Amazon Simple Queue Service

Developer Guide

API Version 2011-10-01

Amazon Web Services

Amazon Simple Queue Service: Developer Guide

Amazon Web Services

Abstract

A concise overview of Amazon Simple Queue Service (Amazon SQS) concepts and programming information.

What is Amazon Simple Queue Service?

Amazon SQS offers reliable and scalable hosted queues for storing messages as they travel between computers. By using Amazon SQS, you can move data between distributed components of your applications that perform different tasks without losing messages or requiring each component to be always available.

Amazon SQS is a distributed queue system that enables web service applications to quickly and reliably queue messages that one component in the application generates to be consumed by another component. A queue is a temporary repository for messages that are awaiting processing.

Using Amazon SQS, you can decouple the components of an application so they run independently, with Amazon SQS easing message management between components. Any component of a distributed application can store messages in a fail-safe queue. Messages can contain up to 64 KiB of text in any format. Any component can later retrieve the messages programmatically using the SQS API.

The queue acts as a buffer between the component producing and saving data, and the component receiving the data for processing. This means the queue resolves issues that arise if the producer is producing work faster than the consumer can process it, or if the producer or consumer are only intermittently connected to the network.

SQS ensures delivery of each message at least once, and supports multiple readers and writers interacting with the same queue. A single queue can be used simultaneously by many distributed application components, with no need for those components to coordinate with each other to share the queue.

Amazon SQS is engineered to always be available and deliver messages. One of the resulting tradeoffs is that SQS does not guarantee first in, first out delivery of messages. For many distributed applications, each message can stand on its own, and as long as all messages are delivered, the order is not important. If your system requires that order be preserved, you can place sequencing information in each message, so that you can reorder the messages when the queue returns them.

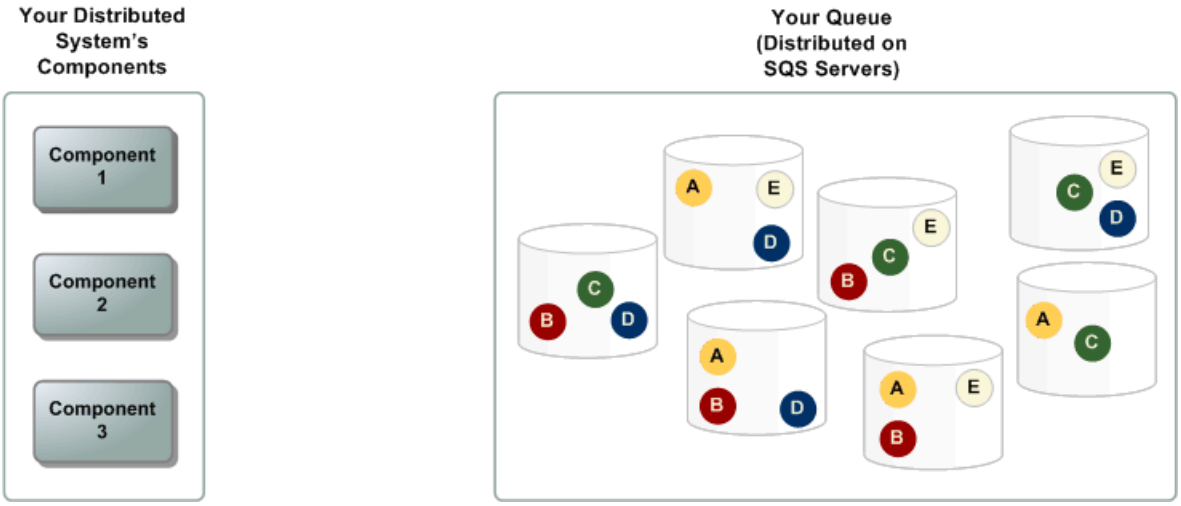
Be sure to read about distributed queues, which will help you understand how to design an application that works correctly with Amazon SQS. For more information, see Properties of Distributed Queues (p. 4).

Architectural Overview

There are three main actors in the overall system:

- The components of your distributed system
- Queues
- Messages in the queues

In the following diagram, your system has several components that send messages to the queue and receive messages from the queue. The diagram shows that a single queue, which has its messages (labeled A-E), is redundantly saved across multiple SQS servers.



Amazon SQS Features

Amazon SQS provides the following major features:

- **Redundant infrastructure—**Guarantees delivery of your messages at least once, highly concurrent access to messages, and high availability for sending and retrieving messages
- **Multiple writers and readers—**Multiple parts of your system can send or receive messages at the same time
  SQS locks the message during processing, keeping other parts of your system from processing the message simultaneously.
- **Configurable settings per queue—**All of your queues don't have to be exactly alike
  For example, one queue can be optimized for messages that require a longer processing time than others.
- **Variable message size—**Your messages can be up to 65536 bytes (64 KiB) in size
  For even larger messages, you can store the contents of the message using the Amazon Simple Storage Service (Amazon S3) or Amazon SimpleDB and use Amazon SQS to hold a pointer to the Amazon S3 or Amazon SDB object. Alternately, you can split the larger message into smaller ones.
  For more information about the services, go to the Amazon S3 detail page and the Amazon SimpleDB detail page.
- **Access control—**You can control who can send messages to a queue, and who can receive messages from a queue
- **Delay Queues—**A delay queue is one which the user sets a default delay on a queue such that delivery of all messages enqueued will be postponed for that duration of time. You can set the delay value when you create a queue with `CreateQueue`, and you can update the value with `SetQueueAttributes`. If you update the value, the new value affects only messages enqueued after the update.

