
                    "trying to communicate with Amazon SQS, such as not " +
                    "being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
```

```
            }
        }
}
```

To configure a dead-letter queue and send, receive, and delete a message

1. Copy the example program for a standard queue (p. 67) or a FIFO queue (p. 73).
2. Set a string that contains JSON-formatted parameters and values for the `RedrivePolicy` queue attribute:

```
final String redrivePolicy =
        "{\"maxReceiveCount\":\"5\", \"deadLetterTargetArn\":\"arn:aws:sqs:us-
east-2:123456789012:MyDeadLetterQueue\"}";
```

3. Use the `CreateQueue` or `SetQueueAttributesRequest` action to configure the `RedrivePolicy` queue attribute:

```
final SetQueueAttributesRequest queueAttributes = new SetQueueAttributesRequest();
final Map<String,String> attributes = new HashMap<String,String>();
attributes.put("RedrivePolicy", redrivePolicy);
queueAttributes.setAttributes(attributes);
queueAttributes.setQueueUrl(myQueueUrl);
sqs.setQueueAttributes(queueAttributes);
```

4. Compile and run your program.

   The dead-letter queue is configured and the message is sent, received, and deleted.

Tutorial: Configuring Visibility Timeout for an Amazon SQS Queue

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 other 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. In this tutorial you'll learn how to configure visibility timeout for a queue using the AWS Management Console and for single or multiple messages using the AWS SDK for Java. For more information, see Amazon SQS Visibility Timeout (p. 85).

> **Note**
> You can use the AWS Management Console to configure visibility timeout only for queues, not for single or multiple messages. To do this, you must use one of the AWS SDKs. For more information, see the second Java example code below.

Topics
- AWS Management Console (p. 57)
- AWS SDK for Java (p. 58)

AWS Management Console

1. Sign in to the Amazon SQS console.
2. Choose **Create New Queue.**
3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

> **Note**
> The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the
> US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney),
> and Asia Pacific (Tokyo) regions.

4. **Standard** is selected by default. Choose **FIFO**.

5. Choose **Configure Queue**.

6. In this example, you set the default visibility timeout to 1 minute.

   Default Visibility Timeout ⓘ | 1 | minutes ▾ | Value must be between 0 seconds and 12 hours.

7. Choose **Create Queue**.

   Your new queue is configured to use a 1-minute visibility timeout, created, and selected in the queue
   list.

   > **Note**
   > When you create a queue, it can take a short time for the queue to propagate throughout
   > Amazon SQS.

   Your queue's **Default Visibility Timeout** is displayed on the **Details** tab.

   Default Visibility Timeout: 1 minutes

AWS SDK for Java

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

To configure visibility timeout for a queue

Prerequisites

Add the `aws-java-sdk-sqs.jar` package to your Java class path. The following example shows this
dependency in a Maven project `pom.xml` file.

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version><replaceable>LATEST</replaceable></version>
    </dependency>
</dependencies>
```

SQSVisibilityTimeoutExample.java

The following example Java code creates a standard queue and sets the visibility timeout for it to 1
minute.

```
/*
 * Copyright 2010-2018 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.AmazonServiceException;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.QueueAttributeName;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;

import java.util.Scanner;

public class SQSVisibilityTimeoutExample {
    public static void main(String[] args) {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the visibility timeout in seconds " +
                "(0 seconds to 12 hours): ");
        final String visibilityTimeout = input.nextLine();

        /*
         * 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 AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

        try {
            // Create a queue.
            final CreateQueueRequest createQueueRequest = new CreateQueueRequest()
                    .withQueueName(queueName);
            sqs.createQueue(createQueueRequest);

            // Set the visibility timeout for the queue.
            final String queueUrl = sqs.getQueueUrl(queueName)
                    .getQueueUrl();
            final SetQueueAttributesRequest request = new SetQueueAttributesRequest()
                    .withQueueUrl(queueUrl)
                    .addAttributesEntry(QueueAttributeName.VisibilityTimeout
                            .toString(), visibilityTimeout);
            sqs.setQueueAttributes(request);

            System.out.println("Created queue " + queueName + " with " +
                    "visibility timeout set to " + visibilityTimeout +
                    " seconds.");

        } catch (final AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means " +
                    "your request made it to Amazon SQS, but was " +
                    "rejected with an error response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (final AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means " +
                    "the client encountered a serious internal problem while " +
```

```
                    "trying to communicate with Amazon SQS, such as not " +
                    "being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

To configure visibility timeout for a single message or multiple messages and send, receive, and delete messages

1. Copy the example program for a standard queue (p. 67) or a FIFO queue (p. 73).

2. To configure visibility timeout for a single message, pass the queue URL, the message receipt handle, and the visibility timeout value in seconds.

    ```
    // Get the message receipt handle.
     String receiptHandle = sqs.receiveMessage(myQueueUrl)
            .getMessages()
            .get(0)
            .getReceiptHandle();

     // Pass the queue URL, the message receipt handle, and the visibility timeout value.
     sqs.changeMessageVisibility(myQueueUrl, receiptHandle, timeoutValue);
    ```

3. To configure visibility timeout for multiple messages (for example, if you want to set different timeout values for different messages), create an `ArrayList`, add messages to it with the visibility timeout value in seconds, and then pass the queue URL and the `ArrayList` of messages.

    ```
    // Create an ArrayList for batched messages.
     List<ChangeMessageVisibilityBatchRequestEntry> entries =
            new ArrayList<ChangeMessageVisibilityBatchRequestEntry>();

     // Add the first message to the ArrayList with a visibility timeout value.
     entries.add(new ChangeMessageVisibilityBatchRequestEntry(
            "uniqueMessageId123", sqs.receiveMessage(myQueueUrl)
            .getMessages()
            .get(0)
            .getReceiptHandle())
            .withVisibilityTimeout(timeoutValue));

     // Add the second message to the ArrayList with a different timeout value.
     entries.add(new ChangeMessageVisibilityBatchRequestEntry(
            "uniqueMessageId456", sqs.receiveMessage(myQueueUrl)
            .getMessages()
            .get(0)
            .getReceiptHandle())
            .withVisibilityTimeout(timeoutValue + 60));

     sqs.changeMessageVisibilityBatch(myQueueUrl, entries);
    ```

4. Compile and run your program.

    The visibility timeout for a single message or multiple messages is configured and the message is sent, received, and deleted.

Tutorial: Configuring an Amazon SQS Delay Queue

Delay queues let you postpone the delivery of new messages to a queue for a number of seconds. 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. In this tutorial you'll learn how to configure a delay queue using the AWS Management Console or using the AWS SDK for Java. For more information, see Amazon SQS Delay Queues (p. 87).

Topics
- AWS Management Console (p. 61)
- AWS SDK for Java (p. 61)
- To configure a delay queue and send, receive, and delete messages (p. 63)

AWS Management Console

1. Sign in to the Amazon SQS console.
2. Choose **Create New Queue.**
3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

   > **Note**
   > The name of a FIFO queue must end with the `.fifo` suffix. FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions.

4. **Standard** is selected by default. Choose **FIFO**.
5. Choose **Configure Queue**.
6. In this example, you set the delivery delay to 1 minute.

   Delivery Delay ⓘ 60 seconds ▾ Value must be between 0 seconds and 15 minutes.

7. Choose **Create Queue**.

   Your new queue is configured to use a 1-minute delay, created, and selected in the queue list.

   > **Note**
   > When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS.

   Your queue's **Delivery Delay** is displayed on the **Details** tab.

   Delivery Delay: 1 minutes

AWS SDK for Java

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

To configure a delay queue

Prerequisites

Add the `aws-java-sdk-sqs.jar` package to your Java class path. The following example shows this dependency in a Maven project `pom.xml` file.

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version><replaceable>LATEST</replaceable></version>
    </dependency>
</dependencies>
```

SQSDelayQueueExample.java

The following example Java code creates a standard queue and sets the delay for it to 1 minute.

```
/*
 * Copyright 2010-2018 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.AmazonServiceException;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.CreateQueueRequest;
import com.amazonaws.services.sqs.model.QueueAttributeName;
import com.amazonaws.services.sqs.model.SetQueueAttributesRequest;

import java.util.Scanner;

public class SQSDelayQueueExample {
    public static void main(String[] args) {

        final Scanner input = new Scanner(System.in);

        System.out.print("Enter the queue name: ");
        final String queueName = input.nextLine();

        System.out.print("Enter the delay in seconds (0 seconds to 15 minutes): ");
        final String queueDelay = input.nextLine();

        /*
         * 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 AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

        try {
            // Create a queue.
            final CreateQueueRequest createQueueRequest = new CreateQueueRequest()
                    .withQueueName(queueName);
            sqs.createQueue(createQueueRequest);

            // Set the delay for the queue.
            final String queueUrl = sqs.getQueueUrl(queueName)
                    .getQueueUrl();
            final SetQueueAttributesRequest request = new SetQueueAttributesRequest()
                    .withQueueUrl(queueUrl)
                    .addAttributesEntry(QueueAttributeName.DelaySeconds
                            .toString(), queueDelay);
            sqs.setQueueAttributes(request);

            System.out.println("Created queue " + queueName + " with " +
                    "delay set to " + queueDelay + " seconds.");
```

```
        } catch (final AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means " +
                    "your request made it to Amazon SQS, but was " +
                    "rejected with an error response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (final AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means " +
                    "the client encountered a serious internal problem while " +
                    "trying to communicate with Amazon SQS, such as not " +
                    "being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

To configure a delay queue and send, receive, and delete messages

1. Copy the example program for a standard queue (p. 67) or a FIFO queue (p. 73).
2. To configure a delay queue, pass the delay value in seconds.

```
// Set the delay for the queue.
 final String queueUrl = sqs.getQueueUrl(queueName).getQueueUrl();
 final SetQueueAttributesRequest request = new SetQueueAttributesRequest()
        .withQueueUrl(queueUrl)
        .addAttributesEntry(QueueAttributeName.DelaySeconds.toString(), queueDelay);
 sqs.setQueueAttributes(request);
```

3. Compile and run your program.

   The delay queue is configured and the message is sent, received, and deleted.

How Amazon SQS Works

This section describes the types of Amazon SQS queues and their basic properties. It also describes the identifiers of queues and messages, and various queue and message management workflows.

Topics

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 components that send messages to the queue and receive messages from the queue. The queue (which holds messages A through E) redundantly stores the messages across multiple Amazon SQS servers.
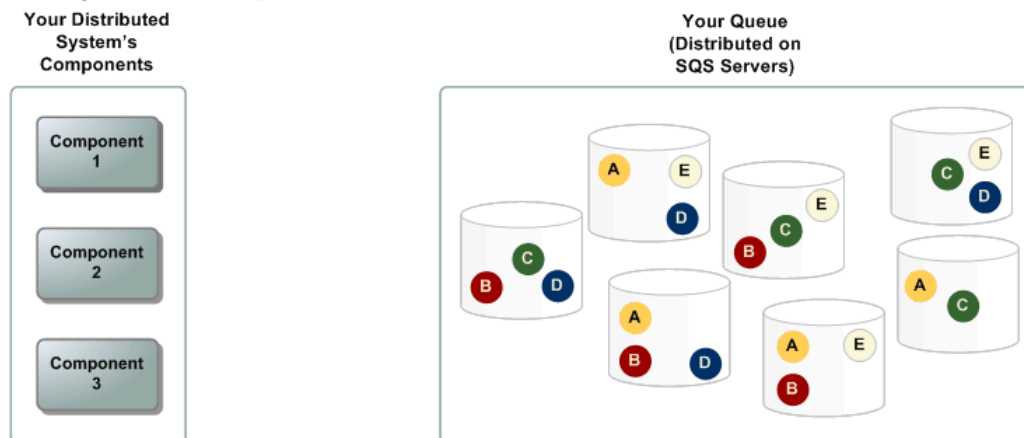
Message Lifecycle

The following scenario describes the lifecycle of an Amazon SQS message in a queue, from creation to deletion.
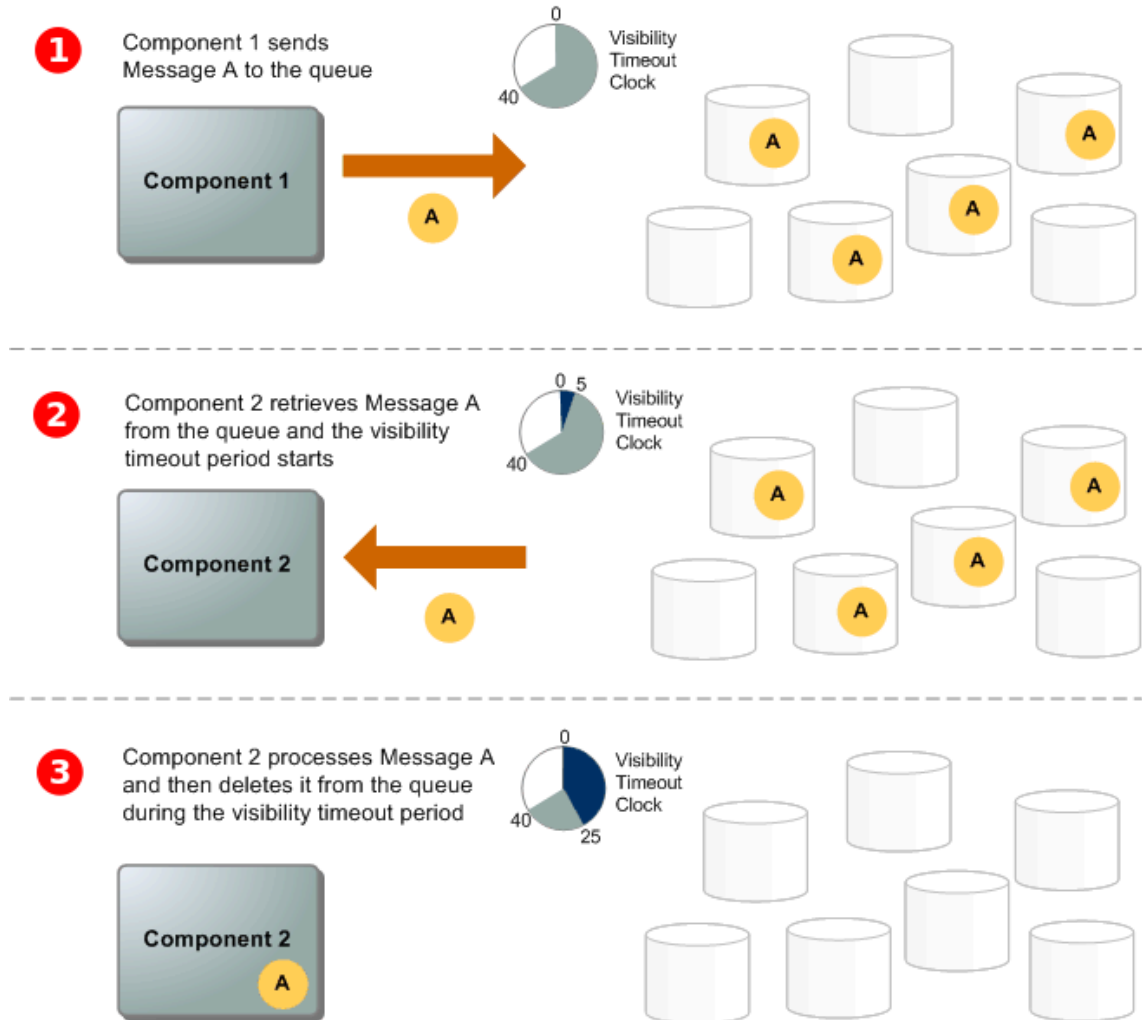
❶ A producer (component 1) sends message A to a queue, and the message is distributed across the Amazon SQS servers redundantly.

❷ 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 (p. 85).

❸ 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 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.

Amazon SQS Standard Queues

Amazon SQS offers *standard* as the default queue type. Standard queues support a nearly unlimited number of transactions per second (TPS) per action. 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.

For information about creating standard queues with or without server-side encryption using the AWS Management Console, the AWS SDK for Java (and the `CreateQueue` action), or AWS CloudFormation, see Creating an Amazon SQS Queue (p. 15) and Creating an Amazon SQS Queue with SSE (p. 19).

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 best practices of working with standard queues, see Recommendations for Amazon SQS Standard and FIFO (First-In-First-Out) Queues (p. 115).

Topics
- Message Ordering (p. 66)
- At-Least-Once Delivery (p. 66)
- Consuming Messages Using Short Polling (p. 66)
- Working Java Example for Standard Queues (p. 67)

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* (p. 69) 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).

Consuming Messages Using Short Polling

The process of consuming messages from a queue depends on whether you use short polling (the default behavior) or long polling. For more information about long polling, see Amazon SQS Long Polling (p. 81).

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 receive 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.
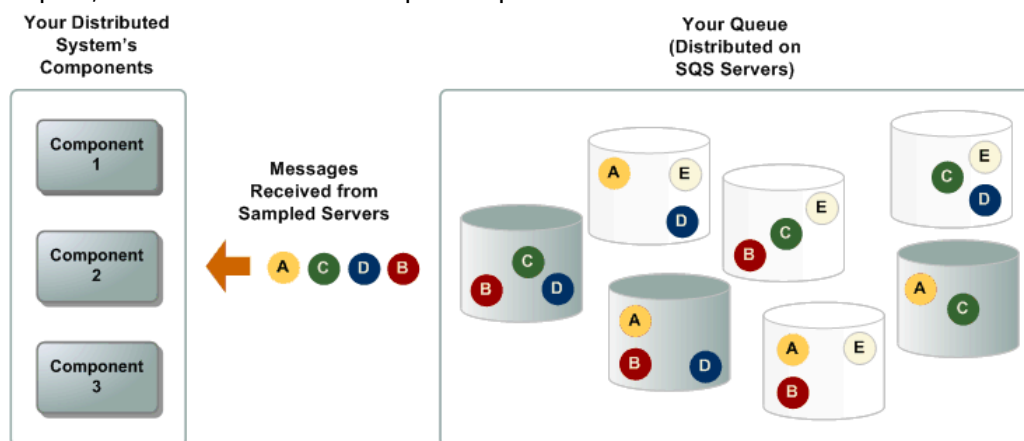
Working Java Example for Standard Queues

Learn about Amazon SQS standard queue functionality using the provided Maven prerequisites and example Java code.

Prerequisites

Add the `aws-java-sdk-sqs.jar` package to your Java class path. The following example shows this dependency in a Maven project `pom.xml` file.

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version><replaceable>LATEST</replaceable></version>
    </dependency>
</dependencies>
```

SQSSimpleJavaClientExample.java

The following example Java code creates a queue and sends, receives, and deletes a message.

```
/*
 * Copyright 2010-2018 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.AmazonServiceException;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;

import java.util.List;
import java.util.Map.Entry;

/**
 * This sample demonstrates how to make basic requests to Amazon SQS using the
 * AWS SDK for Java.
 * <p>
 * Prerequisites: You must have a valid Amazon Web Services developer account,
 * and be signed up to use Amazon SQS. For more information about Amazon SQS,
 * see https://aws.amazon.com/sqs
 * <p>
 * Make sure that your credentials are located in ~/.aws/credentials
 */
public class SQSSimpleJavaClientExample {
    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 AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

        System.out.println("===============================================");
        System.out.println("Getting Started with Amazon SQS Standard Queues");
        System.out.println("===============================================\n");

        try {
            // Create a queue.
            System.out.println("Creating a new SQS queue called MyQueue.\n");
            final CreateQueueRequest createQueueRequest =
                    new CreateQueueRequest("MyQueue");
            final String myQueueUrl = sqs.createQueue(createQueueRequest)
                    .getQueueUrl();

            // List all queues.
            System.out.println("Listing all queues in your account.\n");
            for (final String queueUrl : sqs.listQueues().getQueueUrls()) {
                System.out.println("  QueueUrl: " + queueUrl);
            }
            System.out.println();

            // Send a message.
            System.out.println("Sending a message to MyQueue.\n");
            sqs.sendMessage(new SendMessageRequest(myQueueUrl,
                    "This is my message text."));

            // Receive messages.
            System.out.println("Receiving messages from MyQueue.\n");
            final ReceiveMessageRequest receiveMessageRequest =
                    new ReceiveMessageRequest(myQueueUrl);
            final List<Message> messages = sqs.receiveMessage(receiveMessageRequest)
                    .getMessages();
```

```
            for (final Message message : messages) {
                System.out.println("Message");
                System.out.println("  MessageId:     "
                        + message.getMessageId());
                System.out.println("  ReceiptHandle: "
                        + message.getReceiptHandle());
                System.out.println("  MD5OfBody:     "
                        + message.getMD5OfBody());
                System.out.println("  Body:          "
                        + message.getBody());
                for (final Entry<String, String> entry : message.getAttributes()
                        .entrySet()) {
                    System.out.println("Attribute");
                    System.out.println("  Name:  " + entry
                            .getKey());
                    System.out.println("  Value: " + entry
                            .getValue());
                }
            }
            System.out.println();

            // Delete the message.
            System.out.println("Deleting a message.\n");
            final String messageReceiptHandle = messages.get(0).getReceiptHandle();
            sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl,
                    messageReceiptHandle));

            // Delete the queue.
            System.out.println("Deleting the test queue.\n");
            sqs.deleteQueue(new DeleteQueueRequest(myQueueUrl));
        } catch (final AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means " +
                    "your request made it to Amazon SQS, but was " +
                    "rejected with an error response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (final AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means " +
                    "the client encountered a serious internal problem while " +
                    "trying to communicate with Amazon SQS, such as not " +
                    "being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

Amazon SQS FIFO (First-In-First-Out) Queues

FIFO queues are available in the US East (N. Virginia), US East (Ohio), US West (Oregon), EU (Ireland), Asia Pacific (Sydney), and Asia Pacific (Tokyo) regions. FIFO queues have all the capabilities of the standard queue (p. 66).

For information about creating FIFO queues with or without server-side encryption using the AWS Management Console, the AWS SDK for Java (and the `CreateQueue` action), or AWS CloudFormation, see Creating an Amazon SQS Queue (p. 15) and Creating an Amazon SQS Queue with SSE (p. 19).

FIFO (First-In-First-Out) queues are designed to enhance messaging between applications when the order of operations and events is critical, or where duplicates can't be tolerated, for example:

- Ensure that user-entered commands are executed 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.

FIFO queues also provide exactly-once processing but have a limited number of transactions per second (TPS):

- By default, FIFO queues support up to 3,000 messages per second with batching (p. 174). To request a limit increase, file a support request.

- FIFO queues support up to 300 messages per second (300 send, receive, or delete operations per second) without batching.

> **Note**
> The name of a FIFO queue must end with the `.fifo` suffix. The suffix counts towards the 80-character queue name limit. To determine whether a queue is FIFO (p. 69), you can check whether the queue name ends with the suffix.

For best practices of working with FIFO queues, see Additional Recommendations for Amazon SQS FIFO Queues (p. 118) and Recommendations for Amazon SQS Standard and FIFO (First-In-First-Out) Queues (p. 115).

For information about compatibility of clients and services with FIFO queues, see Compatibility (p. 73).

Topics

Message Ordering

The FIFO queue improves upon and complements the standard queue (p. 66). The most important features of this queue type are *FIFO (First-In-First-Out) delivery* (p. 71) and *exactly-once processing (p. 72)*:

- The order in which messages are sent and received is strictly preserved and a message is delivered once and remains available 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.

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**
> Message deduplication applies to an entire queue, not to individual message groups. 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.

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 ensure that Amazon SQS preserves the order in which messages are sent and received, ensure that each producer uses 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.

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

available with the same message group ID, you might receive messages from another message group ID, in the same batch of 10 messages, but still in FIFO order.

Retrying Multiple Times

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

- If the producer detects a failed `SendMessage` action, it can retry sending as many times as necessary, using the same message deduplication ID. Assuming that the producer receives at least one acknowledgement before the deduplication interval expires, multiple retries neither affect the ordering of messages nor introduce duplicates.
- If the consumer detects a failed `ReceiveMessage` action, it can retry as many times as necessary, using the same receive request attempt ID. Assuming that the consumer receives at least one acknowledgement before the visibility timeout expires, multiple retries don't affect the ordering of messages.
- When you receive a message with a message group ID, no more messages for the same message group ID are returned unless you delete the message or it becomes visible.

Exactly-Once Processing

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.

Use the following checklist to ensure that your application works correctly with a FIFO queue.

- By default, FIFO queues support up to 3,000 messages per second with batching (p. 174). To request a limit increase, file a support request. FIFO queues support up to 300 messages per second (300 send, receive, or delete operations per second) without batching.
- 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.
- Every message sent to a FIFO queue requires a message group ID. If you don't need multiple ordered message groups, specify the same message group ID for all your messages.
- 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*.

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:

- 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.

Working Java Example for FIFO Queues

Learn about Amazon SQS FIFO queue functionality using the provided Maven prerequisites and example Java code.

Prerequisites

Add the `aws-java-sdk-sqs.jar` package to your Java class path. The following example shows this dependency in a Maven project `pom.xml` file.

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version><replaceable>LATEST</replaceable></version>
    </dependency>
</dependencies>
```

SQSFIFOJavaClientExample.java

The following example Java code creates a queue and sends, receives, and deletes a message.

```
/*
```

```java
 * Copyright 2010-2018 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.AmazonServiceException;
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

public class SQSFIFOJavaClientExample {
    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 AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();

        System.out.println("===========================================");
        System.out.println("Getting Started with Amazon SQS FIFO Queues");
        System.out.println("===========================================\n");

        try {

            // Create a FIFO queue.
            System.out.println("Creating a new Amazon SQS FIFO queue called " +
                    "MyFifoQueue.fifo.\n");
            final Map<String, String> attributes = new HashMap<>();

            // A FIFO queue must have the FifoQueue attribute set to true.
            attributes.put("FifoQueue", "true");

            /*
             * If the user doesn't provide a MessageDeduplicationId, generate a
             * MessageDeduplicationId based on the content.
             */
            attributes.put("ContentBasedDeduplication", "true");

            // The FIFO queue name must end with the .fifo suffix.
            final CreateQueueRequest createQueueRequest =
                    new CreateQueueRequest("MyFifoQueue.fifo")
                            .withAttributes(attributes);
            final String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();

            // List all queues.
            System.out.println("Listing all queues in your account.\n");
            for (final String queueUrl : sqs.listQueues().getQueueUrls()) {
                System.out.println("  QueueUrl: " + queueUrl);
            }
```

```
            System.out.println();

            // Send a message.
            System.out.println("Sending a message to MyFifoQueue.fifo.\n");
            final SendMessageRequest sendMessageRequest =
                    new SendMessageRequest(myQueueUrl,
                            "This is my message text.");

            /*
             * When you send messages to a FIFO queue, you must provide a
             * non-empty MessageGroupId.
             */
            sendMessageRequest.setMessageGroupId("messageGroup1");

            // Uncomment the following to provide the MessageDeduplicationId
            //sendMessageRequest.setMessageDeduplicationId("1");
            final SendMessageResult sendMessageResult = sqs
                    .sendMessage(sendMessageRequest);
            final String sequenceNumber = sendMessageResult.getSequenceNumber();
            final String messageId = sendMessageResult.getMessageId();
            System.out.println("SendMessage succeed with messageId "
                    + messageId + ", sequence number " + sequenceNumber + "\n");

            // Receive messages.
            System.out.println("Receiving messages from MyFifoQueue.fifo.\n");
            final ReceiveMessageRequest receiveMessageRequest =
                    new ReceiveMessageRequest(myQueueUrl);

            // Uncomment the following to provide the ReceiveRequestDeduplicationId
            //receiveMessageRequest.setReceiveRequestAttemptId("1");
            final List<Message> messages = sqs.receiveMessage(receiveMessageRequest)
                    .getMessages();
            for (final Message message : messages) {
                System.out.println("Message");
                System.out.println("  MessageId:     "
                        + message.getMessageId());
                System.out.println("  ReceiptHandle: "
                        + message.getReceiptHandle());
                System.out.println("  MD5OfBody:     "
                        + message.getMD5OfBody());
                System.out.println("  Body:          "
                        + message.getBody());
                for (final Entry<String, String> entry : message.getAttributes()
                        .entrySet()) {
                    System.out.println("Attribute");
                    System.out.println("  Name:  " + entry.getKey());
                    System.out.println("  Value: " + entry.getValue());
                }
            }
            System.out.println();

            // Delete the message.
            System.out.println("Deleting the message.\n");
            final String messageReceiptHandle = messages.get(0).getReceiptHandle();
            sqs.deleteMessage(new DeleteMessageRequest(myQueueUrl,
                    messageReceiptHandle));

            // Delete the queue.
            System.out.println("Deleting the queue.\n");
            sqs.deleteQueue(new DeleteQueueRequest(myQueueUrl));
        } catch (final AmazonServiceException ase) {
            System.out.println("Caught an AmazonServiceException, which means " +
                    "your request made it to Amazon SQS, but was " +
                    "rejected with an error response for some reason.");
            System.out.println("Error Message:    " + ase.getMessage());
            System.out.println("HTTP Status Code: " + ase.getStatusCode());
```

```
            System.out.println("AWS Error Code:   " + ase.getErrorCode());
            System.out.println("Error Type:       " + ase.getErrorType());
            System.out.println("Request ID:       " + ase.getRequestId());
        } catch (final AmazonClientException ace) {
            System.out.println("Caught an AmazonClientException, which means " +
                    "the client encountered a serious internal problem while " +
                    "trying to communicate with Amazon SQS, such as not " +
                    "being able to access the network.");
            System.out.println("Error Message: " + ace.getMessage());
        }
    }
}
```

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 and 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 limit. To determine whether a queue is FIFO (p. 69), you can check whether the queue name ends with the suffix.

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
```

> **Important**
> In your system, always store the entire queue URL exactly as Amazon SQS
> returns it to you when you create the queue (for example, `https://sqs.us-east-2.amazonaws.com/123456789012/MyQueue`). Don't build the queue URL from its
> separate components each time you need to specify the queue URL in a request because
> Amazon SQS can change the components that make up the queue URL.

You can also get the queue URL for a queue by listing your queues. 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. (However, to delete a message you need the message's *receipt handle*.) 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 (p. 72) and the *Amazon Simple Queue Service API Reference*.

Message Deduplication ID

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

Message Group ID

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

Sequence Number

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

Amazon SQS 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 AWS Management Console or the AWS SDK for Java, see Sending a Message with Attributes to an Amazon SQS Queue (p. 30).

Topics

- Message Attribute Components (p. 78)
- Message Attribute Data Types (p. 78)
- Calculating the MD5 Message Digest for Message Attributes (p. 79)

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 with UTF-8 binary encoding. For more information, see ASCII Printable 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:
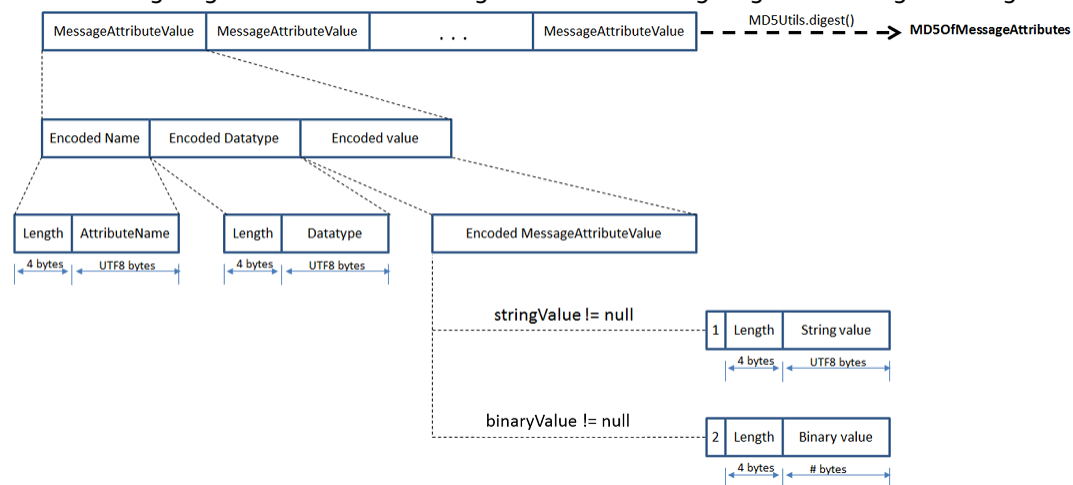
To encode a single Amazon SQS message attribute

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..

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 (p. 85).

> **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`
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`

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. For information about managing Amazon SQS queue tags using the AWS Management Console or the AWS SDK for Java (and the `TagQueue`, `UntagQueue`, and `ListQueueTags` actions), see the Adding, Updating, and Removing Tags from an Amazon SQS Queue (p. 26) tutorial.

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 and Cost Management 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.
- 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 5 TPS per AWS account. If your application requires a higher throughput, file a support request.

For a full list of tag restrictions, see Limits Related to Queues (p. 121).

You can't add tags to a new queue when you create it using the AWS Management Console (you *can* add tags after the queue is created). However, you can add, update, or remove queue tags at any time using the Amazon SQS actions.

Amazon SQS Long Polling

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 AWS Management Console or the AWS SDK for Java (and the `CreateQueue`, `SetQueueAttributes`, and `ReceiveMessage` actions), see the Configuring Long Polling for an Amazon SQS queue (p. 51) tutorial. For best practices, see Setting Up Long Polling (p. 116).

Long polling offers the following benefits:

- Eliminate 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.
- Eliminate false empty responses by querying all—rather than a subset of—Amazon SQS servers.

  Note
  You can confirm that a queue is empty when you perform a long poll and the `ApproximateNumberOfMessagesDelayed`, `ApproximateNumberOfMessagesNotVisible`, and `ApproximateNumberOfMessagesVisible` metrics are equal to 0 at least 1 minute after the producers stop sending messages (when the queue metadata reaches eventual consistency). For more information, see Available CloudWatch Metrics for Amazon SQS (p. 127).

- Return messages as soon as they become available.

Differences Between Long and Short Polling

By default, Amazon SQS uses *short polling*, querying only a subset of its servers (based on a weighted random distribution), to determine whether any messages are available for a response.

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`.

> **Note**
> For the `WaitTimeSeconds` parameter of the `ReceiveMessage` action, a value set between `1` and `20` has priority over any value set for the queue attribute `ReceiveMessageWaitTimeSeconds`.

Amazon SQS Dead-Letter Queues

Amazon SQS supports *dead-letter queues*, 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 problematic messages to determine why their processing doesn't succeed. For information about creating a queue and configuring a dead-letter queue for it using the AWS Management Console or the AWS SDK for Java (and the `CreateQueue`, `SetQueueAttributes`, and `GetQueueAttributes` actions), see Configuring an Amazon SQS Dead-Letter Queue (p. 53).

> **Important**
> When you designate a queue to be a source queue, a dead-letter queue is *not* created automatically. You must first create a normal standard or FIFO queue before designating it a dead-letter queue.

Topics

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. When the `ReceiveCount` for a message exceeds the `maxReceiveCount` for a queue, Amazon SQS moves the message to a dead-letter queue (with its original message ID). For example, if the source queue has a redrive policy with `maxReceiveCount` set to 5, and the consumer of the source queue receives a message 5 times without ever deleting it, Amazon SQS moves the message to the dead-letter queue.

To specify a dead-letter queue, you can use the AWS Management Console or the AWS SDK for Java (p. 53). You must do this for each queue that sends messages to a dead-letter queue. Multiple queues can target a single dead-letter queue. For more information, see Configuring an Amazon SQS Dead-Letter Queue (p. 53) and the `RedrivePolicy` attribute 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.
> The expiration of a message is always based on its original enqueue timestamp. When a message is moved to a dead-letter queue (p. 82), the enqueue timestamp remains unchanged. For example, if a message spends 1 day in the original queue before being moved to a dead-letter queue, and the retention period of the dead-letter queue is set to 4 days, the message is deleted from the dead-letter queue after 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.

What are the Benefits of Dead-Letter Queues?

The main task of a dead-letter queue is handling message failure. 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 delivered to a dead-letter queue.
- Examine logs for exceptions that might have caused messages to be delivered to a dead-letter queue.
- Analyze the contents of messages delivered 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 (p. 66) keep processing messages until the expiration of the retention period. This ensures continuous processing of messages, which minimizes the chances of your queue being blocked by messages that can't be processed. It also ensures fast 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 inflight 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, you must ensure that your application handles message processing correctly.

FIFO Queues

FIFO queues (p. 69) ensure 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.

Note
FIFO queues allow a lower number of inflight messages. Thus, to ensure that your FIFO queue doesn't get blocked by a message, you must ensure that your application handles message processing correctly.

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.

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 AWS Management Console Might Cause Messages to be Moved to a Dead-Letter Queue

Amazon SQS counts viewing a message in the AWS Management Console against the corresponding queue's redrive policy. Thus, if you view a message in the AWS Management 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 AWS Management 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, 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.

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.
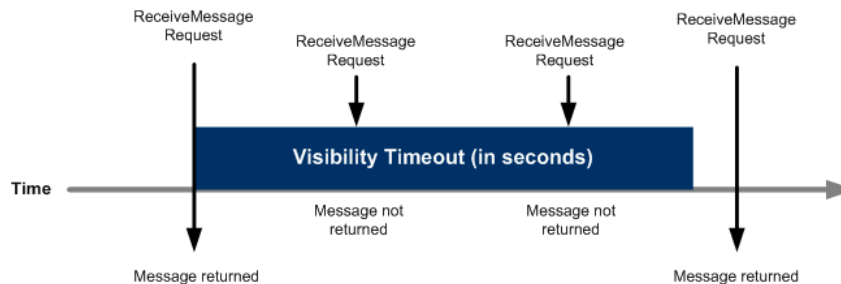
Immediately after a message is received, it remains in the queue. To prevent other consumers from processing the message again, Amazon SQS sets a *visibility timeout*, a period of time during which Amazon SQS prevents other consumers from receiving and processing the message. The default visibility timeout 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 AWS Management Console and for single or multiple messages using the AWS SDK for Java (and the `SetQueueAttributes`, `GetQueueAttributes`, `ReceiveMessage`, `ChangeMessageVisibility`, and `ChangeMessageVisibilityBatch` actions), see Configuring Visibility Timeout for an Amazon SQS Queue (p. 57).

> **Note**
> For standard queues, the visibility timeout isn't a guarantee against receiving a message twice. For more information, see At-Least-Once Delivery (p. 66).
> 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

Inflight 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 limit 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 limit to the number of inflight messages.

> **Important**
> Limits that apply to inflight 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 inflight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this limit, Amazon SQS returns the `OverLimit` error message. To avoid reaching the limit, 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 limit increase, file a support request.

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

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 (p. 115) 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.

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 a queue for a number of seconds. 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 AWS Management Console or the AWS SDK for Java (and the `SetQueueAttributes` action), see Configuring an Amazon SQS Delay Queue (p. 60).

> **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 (p. 85) 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.
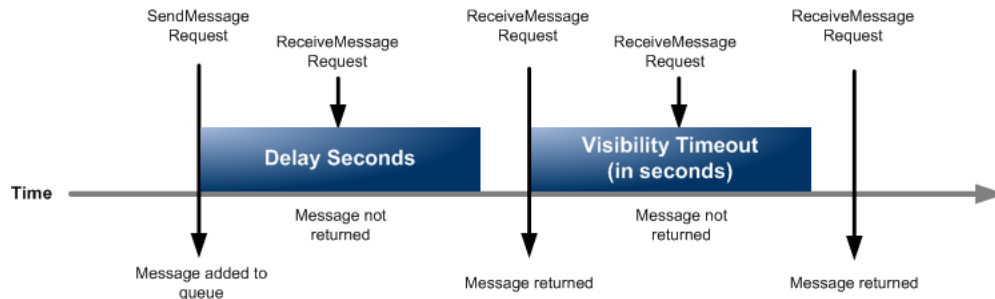
To set delay seconds on *individual messages*, rather than on an entire queue, use message timers (p. 88) to allow Amazon SQS to use the message timer's `DelaySeconds` value instead of the delay queue's `DelaySeconds` value.

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) invisibility period for a message is 0 seconds. The maximum is 15 minutes. For information about sending messages with timers using the AWS Management Console or the AWS SDK for Java (and the `SetQueueAttributes` action), see Sending a Message with a Timer to an Amazon SQS Queue (p. 35).

> **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 (p. 87). A message timer setting for an individual message overrides any `DelaySeconds` value on an Amazon SQS delay queue.

Managing Large Amazon SQS Messages Using Amazon S3

You can use Amazon S3 and the Amazon SQS Extended Client Library for Java to manage Amazon SQS messages. This is especially useful for storing and consuming messages up to 2 GB in size. Unless your application requires repeatedly creating queues and leaving them inactive or storing large amounts of data in your queue, consider using Amazon S3 for storing your data.

You can use the Amazon SQS Extended Client Library for Java library 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 Amazon S3 bucket.
- Get the corresponding message object from an Amazon S3 bucket.
- Delete the corresponding message object from an Amazon S3 bucket.

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

You can use the Amazon SQS Extended Client Library for Java to manage Amazon SQS messages using Amazon S3. However, you can't do this using the AWS CLI, the Amazon SQS console, the Amazon SQS HTTP API, or any of the AWS SDKs—except for the SDK for Java.

Working Java Example for Using Amazon S3 for Large Amazon SQS Messages

Prerequisites

Add the `amazon-sqs-java-extended-client-lib.jar`, `aws-java-sdk-sqs.jar`, and `aws-java-sdk-s3.jar` packages to your Java class path. The following example shows these dependencies in a Maven project `pom.xml` file.

```
<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>amazon-sqs-java-extended-client-lib</artifactId>
        <version>1.0.1</version>
    </dependency>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-sqs</artifactId>
        <version>LATEST</version>
    </dependency>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-s3</artifactId>
        <version>LATEST</version>
    </dependency>
</dependencies>
```

SQSExtendedClientExample.java

The following example code creates an Amazon S3 bucket with a random name and adds a lifecycle rule to permanently delete objects after 14 days. Next, the code creates a queue named `MyQueue` and sends a random message more than 256 KB in size to the queue; the message is stored in the Amazon S3 bucket. Finally, the code consumes the message, returns information about the message, and deletes the message, the queue, and the bucket.

```
/*
 * Copyright 2010-2018 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;
```

```java
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);
    }
```

```
}
```

Working with JMS and Amazon SQS

The Amazon SQS Java Messaging Library is a 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 (p. 113) 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 (p. 99).

```
<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 (p. 15).
  - For information about using the Amazon SQS Java Messaging Library, see Getting Started with the Amazon SQS Java Messaging Library (p. 93).

Getting Started with the Amazon SQS Java Messaging Library

To get started using 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 (p. 101) 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 (p. 72).

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 (p. 70).

```
// 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 (p. 70).

```
// 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 (p. 99) section.

For complete examples of sending and receiving objects, see TextMessageSender.java (p. 101) and SyncMessageReceiver.java (p. 102).

Receiving Messages Asynchronously

In the example in Getting Started with the Amazon SQS Java Messaging Library (p. 93), 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 (p. 93) example. For a complete example of an asynchronous consumer, see `AsyncMessageReceiver.java` in Working Java Example for Using JMS with Amazon SQS Standard Queues (p. 99).

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 (p. 93) 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 (p. 93) 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 (p. 99).

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 (p. 97).

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 (p. 97) 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 Amazon S3 (p. 88).

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 JMS with Amazon SQS standard queues. For more information about working with FIFO queues, see To create a FIFO queue (p. 94), Sending Messages Synchronously (p. 94), and Receiving Messages Synchronously (p. 95). (Receiving messages synchronously is the same for standard and FIFO queues. However, messages in FIFO queues contain more attributes.)

ExampleConfiguration.java

The following Java code example sets the default queue name, the region, and the credentials to be used with the other Java examples.

```
/*
 * Copyright 2010-2018 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.
 *
 */
```

```java
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 2010-2018 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())
```

```java
                              .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.

```java
/*
 * Copyright 2010-2018 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.
```

```java
 *
 */

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 2010-2018 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() ) );

        ReceiverCallback callback = new ReceiverCallback();
        consumer.setMessageListener( callback );

        // No messages are processed until this is called
        connection.start();

        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 2010-2018 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.
```

```java
    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.");
    }
```

```java
    /**
     * 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.

```java
/*
 * Copyright 2010-2018 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 .

```xml
<!--
    Copyright 2010-2018 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 2010-2018 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 );
        }
```

```java
        // 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.

```java
/*
 * Copyright 2010-2018 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)

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 (First-In-First-Out) 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 ensure 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 (p. 86) and Changing a Message's Visibility Timeout (p. 87).

- 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.

> **Note**
> If you need to extend the visibility timeout for longer than 12 hours, consider using AWS 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

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 Long Polling (p. 81).

To ensure 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.

Capturing Problematic Messages

To capture all messages that can't be processed, and to ensure the correctness of CloudWatch metrics, configure a dead-letter queue (p. 82).

- 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 (p. 127)` 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

The expiration of a message is always based on its original enqueue timestamp. When a message is moved to a dead-letter queue (p. 82), the enqueue timestamp remains unchanged. For example, if a message spends 1 day in the original queue before being moved to a dead-letter queue, and the retention period of the dead-letter queue is set to 4 days, the message is deleted from the dead-letter queue after 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.

Avoiding Inconsistent Message Processing

To avoid inconsistent message processing by standard queues, avoid setting the number of maximum receives to 1 when you configure a dead-letter queue.

> **Important**
> In some unlikely scenarios, if you set the number of maximum receives to 1, any time a `ReceiveMessage` call fails, a message might be moved to a dead-letter queue without being received.

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.

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 (p. 174).
- To combine client-side buffering with request batching, use long polling together with the buffered asynchronous client (p. 175) 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 (p. 81).
  - 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 (p. 72).

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 (p. 118)
- Using the Amazon SQS Message Group ID (p. 119)
- Using the Amazon SQS Receive Request Attempt ID (p. 120)

Using the Amazon SQS Message Deduplication ID

The 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**
> Message deduplication applies to an entire queue, not to individual message groups.
> 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 send 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, ensure 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 *minimum* 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 (p. 82).

Using the Amazon SQS Message Group ID

The message group ID 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.

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 Limits

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

Topics

Limits Related to Queues

The following table lists limits related to queues.

Limit	Description
Delay queue	The default (minimum) delay for a queue is 0 seconds. The maximum is 15 minutes.
Inflight messages per queue	For most standard queues (depending on queue traffic and message backlog), there can be a maximum of approximately 120,000 inflight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this limit, Amazon SQS returns the `OverLimit` error message. To avoid reaching the limit, 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 limit increase, file a support request.
	For FIFO queues, there can be a maximum of 20,000 inflight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this limit, Amazon SQS returns no error messages.
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).
	The name of a FIFO queue must end with the `.fifo` suffix. The suffix counts towards the 80-character queue name limit. To determine whether a queue is FIFO (p. 69), you can check whether the queue name ends with the suffix.
Queue tag	We don't recommend adding more than 50 tags to a queue.

Limit	Description
	The tag `Key` is required, but the tag `Value` is optional.
	The tag `Key` and tag `Value` are case-sensitive.
	The tag `Key` and tag `Value` can include Unicode alphanumeric characters in UTF-8 and whitespaces. The following special characters are allowed: `_ . : / = + - @`
	The tag `Key` or `Value` must not include the reserved prefix `aws:` (you can't delete tag keys or values with this prefix).
	The maximum tag `Key` length is 128 Unicode characters in UTF-8. The tag `Key` must not be empty or null.
	The maximum tag `Value` length is 256 Unicode characters in UTF-8. The tag `Value` may be empty or null.
	Tagging actions are limited to 5 TPS per AWS account. If your application requires a higher throughput, file a support request.

Limits Related to Messages

The following table lists limits related to messages.

Limit	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 (p. 176) in the Amazon SQS Batch Actions (p. 174) 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 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 transactions per second (TPS) per action. • By default, FIFO queues support up to 3,000 messages per second with batching (p. 174). To request a limit increase, file a support request.

Limit	Description
	• FIFO queues support up to 300 messages per second (300 send, receive, or delete operations per second) without batching.
Message timer	The default (minimum) invisibility period for a message is 0 seconds. The maximum is 15 minutes.
Message size	The minimum message size is 1 byte (1 character). The maximum is 262,144 bytes (256 KB). To send messages larger than 256 KB, you can use the Amazon SQS Extended Client Library for Java. 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.
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 limit is 8,192 bytes, 20 statements, 50 principals, or 10 conditions. For more information, see Limits Related to Policies (p. 123).

Limits Related to Policies

The following table lists limits related to policies.

Name	Maximum
Actions	7
Bytes	8,192
Conditions	10
Principals	50
Statements	20

Monitoring, Logging, and Automating Amazon SQS Queues

This section provides information about monitoring, logging, and automating Amazon SQS queues.

Topics
- Monitoring Amazon SQS Queues Using CloudWatch (p. 124)
- Logging Amazon SQS API Calls Using AWS CloudTrail (p. 130)
- Automating Notifications from AWS Services to Amazon SQS using CloudWatch Events (p. 134)

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 (p. 124), the CloudWatch console (p. 125), using the AWS CLI (p. 126), or using the CloudWatch API (p. 126). You can also set CloudWatch alarms (p. 126) for Amazon SQS metrics.

CloudWatch metrics for your Amazon SQS queues are automatically collected and pushed to CloudWatch every five minutes. 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.

> **Note**
> There is no charge for the Amazon SQS metrics reported in CloudWatch. They're provided as part of the Amazon SQS service.
> Detailed monitoring (or one-minute metrics) is currently unavailable for Amazon SQS. Making requests to CloudWatch at this resolution might return no data.
> CloudWatch metrics are supported for both standard and FIFO queues.

Topics
- Accessing CloudWatch Metrics for Amazon SQS (p. 124)
- Creating CloudWatch Alarms for Amazon SQS Metrics (p. 126)
- Available CloudWatch Metrics for Amazon SQS (p. 127)

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 (p. 124), the CloudWatch console (p. 125), using the AWS CLI (p. 126), or using the CloudWatch API (p. 126). You can also set CloudWatch alarms (p. 126) 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 (p. 127).

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 (p. 127).

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 (p. 125) procedure.

Amazon CloudWatch Console

1. Sign in to the CloudWatch console.

2. On the navigation panel, choose **Metrics**.

3. Select the **SQS** metric namespace.
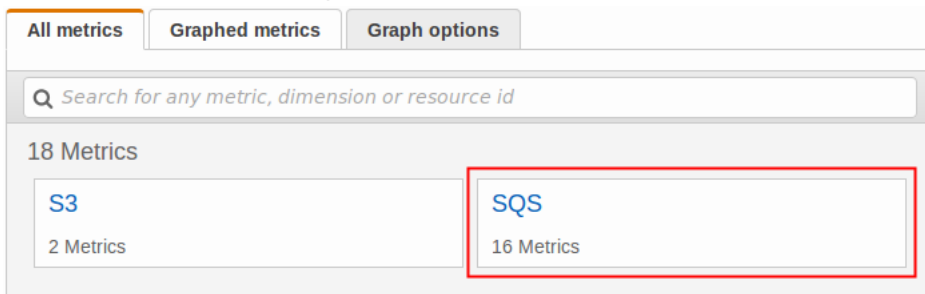
4. Select the **Queue Metrics** metric dimension.
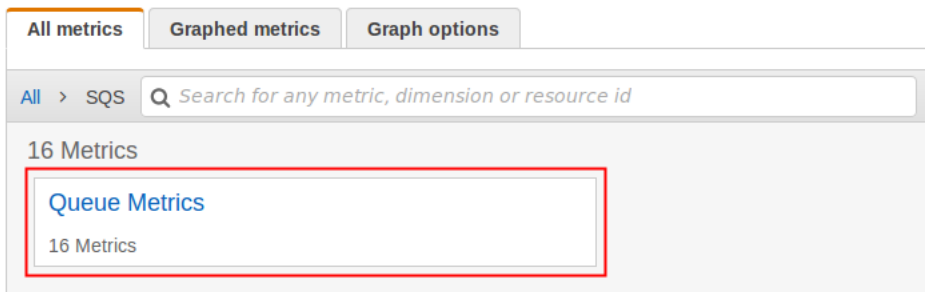
5. You can now examine your Amazon SQS metrics:

   - 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**.
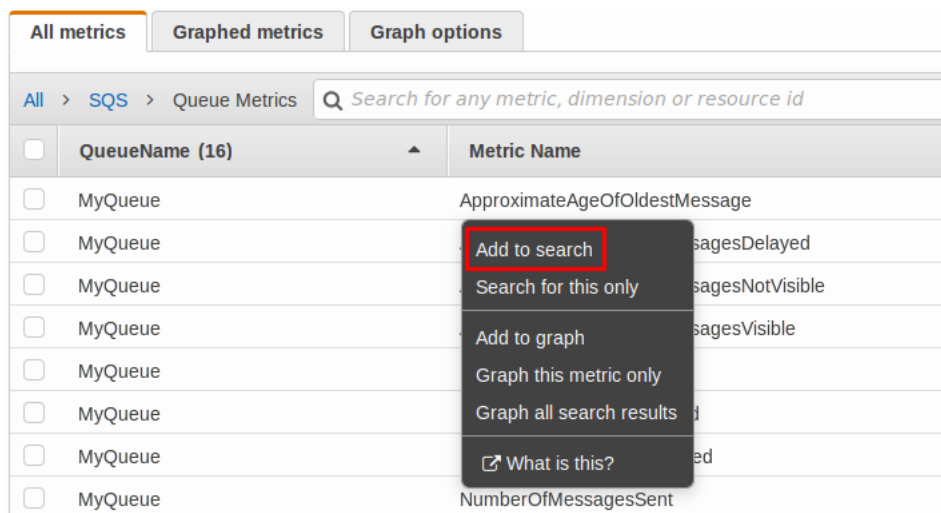
For more information and additional options, see Graph Metrics and Using Amazon CloudWatch Dashboards in the *Amazon CloudWatch User Guide*.

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 (p. 127).

   In the following example, the selection is for an alarm for the `NumberOfMessagesSent` metric for the `MyQueue` queue. The alarm triggers when .
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** For dead-letter queues, the value of `ApproximateAgeOfOldestMessage` is the longest time that a message has been in the queue. 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. Units: *Count*

Metric	Description
	Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)
`ApproximateNumberOfMessagesNotVisible`	The number of messages that are in flight. Messages are considered to be *in flight* if they have been sent to a client but have not yet been deleted or have not yet reached the end of their visibility window. Units: *Count* Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)
`ApproximateNumberOfMessagesVisible`	The number of messages available for retrieval from the queue. Units: *Count* Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)
`NumberOfEmptyReceives`	The number of `ReceiveMessage` API calls that did not return a message. Units: *Count* Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)

Metric	Description
NumberOfMessagesDeleted	The number of messages deleted from the queue. 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 you call the DeleteMessage action again using the same receipt handle, a success status is returned, increasing the value of the NumberOfMessagesDeleted metric.
NumberOfMessagesReceived	The number of messages returned by calls to the ReceiveMessage action. Units: *Count* Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)
NumberOfMessagesSent	The number of messages added to a queue. Units: *Count* Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console)

Metric	Description
`SentMessageSize`	The size of messages added to a queue. Units: *Bytes* Valid Statistics: Average, Minimum, Maximum, Sum, Data Samples (displays as Sample Count in the Amazon SQS console) Note that `SentMessageSize` does not display as an available metric in the CloudWatch console until at least one message is sent to the corresponding queue.

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`.

Logging Amazon SQS API Calls Using AWS CloudTrail

Amazon SQS is integrated with AWS CloudTrail, a service that provides a record of the Amazon SQS calls that a user, role, or AWS service makes. CloudTrail captures API calls related to Amazon SQS queues as events, including calls from the Amazon SQS console and code calls from Amazon SQS APIs. For more information about CloudTrail, see the *AWS CloudTrail User Guide*.

> **Note**
> CloudTrail logging is supported for both standard and FIFO queues.

Using the information that CloudTrail collects, you can identify a specific request to an Amazon SQS API, the IP address of the requester, the requester's identity, the date and time of the request, and so on. If you configure a *trail*, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket. If you don't configure a trail, you can view the most recent events in the event history in the CloudTrail console. For more information, see Overview for Creating a Trail in the *AWS CloudTrail User Guide*.

Amazon SQS Information in CloudTrail

When you create your AWS account, CloudTrail is enabled. When a supported Amazon SQS event activity occurs, it is recorded in a CloudTrail event 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*.

A trail allows CloudTrail to deliver log files to an Amazon S3 bucket. You can create a trail to keep an ongoing record of events in your AWS account. By default, when you create a trail using the AWS Management Console, the trail applies to all AWS Regions. The trail logs events from all AWS Regions and delivers log files to the specified Amazon S3 bucket. You can also configure other AWS services to further analyze and act on the event data collected in CloudTrail logs. For more information, see the following topics in the *AWS CloudTrail User Guide*:

- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail

- Receiving CloudTrail Log Files from Multiple Regions
- Receiving CloudTrail Log Files from Multiple Accounts

Amazon SQS supports logging the following actions:

- `AddPermission`
- `CreateQueue`
- `DeleteQueue`
- `PurgeQueue`
- `RemovePermission`
- `SetQueueAttributes`

Every event or log entry contains information about the requester. This information helps you determine the following:

- Was the request made with root or IAM user credentials?
- Was the request made with temporary security credentials for a role or a federated user?
- Was the request made by another AWS service?

For more information, see CloudTrail userIdentity Element in the *AWS CloudTrail User Guide*.

Example Amazon SQS Log File Entries

CloudTrail log files contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries aren't guaranteed to be in any particular order. That is, they're not an ordered stack trace of the public API calls.

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"
  }
 ]
}
```

Automating Notifications from AWS Services to Amazon SQS using CloudWatch Events

Amazon CloudWatch Events 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 CloudWatch Events 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.

CloudWatch Events 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 the *Amazon CloudWatch Events User Guide*.

Amazon SQS Security

This section provides information about Amazon SQS security, authentication and access control, and the Amazon SQS Access Policy Language.

Topics

Authentication and Access Control for Amazon SQS

Access to Amazon SQS requires credentials that AWS can use to authenticate your requests. These credentials must have permissions to access AWS resources, such an Amazon SQS queues and messages. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon SQS to help secure your resources by controlling access to them.

Topics

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single 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 do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

- **IAM user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create a queue in Amazon SQS). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. Amazon SQS supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

- **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

  - **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

  - **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 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*.

Access Control

Amazon SQS has its own resource-based permissions system that uses policies written in the same language used for AWS Identity and Access Management (IAM) policies. This means that you can achieve similar things with Amazon SQS policies and IAM policies.

> **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*.
> Cross-account permissions don't apply to the following actions:
>
> - `AddPermission`
> - `CreateQueue`
> - `DeleteQueue`
> - `ListQueues`
> - `ListQueueTags`
> - `RemovePermission`
> - `SetQueueAttributes`
> - `TagQueue`
> - `UntagQueue`
>
> Currently, Amazon SQS supports only a limited subset of the condition keys available in IAM. For more information, see Actions and Resource Reference (p. 159).

The following sections describe how to manage permissions for Amazon SQS. We recommend that you read the overview first.

Overview of Managing Access Permissions to Your Amazon Simple Queue Service Resource

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

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 Actions and Resource Reference (p. 159).

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, an IAM 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 an IAM 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 polices) 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.

- **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.

SQS Policy on queue_xyz

Allow who:
   AWS account 1
   AWS account 2

Actions: *

Resource:
   123456789012/queue_xyz
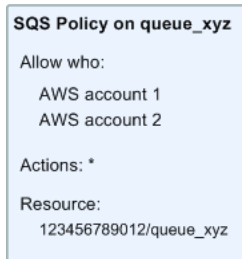
With the introduction of IAM and the concepts of *Users* and *Amazon Resource Names (ARNs)*, a few things have changed about SQS policies. The following diagram and table describe the changes.

SQS Policy on queue_xyz

Allow who:
   AWS account 1
   AWS account 2

1 — User Bob
   User Susan   (in your own account)

2 — Actions: *

Resource:
3 — arn:aws:sqs:*:123456789012/queue_xyz
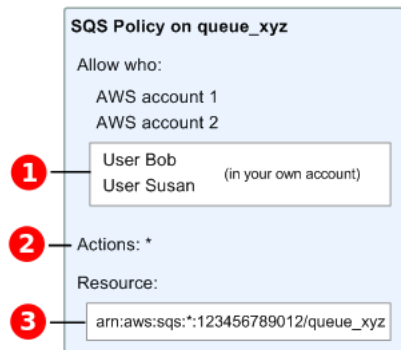
❶ In addition to specifying which AWS Accounts have access to a queue, you can specify which users *in your own AWS account* have access to the queue. If the users are in different accounts, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.

❷ The subset of actions included in * has expanded. For a list of allowed actions, see Actions and Resource Reference (p. 159).

❸ 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 (p. 137).

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 Identity-Based (IAM) Policies for Amazon SQS (p. 141), and Identities (Users, Groups, and Roles) in the *IAM User Guide*.

Resource-Based Policies

Other AWS services, such as Amazon S3, support resource-based permissions policies. For example, to manage access permissions for an S3 bucket, you can attach a policy the S3 bucket.

Amazon SQS doesn't support resource-level permissions in identity-based policies (attached to a user or role), in which you can specify resources on which users are allowed to perform specified actions. For more information, see Overview of AWS IAM Permissions in the *IAM User Guide*.

Specifying Policy Elements: Actions, Effects, Resources, and Principals

For each Amazon Simple Queue Service resource (p. 137), 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 Actions and Resource Reference (p. 159).

Specifying Conditions in a Policy

When you grant permissions, you can use the Amazon SQS Access Policy Language to specify the conditions for when a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Condition in the *IAM User Guide*.

To express conditions, you use predefined condition keys. There are no condition keys specific to Amazon SQS. However, there are AWS-wide condition keys that you can use with Amazon SQS. Currently, Amazon SQS supports only a limited subset of the condition keys available in IAM:

- `aws:CurrentTime`
- `aws:EpochTime`
- `aws:SecureTransport`
- `aws:SourceArn`
- `aws:SourceIP`
- `aws:UserAgent`
- `aws:MultiFactorAuthAge`
- `aws:MultiFactorAuthPresent`
- `aws:TokenAge`

Using Identity-Based (IAM) Policies for 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 Permissions to Your Amazon Simple Queue Service Resource (p. 137).
With the exception of `ListQueues`, all Amazon SQS actions support resource-level permissions. For more information, see Actions and Resource Reference (p. 159).

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.
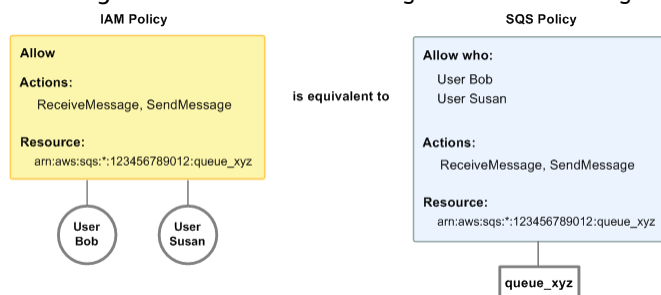
Note

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 (p. 149). The following diagram illustrates the concept.
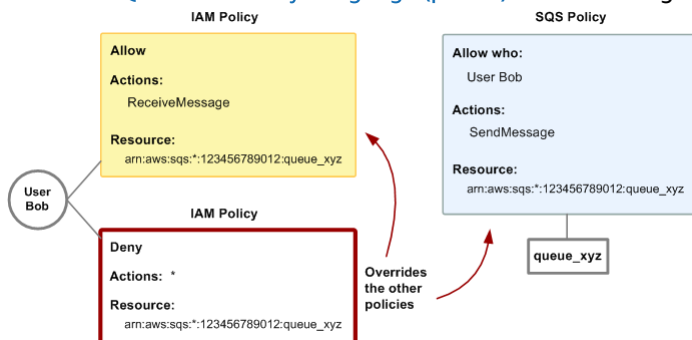
You can also add an additional statement to the Amazon SQS policy that denies Bob any type of access to the queue. It has the same effect as adding an IAM policy that denies Bob access to the queue. For examples of policies that cover Amazon SQS actions and resources, see Advanced Amazon SQS Policy Examples (p. 145). For more information about writing Amazon SQS policies, see Using Custom Policies with the Amazon SQS Access Policy Language (p. 149).

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.

AWS-Managed (Predefined) Policies for Amazon SQS

AWS addresses many common use cases by providing standalone AWS managed IAM policies. These AWS managed policies simplify working with permissions by granting the permissions necessary for common use cases. For more information, see AWS Managed Policies in the *IAM User Guide*.

The following AWS managed policies (that you can attach to users in your account) are specific to Amazon SQS:

- **AmazonSQSReadOnlyAccess** – Grants read-only access to Amazon SQS queues using the AWS Management Console.
- **AmazonSQSFullAccess** – Grants full access to Amazon SQS queues using the AWS Management Console.

You can search and review available policies on the IAM console. You can also create your own custom IAM policies to allow permissions for Amazon SQS actions and queues. You can attach these custom policies to the IAM users or groups that require permissions.

Basic Amazon SQS Policy Examples

The following examples provide an introduction to Amazon SQS permission policies.

Example 1: 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 `bob_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:bob_queue_*"
   }]
}
```

Example 2: 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 onlt the `ReceiveMessage` permission, first add the `ReceiveMessage` permission and then remove the * permission.

Example 3: 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": "*"
    }]
}
```

Example 4: 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"
    }]
}
```

Advanced Amazon SQS Policy Examples

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 User Name

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
- CreateQueue
- DeleteQueue
- ListQueues
- ListQueueTags
- RemovePermission
- SetQueueAttributes
- 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 `192.168.143.0/24` CIDR range.

```
{
   "Version": "2012-10-17",
   "Id": "Queue1_Policy_UUID",
   "Statement": [{
      "Sid":"Queue1_AnonymousAccess_AllActions_WhitelistIP",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "sqs:*",
      "Resource": "arn:aws:sqs:*:111122223333:queue1",
      "Condition" : {
         "IpAddress" : {
            "aws:SourceIp":"192.168.143.0/24"
         }
      }
   }]
}
```

Example 8: Whitelist and Blacklist Permissions for Users in Different CIDR Ranges

The following example policy has two statements:

- The first statement grants all users (anonymous users) in the `192.168.143.0/24` CIDR range (except for `192.168.143.188`) permission to use the `SendMessage` action for the queue named `111122223333/queue1`.
- The second statement blacklists all users (anonymous users) in the `10.1.2.0/24` 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":"192.168.143.0/24"
            },
            "NotIpAddress" : {
               "aws:SourceIp":"192.168.143.188/32"
            }
          }
      }, {
          "Sid":"Queue1_AnonymousAccess_AllActions_IPLimit_Deny",
          "Effect": "Deny",
          "Principal": "*",
          "Action": "sqs:*",
          "Resource": "arn:aws:sqs:*:111122223333:queue1",
          "Condition" : {
            "IpAddress" : {
               "aws:SourceIp":"10.1.2.0/24"
            }
          }
      }]
}
```

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 (p. 149)
- Amazon SQS Access Control Process Workflow (p. 150)
- Amazon SQS Access Policy Language Key Concepts (p. 151)
- Amazon SQS Access Policy Language Evaluation Logic (p. 152)
- Relationships Between Explicit and Default Denials in the Amazon SQS Access Policy Language (p. 154)
- Custom Amazon SQS Access Policy Language Examples (p. 155)

Amazon SQS Access Control Architecture

The following diagram describes the access control for your Amazon SQS resources.
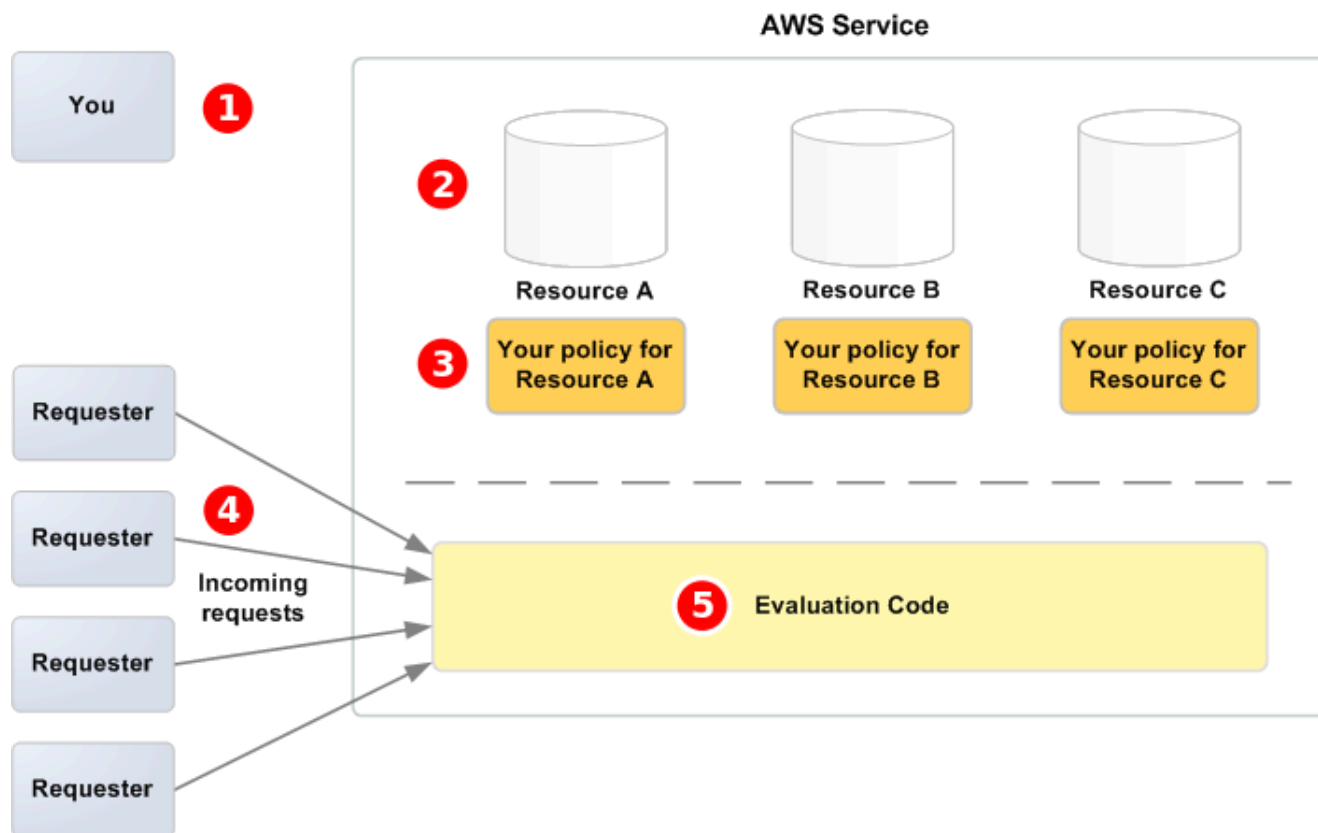
❶ You, the resource owner.

❷ Your resources contained within the AWS service (for example, Amazon SQS queues).

❸ Your policies. It is a good practice to have one policy per resource The AWS service itself provides an API you use to upload and manage your policies.

❹ Requesters and their incoming requests to the AWS service.

❺ 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.
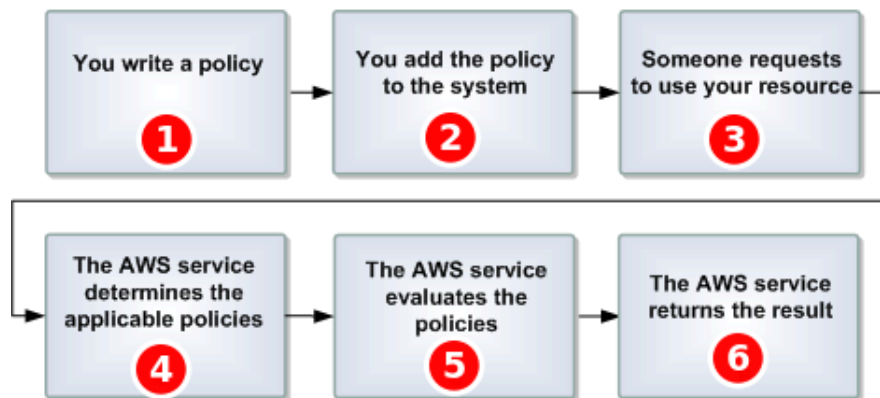
❶ You write an Amazon SQS policy for your queue.

❷ 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.

❸ Someone sends a request to use your Amazon SQS queue.

❹ Amazon SQS examines all available Amazon SQS policies and determines which ones are applicable.

❺ Amazon SQS evaluates the policies and determines whether the requester is allowed to use your queue.

❻ 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** (p. 152) that has **effect** (p. 151) set to `allow`.

action

> The activity that the **principal** (p. 152) has permission to perform, typically a request to AWS.

default-deny

> The result of a **statement** (p. 152) that that has no **allow** (p. 151) or **explicit-deny** (p. 151) settings.

condition

> Any restriction or detail about a **permission** (p. 152). Typical conditions are related to date and time and IP addresses.

effect

> The result that you want the **statement** (p. 152) of a **policy** (p. 152) 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** (p. 151), **allow** (p. 151), and **explicit-deny** (p. 151).

explicit-deny

> The result of a **statement** (p. 152) that has **effect** (p. 151) 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** (p. 152).

issuer

The user who writes a **policy** (p. 152) 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** (p. 151) and a **key** (p. 152).

policy

The document that acts as a container for one or more **statements (p. 152)**.

Amazon SQS uses the policy to determine whether to grant access to a user for a resource.

principal

The user who receives **permission** (p. 152) in the **policy** (p. 152).

resource

The object that the **principal** (p. 152) requests access to.

statement

The formal description of a single permission, written in the access policy language as part of a broader **policy** (p. 152) document.

requester

The user who sends a request for access to a **resource** (p. 152).

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 (p. 151)* overrides any *default-deny (p. 151)*.
- An *explicit-deny (p. 151)* 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.
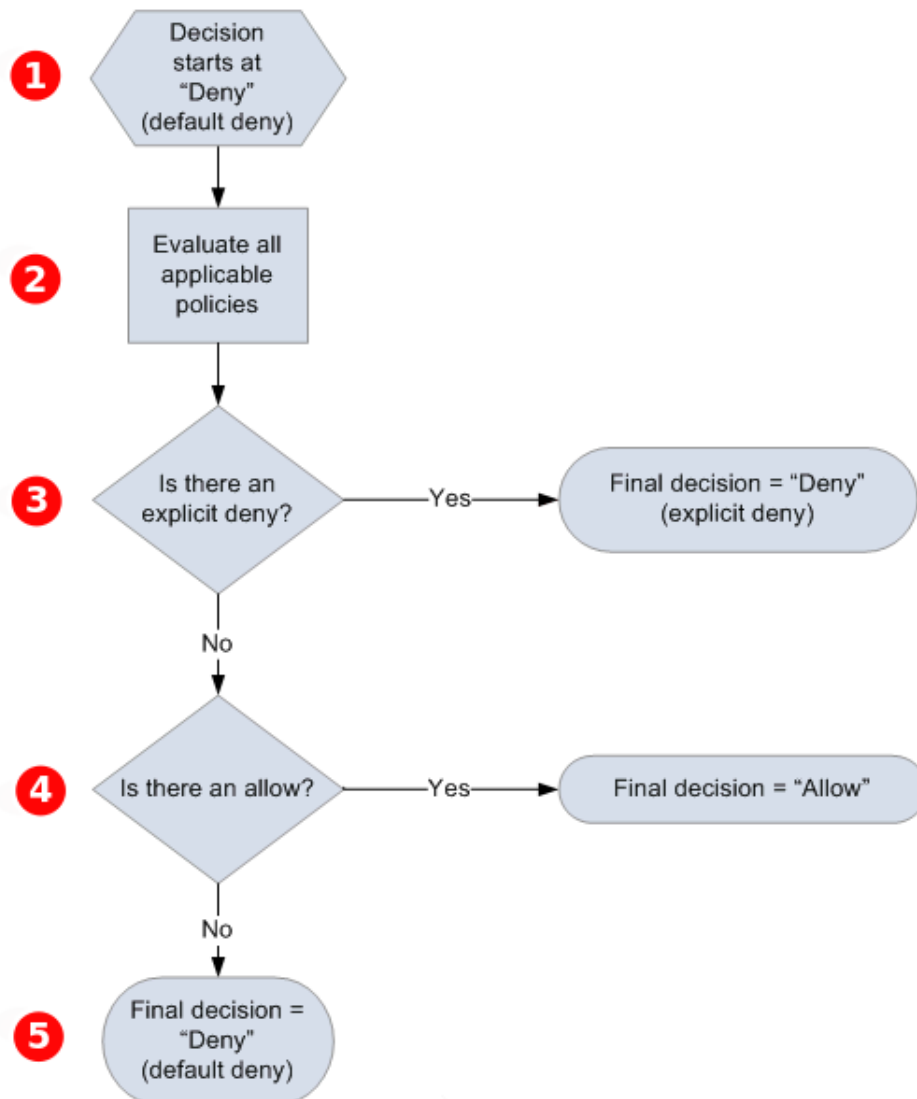
❶ The decision starts with a **default-deny**.

❷ 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.

❸ 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.

❹ 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).

❺ 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 (p. 151)*. 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 (p. 151)* or an *explicit-deny (p. 151)* based on the value of the *effect (p. 151)* 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.

Policy A1

Effect = **Allow**

Condition:
if request is NOT from Antarctica
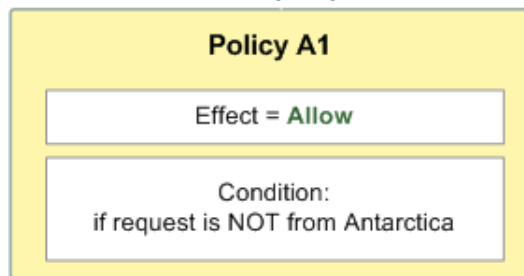
If a user sends a request from the U.S., the condition is met (the request isn't from Antarctica), and the request results in an **allow**. However, if a user sends a request from Antarctica, the condition isn't met and the request defaults to a **default-deny**. You can change the result to an **explicit-deny** by writing Policy A2 that explicitly denies a request if it comes from Antarctica. The following diagram illustrates the policy.

Policy A2

Effect = **Deny**

Condition:
if request is from Antarctica
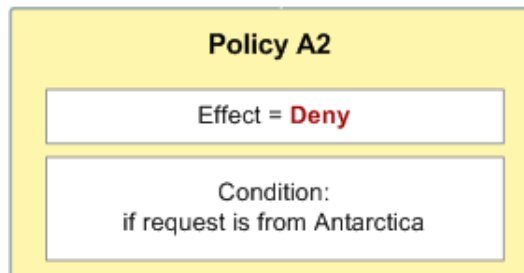
If a user sends a request from Antarctica, the condition is met and the request results in an **explicit-deny**.

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.
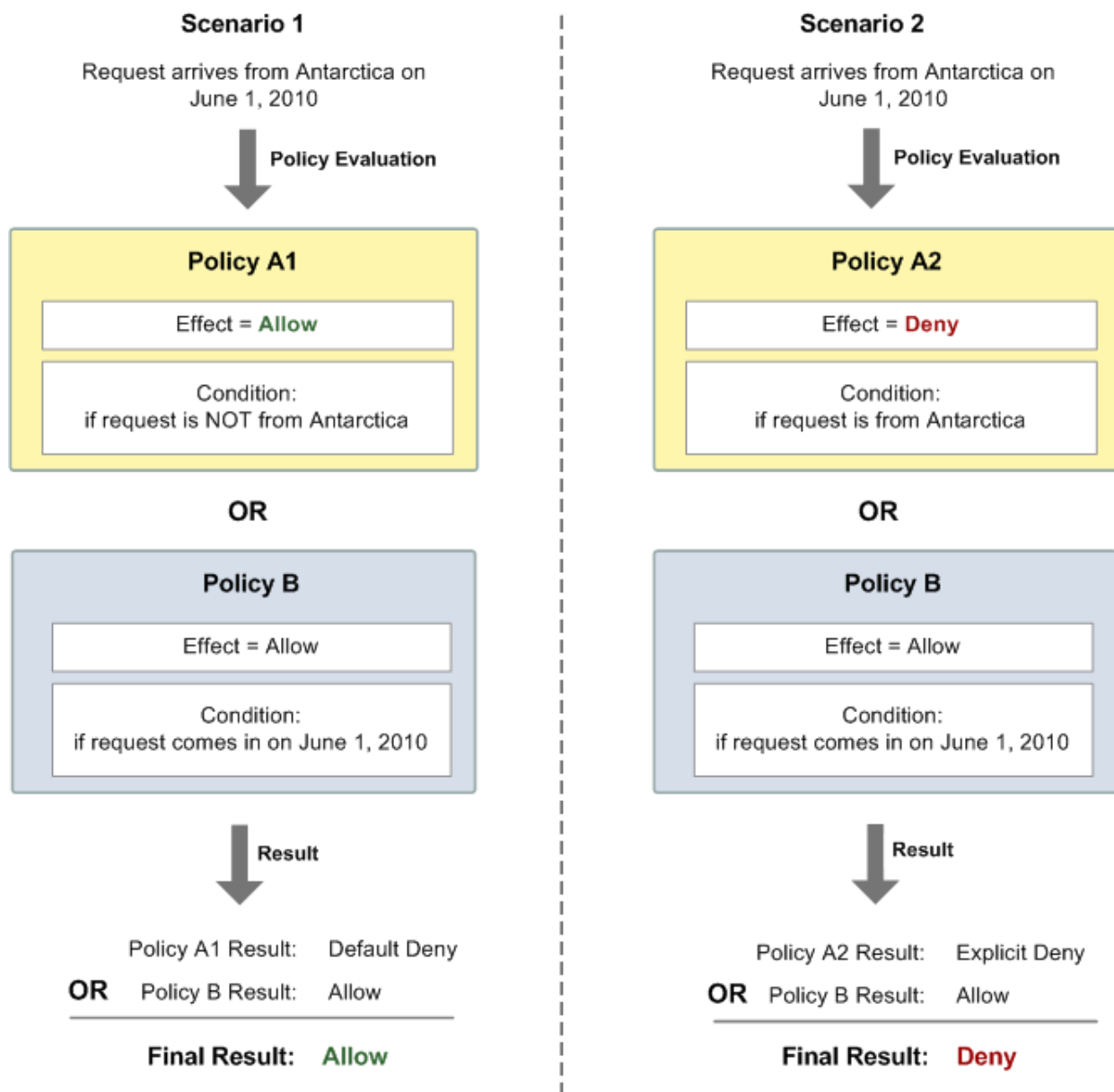
In Scenario 1, Policy A1 results in a **default-deny** and Policy B results in an allo because the policy allows requests that come in on June 1, 2010. The **allow** from Policy B overrides the **default-deny** from Policy A1, and the request is allowed.

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.

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 "Give Permission to One or More Accounts (p. 156)" 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 "Give Permission to One Account (p. 155)" 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"
    }]
}
```

Using Temporary Security Credentials

In addition to creating IAM 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 (IAM users). 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
```

Amazon SQS API Permissions: Actions and Resource Reference

When you set up Access Control (p. 136) 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 only a limited subset of the condition keys available in IAM:

- `aws:CurrentTime`
- `aws:EpochTime`
- `aws:SecureTransport`
- `aws:SourceArn`
- `aws:SourceIP`
- `aws:UserAgent`
- `aws:MultiFactorAuthAge`
- `aws:MultiFactorAuthPresent`
- `aws:TokenAge`

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`*

Protecting Data Using Server-Side Encryption (SSE) and AWS KMS

SSE lets you transmit sensitive data in encrypted queues. SSE protects the contents of messages in Amazon SQS queues using keys managed in AWS Key Management Service (AWS KMS). For information about managing SSE using the AWS Management Console or the AWS SDK for Java (and the `CreateQueue`, `SetQueueAttributes`, and `GetQueueAttributes` actions), see the following tutorials:

- Creating an Amazon SQS Queue with SSE (p. 19)
- Configuring SSE for an Existing Amazon SQS Queue (p. 48)
- Enable Compatibility between AWS Services and Queues with SSE (p. 166)

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.
You can't associate an encrypted queue (p. 161) that uses an AWS managed Customer Master Key for Amazon SQS with 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

Other features of AWS services or third-party services that send notifications to Amazon SQS aren't compatible with SSE, despite allowing you to set an encrypted queue as a target:

- AWS IoT Rule Actions

For information about compatibility of other services with encrypted queues, see Enable Compatibility between AWS Services and Queues with SSE (p. 166) 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 (p. 163) 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 customer master keys (CMKs) (p. 163) yourself.
- You can also use the AWS managed CMK 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* and the AWS Key Management Service Cryptographic Details whitepaper.

Topics

What Does SSE for Amazon SQS Encrypt?

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. 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 (p. 82) 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 a 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 data encryption 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* and Envelope Encryption 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 How Does the Data Key Reuse Period Work? (p. 163).

> **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.

Customer Master Key ID

The alias, alias ARN, key ID, or key ARN of an AWS managed customer master key (CMK) or a custom CMK—in your account or in another account. While the alias of the AWS managed CMK for Amazon SQS is always `alias/aws/sqs`, the alias of a custom CMK can, for example, be `alias/`*MyAlias*. You can use these CMKs to protect the messages in Amazon SQS queues.

> **Note**
> Keep the following in mind:
> - If you don't specify a custom CMK, Amazon SQS uses the AWS managed CMK for Amazon SQS. For instructions on creating custom CMKs, see Creating Keys in the *AWS Key Management Service Developer Guide*.
> - The first time you use the AWS Management Console to specify the AWS managed CMK for Amazon SQS for a queue, AWS KMS creates the AWS managed CMK 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 CMK for Amazon SQS.

You can create CMKs, define the policies that control how CMKs can be used, and audit CMK usage using the **Encryption Keys** section of the AWS KMS console or using AWS KMS actions. For more information about CMKs, see Customer Master Keys in the *AWS Key Management Service Developer Guide*. For more examples of CMK identifiers, see KeyId in the *AWS Key Management Service API Reference*.

> **Important**
> There are additional charges for using AWS KMS. For more information, see How Do I Estimate My AWS KMS Usage Costs? (p. 164) and AWS Key Management Service Pricing.

How Does the Data Key Reuse Period Work?

Amazon SQS uses a single customer master key (either the AWS managed CMK for Amazon SQS or a custom CMK) to provide envelope encryption and decryption of multiple Amazon SQS messages during

the *data key reuse period*. To make the most of the data key reuse period (p. 163), keep the following in mind:

- 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.
- Principals (AWS accounts or IAM 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:
  - When you send messages using the `SendMessage` or `SendMessageBatch` action, Amazon SQS typically calls the AWS KMS `GenerateDataKey` and `Decrypt` actions once per every data key reuse period.

    Note
    For each data key that AWS KMS generates, SSE calls the `Decrypt` action to verify the integrity of the data key before using it.

  - When you receive messages using the `ReceiveMessage` action, Amazon SQS typically calls the AWS KMS `Decrypt` action once per every data key reuse period.

How Do I Estimate My AWS KMS Usage Costs?

To predict costs and better understand your AWS bill, you might want to know how often Amazon SQS uses your customer master key (CMK).

> **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 (p. 163) (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 How Does the Data Key Reuse Period Work? (p. 163)
> If the producer and consumer have different IAM 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
```

What AWS KMS Permissions Do I Need to Use SSE for Amazon SQS?

Before you can use SSE, you must configure AWS KMS key policies to allow encryption of queues and encryption and decryption of messages. For examples and more information about AWS KMS permissions, see AWS KMS API Permissions: Actions and Resources Reference in the *AWS Key Management Service Developer Guide*.

> **Note**
> You can also manage permissions for KMS keys using IAM policies. For more information, see Using IAM Policies with AWS KMS.
> While you can configure global permissions to send to and receive from Amazon SQS, AWS KMS requires explicitly naming the full ARN of CMKs in specific regions in the `Resource` section of an IAM policy.

You must also ensure that the key policies of the customer master key (CMK) allow the necessary permissions. To do this, name the principals that produce and consume encrypted messages in Amazon SQS as users in the CMK key policy.

Alternatively, you can specify the required AWS KMS actions and CMK ARN in an IAM policy assigned to the principals which produce and consume encrypted messages in Amazon SQS. For more information, see Managing Access to AWS KMS CMKs in the *AWS Key Management Service Developer Guide*.

Example 1: Allow a User to Send Single or Batched Messages to a Queue with SSE

The producer must have the `kms:GenerateDataKey` and `kms:Decrypt` permissions for the customer master key (CMK).

```
{
   "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",
            "sqs:SendMessageBatch"
        ],
        "Resource": "arn:aws:sqs:*:123456789012:MyQueue"
    }]
}
```

Example 2: Allow a User to Receive Messages from a Queue with SSE

The consumer must have the `kms:Decrypt` permission for any customer master key (CMK) that is used to encrypt the messages in the specified queue. If the queue acts as a dead-letter queue (p. 82), the consumer must also have the `kms:Decrypt` permission for any CMK that is used to encrypt the messages in the source queue.

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

Example 3: Enable Compatibility between AWS Services Such as Amazon CloudWatch Events, Amazon S3, and Amazon SNS and Queues with SSE

To allow Amazon CloudWatch Events, Amazon S3 event notifications, or Amazon SNS topic subscriptions to work with encrypted queues, you must perform the following steps:

1. Create a customer master key (CMK).
2. To allow the AWS service feature to have the `kms:GenerateDataKey*` and `kms:Decrypt` permissions, add the following statement to the policy of the CMK.

   Note

   - For Amazon CloudWatch Events, use `events`
   - For Amazon S3 event notifications, use `s3`
   - For Amazon SNS topic subscriptions, use `sns`

```
{
    "Version": "2012-10-17",
        "Statement": [{
            "Effect": "Allow",
            "Principal": {
                "Service": "service.amazonaws.com"
            },
            "Action": [
                "kms:GenerateDataKey*",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }]
}
```

3. Create a new SSE queue (p. 19) or configure an existing SSE queue (p. 48) using the ARN of your CMK.

Learn More

- Subscribe to a Topic in the *Amazon Simple Notification Service Developer Guide*
- Creating a CloudWatch Events Rule That Triggers on an Event in the *Amazon CloudWatch Events User Guide*
- Configuring Amazon S3 Event Notifications in the *Amazon Simple Storage Service Developer Guide*

Errors

When you work with Amazon SQS and AWS KMS, you might encounter errors. The following list describes the errors and possible troubleshooting solutions.

KMSAccessDeniedException

The ciphertext references a key that doesn't exist or that you don't have access to.

HTTP Status Code: 400

KMSDisabledException

The request was rejected because the specified CMK isn't enabled.

HTTP Status Code: 400

KMSInvalidStateException

The request was rejected because the state of the specified resource isn't valid for this request. For more information, see How Key State Affects Use of a Customer Master Key in the *AWS Key Management Service Developer Guide*.

HTTP Status Code: 400

KMSNotFoundException

The request was rejected because the specified entity or resource can't be found.

HTTP Status Code: 400

KMSOptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

KMSThrottlingException

The request was denied due to request throttling. For more information about throttling, see Limits in the *AWS Key Management Service Developer Guide*.

HTTP Status Code: 400

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*.

Topics

Making Query API Requests

In this section you'll 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 region-specific Amazon SQS endpoints, see the *Amazon Web Services General Reference*.

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 (p. 76)), 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` requests send query parameters as a form in the body of an HTTP request.

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

```
POST /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` `POST` 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.

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 Create an AWS Account (p. 5) and Create an IAM User (p. 5)). 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.
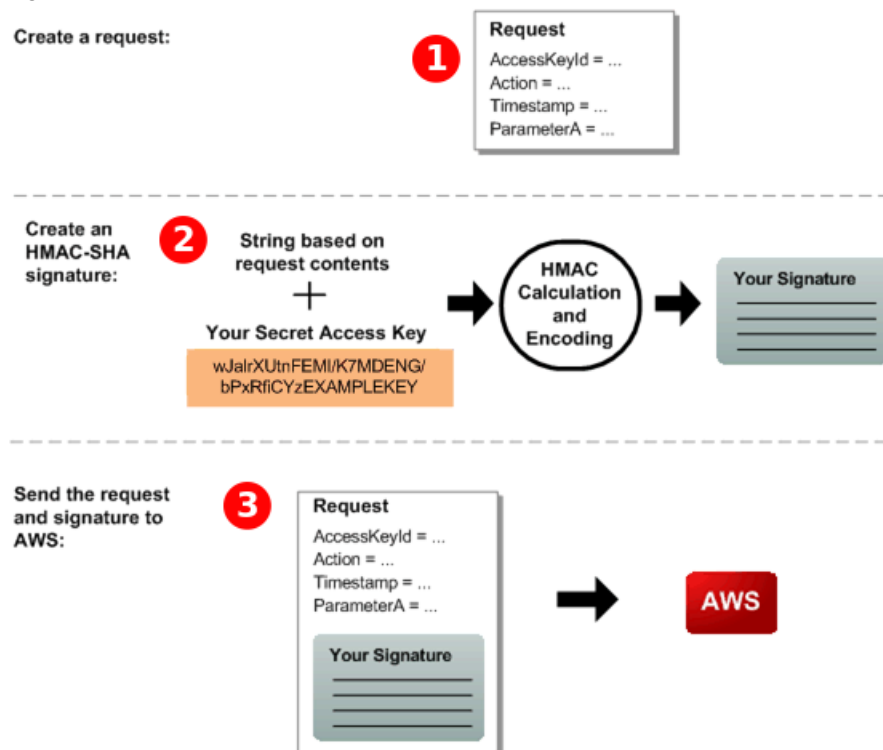
1. Construct a request to AWS.
2. Calculate a keyed-hash message authentication code (HMAC-SHA) signature using your Secret Access Key.
3. Include the signature and your Access Key ID in the request, and then send the request to AWS.

Part 2: The Response from AWS

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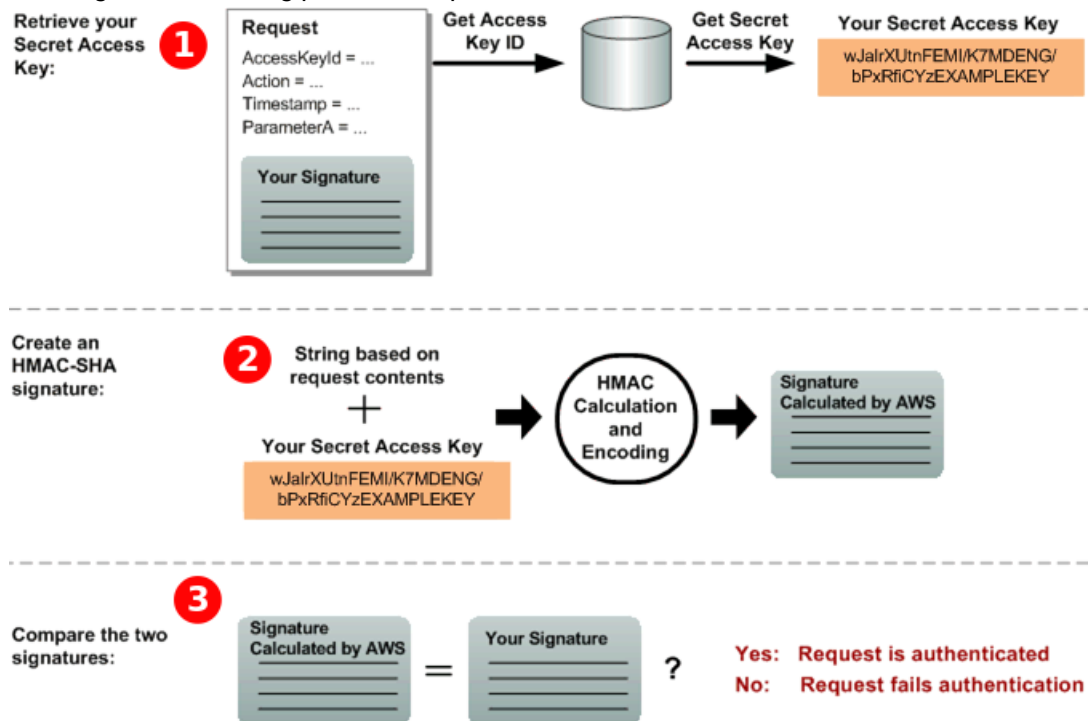
1. AWS uses the Access Key ID to look up your Secret Access Key.
2. AWS generates a signature from the request data and the Secret Access Key, using the same algorithm that you used to calculate the signature you sent in the request.
3. One of the following happens:

   - 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.

Interpreting 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 Response Structure (p. 173)
- Error Response Structure (p. 174)

Successful Response Structure

If the request is successful, the main response element is named after the action, with `Response` appended (`ActionNameResponse`).

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 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>
```

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>
```

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 (p. 175)
- Increasing Throughput using Horizontal Scaling and Action Batching (p. 178)

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 (p. 81) can also help increase throughput. For more information, see
Increasing Throughput using Horizontal Scaling and Action Batching (p. 178).

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 (p. 175)
- Configuring AmazonSQSBufferedAsyncClient (p. 176)

Using AmazonSQSBufferedAsyncClient

Before you begin, complete the steps in Setting Up Amazon SQS (p. 5).

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 SendMessageResult sendResult = bufferedSqs.sendMessageAsync(request);

final ReceiveMessageRequest receiveRq = new ReceiveMessageRequest()
    .withMaxNumberOfMessages(1)
    .withQueueUrl(queueUrl);
final ReceiveMessageResult rx = bufferedSqs.receiveMessage(receiveRq);
```

Configuring AmazonSQSBufferedAsyncClient

`AmazonSQSBufferedAsyncClient` is preconfigured with settings that work for most use cases. You can further configure `AmazonSQSBufferedAsyncClient`, for example:

1. Create an instance of the `QueueBufferConfig` class with the required configuration parameters.
2. Provide the instance to the `AmazonSQSBufferedAsyncClient` constructor.

```
// Create the basic Amazon SQS async client
final AmazonSQSAsync sqsAsync = new AmazonSQSAsyncClient();

final QueueBufferConfig config = new QueueBufferConfig()
    .withMaxInflightReceiveBatches(5)
    .withMaxDoneReceiveBatches(15);

// Create the buffered client
final AmazonSQSAsync bufferedSqs = new AmazonSQSBufferedAsyncClient(sqsAsync, config);
```

QueueBufferConfig Configuration Parameters

Parameter	Default Value	Description
longPoll	true	When `longPoll` is set to `true`, `AmazonSQSBufferedAsyncClient` attempts to use long polling when it consumes messages.
longPollWaitTimeoutSeconds	20 s	The maximum amount of time (in seconds) which a `ReceiveMessage` call blocks off on the server, waiting for messages to appear in the queue before returning with an empty receive result. **Note** When long polling is disabled, this setting has no effect.
maxBatchOpenMs	200 ms	The maximum amount of time (in milliseconds) that an outgoing call waits for other calls with which it batches messages of the same type.

Parameter	Default Value	Description
		The higher the setting, the fewer batches are required to perform the same amount of work (however, the first call in a batch has to spend a longer time waiting). When you set this parameter to 0, submitted requests don't wait for other requests, effectively disabling batching.
maxBatchSize	10 requests per batch	The maximum number of messages that are batched together in a single request. The higher the setting, the fewer batches are required to carry out the same number of requests. **Note** 10 requests per batch is the maximum allowed value for Amazon SQS.
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.
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 limits such as CPU or bandwidth) and the more threads are consumed by `AmazonSQSBufferedAsyncClient`.
`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 limits 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.
`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. Standard queues support a nearly unlimited number of transactions per second (TPS) per action. By default, FIFO queues support up to 3,000 messages per second with batching (p. 174). To request a limit increase, file a support request. FIFO queues support up to 300 messages per second (300 send, receive, or delete operations per second) without batching.

To achieve high throughput, you must scale message producers and consumers horizontally (add more producers and consumers).

Topics

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, you must ensure that the Amazon SQS queue that you use 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.

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 (p. 175).

> **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 2010-2018 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);
```

```java
    }

    /**
     * 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.

Amazon SQS Release Notes

The following table lists Amazon SQS feature releases and improvements. For changes to the *Amazon Simple Queue Service Developer Guide*, see Amazon SQS Document History (p. 193).

Date	Feature Release
November 8, 2018	FIFO (First-In-First-Out) queues are available in the Asia Pacific (Sydney) and Asia Pacific (Tokyo) regions, in addition to the US East (N. Virginia), EU (Ireland), US East (Ohio), and US West (Oregon) regions. For more information about how FIFO queues work and how to get started using them, see Amazon SQS FIFO (First-In-First-Out) Queues (p. 69).
August 28, 2018	Server-side encryption (SSE) for Amazon SQS is available in the AWS GovCloud (US-West) Region, in addition to all commercial regions where Amazon SQS is available, except for the China Regions. For more information about server-side encryption and how to get started using it, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161).
June 28, 2018	You can configure incoming messages to trigger a Lambda function. For more information, see Configuring Messages Arriving in an Amazon SQS Queue to Trigger a Lambda Function (p. 43). **Note** Your queue and Lambda function must be in the same AWS Region. FIFO queues don't support Lambda function triggers. You can associate only one queue with one or more Lambda functions. You can't associate an encrypted queue (p. 161) that uses an AWS managed Customer Master Key for Amazon SQS with a Lambda function in a different AWS account.
May 24, 2018	Server-side encryption (SSE) for Amazon SQS is available in all commercial regions where Amazon SQS is available, except for the China Regions. For more information about server-side encryption and how to get started using it, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161).
March 20, 2018	Amazon CloudWatch Events can use Amazon SQS FIFO queues as targets. For more information, see Automating Notifications from AWS Services to Amazon SQS using CloudWatch Events (p. 134).
January 23, 2018	Amazon S3 Event Notifications are compatible with Amazon SQS SSE. For more information, see the updated Enable Compatibility between AWS Services and Queues with SSE (p. 166) section.
January 2, 2018	The following features of AWS services are compatible with Amazon SQS SSE: • Amazon CloudWatch Events Notifications • Amazon SNS Topic Subscriptions
October 19, 2017	You can track cost allocation by adding, updating, removing, and listing metadata tags for Amazon SQS queues using the `TagQueue`, `UntagQueue`, and `ListQueueTags` actions and the AWS Management Console. For more information, see Amazon SQS Cost Allocation Tags (p. 80) and the Adding, Updating, and Removing Tags from an Amazon SQS Queue (p. 26) tutorial.

Date	Feature Release
September 1, 2017	The complete set of Amazon SQS actions is displayed in the **Actions** list on the **Add a Permission to `MyQueue`** dialog box. For more information, see the Adding Permissions to an Amazon SQS Queue (p. 23) tutorial.
June 14, 2017	FIFO (First-In-First-Out) queues are available in the US East (N. Virginia) region, in addition to the EU (Ireland), US East (Ohio), and US West (Oregon) regions. For more information about how FIFO queues work and how to get started using them, see Amazon SQS FIFO (First-In-First-Out) Queues (p. 69).
June 8, 2017	FIFO (First-In-First-Out) queues are available in the EU (Ireland) region, in addition to the US East (Ohio) and US West (Oregon) regions. For more information about how FIFO queues work and how to get started using them, see Amazon SQS FIFO (First-In-First-Out) Queues (p. 69).
May 23, 2017	Server-side encryption (SSE) for Amazon SQS is available in the US East (N. Virginia) Region, in addition to the US East (Ohio) and US West (Oregon) Regions. For more information about server-side encryption and how to get started using it, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161).
May 19, 2017	• You can use the Amazon SQS Extended Client Library for Java together with the Amazon SQS Java Message Service (JMS) Client. • The Amazon SQS Java Messaging Library has been updated to `1.0.3`. For more information, see Working with JMS and Amazon SQS (p. 92). • Updated the Working with JMS and Amazon SQS (p. 92) section.
May 1, 2017	AWS has expanded its HIPAA compliance program to include Amazon SQS as a HIPAA Eligible Service.

Date	Feature Release
April 28, 2017	Server-side encryption (SSE) for Amazon SQS is available in the US East (Ohio) and US West (Oregon) Regions. SSE lets you protect the contents of messages in Amazon SQS queues using keys managed in the AWS Key Management Service (AWS KMS). For more information about server-side encryption and how to get started using it, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161). For tutorials, see the following: • Creating an Amazon SQS Queue with SSE (p. 19) • Configuring SSE for an Existing Amazon SQS Queue (p. 48) SSE adds the `KmsMasterKeyId` and `KmsDataKeyReusePeriodSeconds` attributes to the `CreateQueue`, `GetQueueAttributes`, and `SetQueueAttributes` actions. **Important** 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 Other features of AWS services or third-party services that send notifications to Amazon SQS aren't compatible with SSE, despite allowing you to set an encrypted queue as a target: • AWS IoT Rule Actions For information about compatibility of other services with encrypted queues, see Enable Compatibility between AWS Services and Queues with SSE (p. 166) and your service documentation.
April 24, 2017	• The Amazon SQS Extended Client Library for Java and Amazon SQS Java Message Service (JMS) Client support FIFO queues. • The Amazon SQS Java Messaging Library has been updated to `1.0.2`. • Updated the Working with JMS and Amazon SQS (p. 92) section.
March 28, 2017	AWS CloudFormation lets your create FIFO queues. Added the AWS CloudFormation (p. 18) tutorial.

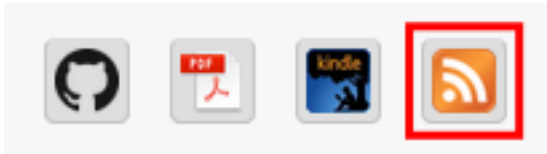| Date	Feature Release
November 17, 2016	*FIFO (First-In-First-Out) queues* or *standard queues* (another name for existing queues) are available in the US West (Oregon) and US East (Ohio) regions. For more information about how FIFO queues work and how to get started using them, see the following: • Amazon SQS FIFO (First-In-First-Out) Queues (p. 69) • Moving from a Standard Queue to a FIFO Queue (p. 72) • Additional Recommendations for Amazon SQS FIFO Queues (p. 118) For revised Amazon SQS tutorials, see the following: • Creating an Amazon SQS Queue (p. 15) • Sending a Message to an Amazon SQS Queue (p. 27) • Receiving and Deleting a Message from an Amazon SQS Queue (p. 37) FIFO queues add the following API functionality: • The `FifoQueue` and `ContentBasedDeduplication` attributes for the `CreateQueue`, `GetQueueAttributes`, and `SetQueueAttributes` actions. • The `MessageDeduplicationId` and `MessageGroupId` request parameters for the `SendMessage` and `SendMessageBatch` actions and attributes for the `ReceiveMessage` action. • The `ReceiveRequestAttemptId` request parameter for the `ReceiveMessage` action. • The `SequenceNumber` response parameter for the `SendMessage` and `SendMessageBatch` actions and the `SequenceNumber` attribute for the `ReceiveMessage` action. **Important** As of November 17, 2016, Amazon SQS no longer publishes a WSDL. The Amazon SQS Buffered Asynchronous Client doesn't currently 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: • Auto Scaling Lifecycle Hooks • AWS IoT Rule Actions • AWS Lambda Dead-Letter Queues For information about compatibility of other services with FIFO queues, see your service documentation. FIFO queues don't support timers on individual messages.
August 31, 2016	The `ApproximateAgeOfOldestMessage` CloudWatch metric lets you find the approximate age of the oldest non-deleted message in the queue. For more information, see Available CloudWatch Metrics for Amazon SQS (p. 127).

Date	Feature Release
February 12, 2016	You can view CloudWatch metrics from the Amazon SQS console for up to 10 of your queues at a time. For more information, see Monitoring Amazon SQS Queues Using CloudWatch (p. 124).
October 27, 2015	The Amazon SQS Extended Client Library for Java lets you manage Amazon SQS messages with Amazon S3. For more information, see Managing Large Amazon SQS Messages Using Amazon S3 (p. 88) in the *Amazon Simple Queue Service Developer Guide*.
December 29, 2014	Amazon SQS lets you use JMS (Java Message Service) with Amazon SQS queues. For more information, see Working with JMS and Amazon SQS (p. 92) in the *Amazon Simple Queue Service Developer Guide*.
December 8, 2014	Amazon SQS lets you delete the messages in a queue using the `PurgeQueue` action. For more information, see PurgeQueue in the *Amazon Simple Queue Service API Reference*.
July 16, 2014	Amazon SQS lets you log actions using AWS CloudTrail. For more information, see Logging Amazon SQS API Calls Using AWS CloudTrail (p. 130).
May 6, 2014	Amazon SQS provides support for message attributes. For more information, see Amazon SQS Message Attributes (p. 77).
January 29, 2014	Amazon SQS provides support for dead-letter queues. For more information, see Amazon SQS Dead-Letter Queues (p. 82).
November 21, 2012	You can subscribe an Amazon SQS queue to an Amazon SNS topic using the Amazon SQS console. For more information, see Subscribing an Amazon SQS Queue to an Amazon SNS Topic (p. 41).
November 5, 2012	The 2012-11-05 API version of Amazon SQS adds support for Signature Version 4, which provides improved security and performance. For more information about Signature Version 4, see Basic Authentication Process with HMAC-SHA (p. 171).
November 5, 2012	The AWS SDK for Java includes a buffered asynchronous client, `AmazonSQSBufferedAsyncClient`, for accessing Amazon SQS. This client allows for easier request batching by enabling client-side buffering, where calls made from the client are first buffered and then sent as a batch request to Amazon SQS. For more information about client-side buffering and request batching, see Enabling Client-Side Buffering and Request Batching (p. 175).
November 5, 2012	The 2012-11-05 API version of Amazon SQS adds long polling support. Long polling allows Amazon SQS to wait for a specified amount time for a message to be available instead of returning an empty response if one isn't available. For more information about long polling, see Amazon SQS Long Polling (p. 81).

Amazon SQS Document History

The following table lists changes to the *Amazon Simple Queue Service Developer Guide*. For Amazon SQS feature releases and improvements, see Amazon SQS Release Notes (p. 189).

Date	Documentation Update
October 8, 2018	Updated the New and Frequently Viewed Amazon SQS Topics (p. 4) section.

Date	Documentation Update
October 3, 2018	Corrected the example IP address in the Give Permission to Requests from Amazon EC2 Instances (p. 156) section.
September 25, 2018	Corrected the information in the following sections: • SetQueueAttributes (p. 133) • Receiving and Deleting a Message from an Amazon SQS Queue (p. 37)
September 10, 2018	Updated the New and Frequently Viewed Amazon SQS Topics (p. 4) section.
September 4, 2018	• Corrected the information in the following sections: • Enable Compatibility between AWS Services and Queues with SSE (p. 166) • How Do Dead-Letter Queues Work? (p. 82) • Clarified the preamble in the Subscribing an Amazon SQS Queue to an Amazon SNS Topic (p. 41) section. • Further revised the following statement regarding the visibility timeout (p. 85) throughout this guide: The default visibility timeout for a message is 30 seconds. The minimum is 0 seconds. The maximum is 12 hours. • Rewrote the Inflight Messages (p. 86) section.
August 28, 2018	Updated the Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161) section.
August 27, 2018	Corrected the information in the Actions and Resource Reference (p. 159) section.
August 22, 2018	Rewrote the Logging Amazon SQS API Calls Using AWS CloudTrail (p. 130) section.
August 21, 2018	Revised the attribute table in the Resources Required to Process Amazon SQS Messages (p. 80) section.
August 20, 2018	Added the following note to the Amazon SQS Dead-Letter Queues (p. 82) and Configuring an Amazon SQS Dead-Letter Queue (p. 53) section: When you designate a queue to be a source queue, a dead-letter queue is *not* created automatically. You must first create a normal standard or FIFO queue before designating it a dead-letter queue.
August 15, 2018	Added the following note to the Amazon SQS Long Polling (p. 81) section: You can confirm that a queue is empty when you perform a long poll and the `ApproximateNumberOfMessagesDelayed`, `ApproximateNumberOfMessagesNotVisible`, and `ApproximateNumberOfMessagesVisible` metrics are equal to 0 at least 1 minute after the producers stop sending messages (when the queue metadata reaches eventual consistency). For more information, see Available CloudWatch Metrics for Amazon SQS (p. 127).
August 7, 2018	Clarified the information in the Inflight Messages (p. 86) section.
August 3, 2018	Updated the New and Frequently Viewed Amazon SQS Topics (p. 4) section.
July 25, 2018	• Rewrote the Creating CloudWatch Alarms for Amazon SQS Metrics (p. 126) section and removed outdated screenshots. • Renamed the Accessing CloudWatch Metrics for Amazon SQS (p. 124) section.

Date	Documentation Update
July 6, 2018	Revised the following statements throughout this guide: • Visibility Timeout (p. 85): The default visibility timeout for a message is 30 seconds. The minimum is 0 seconds. The maximum is 12 hours. • Message Timers (p. 88): The default (minimum) invisibility period for a message is 0 seconds. The maximum is 15 minutes. • Delay Queues (p. 87): The default (minimum) delay for a queue is 0 seconds. The maximum is 15 minutes.
July 5, 2018	• Updated the New and Frequently Viewed Amazon SQS Topics (p. 4) section. • Added the following note regarding the `ApproximateAgeOfOldestMessage` metric to the Available CloudWatch Metrics for Amazon SQS (p. 127) section: For dead-letter queues (p. 82), the value of `ApproximateAgeOfOldestMessage` is the longest time that a message has been in the queue.
July 3, 2018	• Corrected the links for filing support requests throughout this guide. • Corrected some minor errors in the Amazon SQS Tutorials (p. 15) section.
June 28, 2018	Created the Configuring Messages Arriving in an Amazon SQS Queue to Trigger a Lambda Function (p. 43) tutorial.
June 26, 2018	Corrected the information in the How Do Dead-Letter Queues Work? (p. 82) section.
June 11, 2018	• Added the following information to the Limits Related to Messages (p. 122) section: A batched message ID can have up to 80 characters. The following characters are accepted: alphanumeric characters, hyphens (–), and underscores (_). • Clarified and corrected the information in the Using AmazonSQSBufferedAsyncClient (p. 175) section.
June 8, 2018	Updated the New and Frequently Viewed Amazon SQS Topics (p. 4) section.
June 5, 2018	In addition to GitHub, HTML, PDF, and Kindle, the *Amazon MQ Developer Guide* release notes are available as an RSS feed.
May 29, 2018	• Improved the Java example in the following sections: • SQSExtendedClientExample.java (p. 89) • SimpleProducerConsumer.java (p. 180)
May 24, 2018	Updated the Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161) section.

Date	Documentation Update
May 22, 2018	• Corrected the information in all Java dependency sections. • Revised the following statement: For most standard queues (depending on queue traffic and message backlog), there can be a maximum of approximately 120,000 inflight messages (received from a queue by a consumer, but not yet deleted from the queue). If you reach this limit, Amazon SQS returns the `OverLimit` error message. To avoid reaching the limit, 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 limit increase, file a support request.
May 15, 2018	Clarified the information in the Processing Messages in a Timely Manner (p. 115) section.
May 10, 2018	• Created the Sending a Message with a Timer to an Amazon SQS Queue (p. 35) tutorial. • Created the To send a message with a timer to a queue (p. 36) section with example Java code. • Grouped related tutorials into sections: • Creating Amazon SQS Queues (p. 15) • Sending Messages to Amazon SQS Queues (p. 27) • Configuring Amazon SQS Queues (p. 47)
May 9, 2018	Rewrote the Amazon SQS Message Timers (p. 88) section.
May 8, 2018	• Rewrote the Amazon SQS Delay Queues (p. 87) section. • Created the Configuring an Amazon SQS Delay Queue (p. 60) tutorial. • Created the following sections with example Java code: • To configure a delay queue (p. 61) • To configure a delay queue and send, receive, and delete messages (p. 63)
May 4, 2018	Updated the New and Frequently Viewed Amazon SQS Topics (p. 4) section.
May 3, 2018	• Created the following sections with example Java code: • To configure visibility timeout for a queue (p. 58) • To configure visibility timeout for a single message or multiple messages and send, receive, and delete messages (p. 60) • Improved the example Java code in the following sections: • SQSSimpleJavaClientExample.java (p. 67) • SQSFIFOJavaClientExample.java (p. 73) • SQSLongPollingExample.java (p. 52) • SQSDeadLetterQueueExample.java (p. 55)
May 2, 2018	• Corrected and clarified the information in the Amazon SQS Visibility Timeout (p. 85) section. • Created the Configuring Visibility Timeout for an Amazon SQS Queue (p. 57) tutorial.
May 1, 2018	Corrected and clarified the information in the Limits Related to Queues (p. 121) section.

Date	Documentation Update
April 25, 2018	• Corrected and clarified the information in the following sections: • Configuring Long Polling for an Amazon SQS queue (p. 51) • Identifiers for Amazon SQS Standard and FIFO Queues (p. 76) • Getting Started with Amazon SQS (p. 8) • Added the following note throughout this guide: Cross-account permissions don't apply to the following actions: • `AddPermission` • `CreateQueue` • `DeleteQueue` • `ListQueues` • `ListQueueTags` • `RemovePermission` • `SetQueueAttributes` • `TagQueue` • `UntagQueue` • Corrected a minor error in the Java example in the SQSLongPollingExample.java (p. 52) section.
April 24, 2018	• Rewrote the Amazon SQS Long Polling (p. 81) section. • Created the Configuring Long Polling for an Amazon SQS queue (p. 51) tutorial. • Created the Setting Up Long Polling (p. 116) section.
April 23, 2018	Rewrote the Processing Messages in a Timely Manner (p. 115) section.
April 11, 2018	• Removed outdated information from the following sections and added cross-references to related tutorials: • Amazon SQS Standard Queues (p. 66) • Amazon SQS FIFO (First-In-First-Out) Queues (p. 69) • Amazon SQS Dead-Letter Queues (p. 82) • Amazon SQS Cost Allocation Tags (p. 80) • Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161) • Restructured the following sections: • Managing Large Amazon SQS Messages Using Amazon S3 (p. 88) • Making Query API Requests (p. 169) • Amazon SQS Batch Actions (p. 174)

Date	Documentation Update
April 10, 2018	• Created the To configure a dead-letter queue (p. 55) section with example Java code. • Restructured the following sections: • Amazon SQS FIFO (First-In-First-Out) Queues (p. 69) • Working with Amazon SQS Messages (p. 115) • Reducing Amazon SQS Costs (p. 117) • Additional Recommendations for Amazon SQS FIFO Queues (p. 118) • Rewrote the following sections: • Message Ordering (p. 70) • To configure a dead-letter queue and send, receive, and delete a message (p. 57)
April 9, 2018	• Rewrote the following sections: • Differences Between Long and Short Polling (p. 82) • Amazon SQS Delay Queues (p. 87) • Moved the Query API examples for the following concepts to the *Amazon Simple Queue Service API Reference*: • Amazon SQS Dead-Letter Queues (p. 82) • Amazon SQS Long Polling (p. 81) • Amazon SQS Delay Queues (p. 87) • Amazon SQS Message Timers (p. 88)
April 6, 2018	• Rewrote the following sections: • Amazon SQS Access Policy Language Evaluation Logic (p. 152) • Relationships Between Explicit and Default Denials in the Amazon SQS Access Policy Language (p. 154) • Using Temporary Security Credentials (p. 157) • Restructured the following sections: • Using Identity-Based (IAM) Policies for Amazon SQS (p. 141) • Using Custom Policies with the Amazon SQS Access Policy Language (p. 149) • Updated the New and Frequently Viewed Amazon SQS Topics (p. 4) section.
April 5, 2018	• Added Java examples (p. 33) to the Sending a Message with Attributes to an Amazon SQS Queue (p. 30) tutorial. • Rewrote the Calculating the MD5 Message Digest for Message Attributes (p. 79) section. • Corrected the instructions for the Java example (p. 26) in the Adding, Updating, and Removing Tags from an Amazon SQS Queue (p. 26) tutorial.
April 4, 2018	• Created the Sending a Message with Attributes to an Amazon SQS Queue (p. 30) tutorial. • Rewrote the following sections: • Message Attribute Components (p. 78) • Message Attribute Data Types (p. 78) • Next Steps (p. 13) (Getting Started)
March 29, 2018	Added the following note to the Calculating the MD5 Message Digest for Message Attributes (p. 79) section: Always include custom data type suffixes in the MD5 message-digest calculation.

Date	Documentation Update
March 27, 2018	• Create the How Can I Get Started with Amazon SQS? (p. 2) section. • Rewrote and moved the Message Lifecycle (p. 65) section. • Restructured the following sections for better readability: • Best Practices for Amazon SQS (p. 115) • How Amazon SQS Works (p. 64)
March 26, 2018	• Added Implementing Request-Response Systems (p. 117) to the Best Practices for Amazon SQS (p. 115) section. • Rewrote the What Are the Main Benefits of Amazon SQS? (p. 1) section to include information about recently released features. • Refactored the contents of the Amazon SQS Prerequisites section into existing sections and removed the section. • Added explanatory notes to the Deleting an Amazon SQS Queue (p. 46) section. • Clarified the introduction to the Managing Large Amazon SQS Messages Using Amazon S3 (p. 88) section.
March 23, 2018	• Restructured and rewrote the What Is Amazon Simple Queue Service? (p. 1) section. • Added the How is Amazon SQS Different from Amazon MQ or Amazon SNS? (p. 1) section. • Moved the Basic Amazon SQS Architecture (p. 64) section.
March 22, 2018	• Renamed the Monitoring and Logging section to Amazon SQS Monitoring, Logging, and Automating (p. 124). • Created the Automating Notifications from AWS Services to Amazon SQS using CloudWatch Events (p. 134) section. • Clarified the titles of the examples in the What AWS KMS Permissions Do I Need to Use SSE for Amazon SQS? (p. 165) section.
March 20, 2018	• Clarified the following statement throughout this guide: 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 limit 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 limit to the number of inflight messages. • Removed unnecessary screenshots and improved the explanation of diagrams in the Amazon SQS Tutorials (p. 15), How Amazon SQS Works (p. 64), and Amazon SQS Security (p. 135) sections. • Added a preamble to the Amazon SQS Queue and Message Identifiers (p. 76) section and clarified to which queue types various Amazon SQS identifiers correspond. • Corrected and clarified information and fixed the syntax highlighting in the Logging Amazon SQS API Calls Using AWS CloudTrail (p. 130) and Amazon SQS Security (p. 135) sections.

Date	Documentation Update
March 19, 2018	Clarified in the Limits Related to Policies (p. 123) and Adding Permissions to an Amazon SQS Queue (p. 23) sections that an Amazon SQS policy can have a maximum of 7 actions.
March 14, 2018	• Updated and optimized the screenshots throughout this guide for better usability. • Improved the navigation in all the tutorials.
March 13, 2018	Clarified the batched and unbatched throughput for FIFO queues throughout this guide.
March 7, 2018	Updated the New and Frequently Viewed Amazon SQS Topics (p. 4) section.
March 2, 2018	• Added the `sqs:ListQueueTags`, `sqs:TagQueue`, and `sqs:UntagQueue` permissions to the Actions and Resource Reference (p. 159) section. • Clarified the permissions for the following API actions: • `ChangeMessageVisibilityBatch` (`sqs:ChangeMessageVisibility`) • `DeleteMessageBatch` (`sqs:DeleteMessage`) • `SendMessageBatch` (`sqs:SendMessage`)
February 28, 2018	Corrected image display in GitHub.
February 27, 2018	In addition to HTML, PDF, and Kindle, the *Amazon Simple Queue Service Developer Guide* is available on GitHub. To leave feedback, choose the GitHub icon in the upper right-hand corner.
February 26, 2018	• Made regions consistent in all examples. • Optimized links to the AWS console and product webpages.
February 23, 2018	• Rewrote the Making Query API Requests (p. 169) section. • Corrected the Java code samples in the Enabling Client-Side Buffering and Request Batching (p. 175) section. • Replaced `http://` with `https://` in all examples of Amazon SQS endpoints.
February 21, 2018	• Corrected the Java code sample in the Using AmazonSQSBufferedAsyncClient (p. 175) section. • Clarified the information in the Purging Messages from an Amazon SQS Queue (p. 45) section.

Date	Documentation Update

February 20, 2018	• Clarified the information in the Moving from a Standard Queue to a FIFO Queue (p. 72) section.
	• Optimized the Java code sample in the following sections:
	  • AWS SDK for Java (p. 17)
	  • Create an Amazon SQS Queue with SSE using the AWS SDK for Java (p. 21)
	  • Configure SSE for an Amazon SQS Queue Using the AWS SDK for Java (p. 49)
	  • AWS SDK for Java (p. 23)
	  • AWS SDK for Java (p. 29)
	  • AWS SDK for Java (p. 39)
	  • AWS SDK for Java (p. 55)
	  • AWS SDK for Java (p. 26)

February 19, 2018	Optimized the example Java code and corrected `pom.xml` prerequisites in the following sections: • Working Java Example for Using Amazon S3 for Large Amazon SQS Messages (p. 89) • Working Java Example for Standard Queues (p. 67) • Working Java Example for FIFO Queues (p. 73)
February 16, 2018	Simplified the example Java code and added `pom.xml` prerequisites to the following sections: • Working Java Example for Standard Queues (p. 67) • Working Java Example for FIFO Queues (p. 73)
February 15, 2018	Updated the Related Amazon SQS Resources (p. 188) section.
February 14, 2018	• Clarified the information in the Consuming Messages Using Short Polling (p. 66) section. • Restructured the Amazon SQS Limits (p. 121) section.
February 13, 2018	• Clarified the following statement: You can't add tags to a new queue when you create it using the AWS Management Console (you *can* add tags after the queue is created). However, you can add, update, or remove queue tags at any time using the Amazon SQS actions. • Clarified and corrected the information in the Setting the Visibility Timeout (p. 86) and Processing Messages in a Timely Manner (p. 115) sections. • Updated the Related Amazon SQS Resources (p. 188) section. • Added the We Want to Hear from You (p. 3) section.
February 9, 2018	• Made the Java example in the Working Java Example for Single-Operation and Batch Requests (p. 180) section self-contained and added `pom.xml` prerequisites. • Added an explanation for monitoring the example run (p. 187).
February 8, 2018	Rewrote the Java example in the Working Java Example for Single-Operation and Batch Requests (p. 180) section.
February 7, 2018	Rewrote the following sections: • Increasing Throughput using Horizontal Scaling and Action Batching (p. 178)

Date	Documentation Update
February 6, 2018	Rewrote the following sections: • Amazon SQS Batch Actions (p. 174) • Enabling Client-Side Buffering and Request Batching (p. 175)
February 5, 2018	Clarified the information in the Configuring an Amazon SQS Dead-Letter Queue (p. 53) section.
February 2, 2018	Created the New and Frequently Viewed Amazon SQS Topics (p. 4) section.
February 1, 2018	• Clarified the information in the FIFO Delivery Logic (p. 71) section. • Corrected the code examples in—and renamed—the following sections: • Working Java Example for Standard Queues (p. 67) • Working Java Example for FIFO Queues (p. 73) • Working Java Example for Using Amazon S3 for Large Amazon SQS Messages (p. 89) • Working Java Example for Using JMS with Amazon SQS Standard Queues (p. 99)
January 31, 2018	Clarified the information in the following sections: • Allow Developers to Write Messages to a Shared Queue (p. 144) • Amazon SQS Batch Actions (p. 174) • How Do Dead-Letter Queues Work? (p. 82)
January 30, 2018	Rewrote the following sections: • Basic Authentication Process with HMAC-SHA (p. 171) • Interpreting Responses (p. 173)
January 29, 2018	Rewrote the following sections: • Making Query API Requests (p. 169) • Constructing an Endpoint (p. 169) • Making a GET Request (p. 170) • Authenticating Requests (p. 171)
January 25, 2018	• Added links to explain why client constructors are deprecated in the AWS SDK for Java throughout this guide. For more information, see Creating Service Clients in the *AWS SDK for Java Developer Guide*. • Clarified the following statement in the Monitoring Amazon SQS Queues Using CloudWatch (p. 124) section: Detailed monitoring (or one-minute metrics) is currently unavailable for Amazon SQS. Making requests to CloudWatch at this resolution might return no data.
January 24, 2018	Clarified the wording for Amazon SQS actions throughout this guide.
January 22, 2018	Added the Enable Compatibility between AWS Services and Queues with SSE (p. 166) section.
January 19, 2018	Clarified the information in the How Do Dead-Letter Queues Work? (p. 82) section.

Date	Documentation Update
| January 18, 2018 | • Rewrote the code in the the section called "Creating a JMS Connection" (p. 93) section, replacing the deprecated `AmazonSQSClient` constructor with `AmazonSQSClientBuilder`. Use the following syntax to create a new connection factory with all defaults (such as credentials and region) set:<br><br>```java
final SQSConnectionFactory connectionFactory = new
 SQSConnectionFactory(
        new ProviderConfiguration(),
        AmazonSQSClientBuilder.defaultClient());
```<br><br>• Rewrote the code in the Horizontal Scaling (p. 179) section. Use the following syntax to adjust the maximum number of allowable producer and consumer threads on an `AmazonSQSClientBuilder` object:<br><br>```java
final AmazonSQS sqsClient = AmazonSQSClientBuilder.standard()
        .withClientConfiguration(new ClientConfiguration()
                .withMaxConnections(producerCount + consumerCount))
        .build();
``` |
| January 17, 2018 | • Rewrote and simplified the code in the Working Java Example for Standard Queues (p. 67) and Working Java Example for FIFO Queues (p. 73) sections, replacing the deprecated `AmazonSQSClient` constructor with `AmazonSQSClientBuilder`. Use the following syntax to create a new instance of the builder with all defaults (such as credentials and region) set:<br><br>```java
final AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
```<br><br>• Rewrote and simplified the code in the Working Java Example for Using Amazon S3 for Large Amazon SQS Messages (p. 89) section, replacing the deprecated `AmazonS3Client` constructor with `AmazonS3ClientBuilder`:<br><br>```java
final AmazonSQS s3 = AmazonS3ClientBuilder.defaultClient();
``` |
| January 16, 2018 | • Added more cross-references to Message ID (p. 77) and Receipt Handle (p. 77) throughout this guide.
• Rewrote the Managing Large Amazon SQS Messages Using Amazon S3 (p. 88) section. |

Date	Documentation Update
January 15, 2018	• In the Managing Large Amazon SQS Messages Using Amazon S3 (p. 88) and Working with JMS and Amazon SQS (p. 92) sections, clarified the following explanation: The SDK for Java and Amazon SQS Extended Client Library for Java require the J2SE Development Kit 8.0 or later. • Added sub-sections to the Best Practices for Amazon SQS (p. 115) section and clarified and reorganized the content. • Added the following explanation to the Setting Up Dead-Letter Queue Retention (p. 117) section: The expiration of a message is always based on its original enqueue timestamp. When a message is moved to a dead-letter queue (p. 82), the enqueue timestamp remains unchanged. For example, if a message spends 1 day in the original queue before being moved to a dead-letter queue, and the retention period of the dead-letter queue is set to 4 days, the message is deleted from the dead-letter queue after 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. • Clarified the information in the How Do Dead-Letter Queues Work? (p. 82) section. • Added the following explanation to the Amazon SQS Visibility Timeout (p. 85) section: The default visibility timeout for a message is 30 seconds. The minimum is 0 seconds. The maximum is 12 hours.
January 3, 2018	Further clarified the throughput for FIFO queues throughout this guide.
December 7, 2017	• Added the following note to the Using Identity-Based (IAM) Policies for Amazon SQS (p. 141) section: With the exception of `ListQueues`, all Amazon SQS actions support resource-level permissions. For more information, see Actions and Resource Reference (p. 159). • Added the "Introducing Amazon Simple Queue Service (SQS) FIFO Queues" video to the Amazon SQS FIFO (First-In-First-Out) Queues (p. 69) section. • Added the "Introducing Amazon Simple Queue Service (SQS) Server-Side Encryption" video to the Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161) section. • Added the "Introducing Cost Allocation Tags for Amazon Simple Queue Service (SQS)" video to the Amazon SQS Cost Allocation Tags (p. 80) section.

Date	Documentation Update
December 6, 2017	• Suggested using Step Functions instead of Amazon SWF for visibility timeouts longer than 12 hours in the Best Practices for Amazon SQS (p. 115) section. • Clarified the following explanation in the Limits Related to Messages (p. 122) section: The default visibility timeout for a message is 30 seconds. The minimum is 0 seconds. The maximum is 12 hours. • Added the following note to the Subscribing an Amazon SQS Queue to an Amazon SNS Topic (p. 41) section: If your Amazon SQS queue and Amazon SNS topic are in different AWS accounts, the owner of the topic must first confirm the subscription. For more information, see Confirm the Subscription in the *Amazon Simple Notification Service Developer Guide*. • Added the following note to the Key Terms (p. 70) section: Amazon SQS continues to keep track of the message deduplication ID even after the message is received and deleted. • Clarified and reorganized the information in the Making Query API Requests (p. 169) and Monitoring Amazon SQS Queues Using CloudWatch (p. 124) sections.
December 1, 2017	Clarified and reorganized the information in the Monitoring Amazon SQS Queues Using CloudWatch (p. 124) section.
October 30, 2017	• Corrected and reorganized the table of contents. • Rewrote the Amazon SQS Visibility Timeout (p. 85) section.
October 27, 2017	Clarified the explanation of throughput for FIFO queues in the Amazon SQS FIFO (First-In-First-Out) Queues (p. 69) section.
September 29, 2017	Added a note about the Amazon SQS Buffered Asynchronous Client to the Increasing Throughput using Horizontal Scaling and Action Batching (p. 178) section.
September 19, 2017	Corrected the diagrams in the Using Amazon SQS and IAM Policies (p. 142) section.
August 29, 2017	Clarified the information in the Changing the Visibility Timeout for a Message (p. 87) section.
August 17, 2017	Clarified the permissions for the `SendMessage` and `SendMessageBatch` actions in Actions and Resource Reference (p. 159).
August 15, 2017	Updated information about dead-letter queues in the Recommendations for Amazon SQS Standard and FIFO (First-In-First-Out) Queues (p. 115) section.
August 9, 2017	• The Amazon SQS Java Messaging Library has been updated to `1.0.4`. For more information, see Working with JMS and Amazon SQS (p. 92). • Updated the Working with JMS and Amazon SQS (p. 92) section.
July 27, 2017	Changed the deprecated `AmazonSQSClient` constructor to `AmazonSQSClientBuilder` and revised the corresponding region specification in the Working Java Example for Standard Queues (p. 67) section.
July 25, 2017	Clarified the throughput for standard and FIFO queues throughout this guide.

Date	Documentation Update
July 20, 2017	Clarified the compatibility between Amazon SQS SSE queues and AWS and third-party service features throughout this guide: 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 Other features of AWS services or third-party services that send notifications to Amazon SQS aren't compatible with SSE, despite allowing you to set an encrypted queue as a target: • AWS IoT Rule Actions For information about compatibility of other services with encrypted queues, see Enable Compatibility between AWS Services and Queues with SSE (p. 166) and your service documentation.
June 23, 2017	Corrected the information in the Limits Related to Messages (p. 122) section.
June 20, 2017	Clarified the information in the Amazon SQS Dead-Letter Queues (p. 82) section.
June 2, 2017	• Restructured and updated the Amazon SQS Dead-Letter Queues (p. 82) section. • Created the Configuring an Amazon SQS Dead-Letter Queue (p. 53) section.
June 1, 2017	Updated the What Is Amazon Simple Queue Service? (p. 1) section.
May 24, 2017	• Restructured the Working with JMS and Amazon SQS (p. 92) section. • Created the Using the JMS Client with Other Amazon SQS Clients (p. 98) section.
May 23, 2017	Server-side encryption (SSE) for Amazon SQS is available in the US East (N. Virginia) Region, in addition to the US East (Ohio) and US West (Oregon) Regions. For more information about server-side encryption and how to get started using it, see Protecting Data Using Server-Side Encryption (SSE) and AWS KMS (p. 161).
May 19, 2017	• You can use the Amazon SQS Extended Client Library for Java together with the Amazon SQS Java Message Service (JMS) Client. • The Amazon SQS Java Messaging Library has been updated to `1.0.3`. For more information, see Working with JMS and Amazon SQS (p. 92). • Updated the Working with JMS and Amazon SQS (p. 92) section.
April 25, 2017	Restructured and updated the Amazon SQS Long Polling (p. 81) section.
February 6, 2017	Updated the Authentication and Access Control (p. 135) section.
December 16, 2016	Retired the *Amazon Simple Queue Service Getting Started Guide* and incorporated some of its content into the following sections of this guide: • Setting Up Amazon SQS (p. 5) • Getting Started with Amazon SQS (p. 8) • Amazon SQS Tutorials (p. 15)

Date	Documentation Update
December 2, 2016	Restructured and updated the Authentication and Access Control (p. 135) section.
November 2, 2016	Renamed the Walkthroughs section to Amazon SQS Tutorials (p. 15).
May 27, 2016	Added the Best Practices for Amazon SQS (p. 115) section.
May 12, 2016	Added the Amazon SQS Limits (p. 121) section.
December 7, 2015	Updated Amazon SQS console screenshots.
August 4, 2014	Updated information about access keys. For more information, see Authenticating Requests (p. 171).

AWS Glossary

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