How SQS Queues Work

Topics

This section describes the basic properties of Amazon SQS queues, identifiers for queues and messages, how you determine the general size of the queue, and how you manage the messages in a queue.

A queue can be empty if you haven't sent any messages to it or if you have deleted all the messages from it.

You must assign a name to each of your queues (for more information, see Queue URLs (p. 5)). You can get a list of all your queues or a subset of your queues that share the same initial characters in their names (for example, you could get a list of all your queues whose names start with "T3").

You can delete a queue at any time, whether it is empty or not. Be aware, however, that queues retain messages for a set period of time. By default, a queue retains messages for four days. However, you can configure a queue to retain messages for up to 14 days after the message has been sent.

Amazon SQS can delete your queue without notification if one of the following actions hasn't been performed on it for 30 consecutive days: `SendMessage`, `ReceiveMessage`, `DeleteMessage`, `GetQueueAttributes`, `SetQueueAttributes`, `AddPermission`, and `RemovePermission`.

> **Important**
>
> It is a violation of the intended use of Amazon SQS if you repeatedly create queues and then leave them inactive, or if you store excessive amounts of data in your queue.

The following table lists the API actions to use.

To do this...	Use this action
Create a queue	CreateQueue
Get the URL of an existing queue	GetQueueUrl

To do this...	Use this action
List your queues	ListQueues
Delete a queue	DeleteQueue

Properties of Distributed Queues

The following information can help you design your application to work with Amazon SQS correctly.

Message Order

SQS makes a best effort to preserve order in messages, but due to the distributed nature of the queue, we cannot guarantee you will receive messages in the exact order you sent them. If your system requires that order be preserved, we recommend you place sequencing information in each message so you can reorder the messages upon receipt.

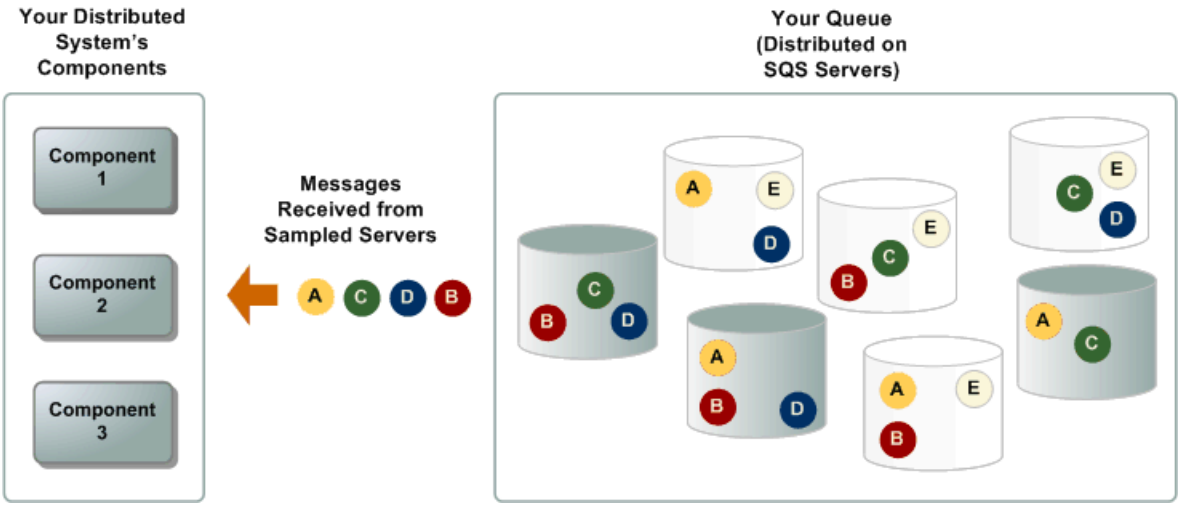
At-Least-Once Delivery

SQS stores copies of your messages on multiple servers for redundancy and high availability. On rare occasions, one of the servers storing a copy of a message might be unavailable when you receive or delete the message. If that occurs, the copy of the message will not be deleted on that unavailable server, and you might get that message copy again when you receive messages. Because of this, you must design your application to be idempotent (i.e., it must not be adversely affected if it processes the same message more than once).

Message Sample

When you retrieve messages from the queue, SQS samples a subset of the servers (based on a weighted random distribution) and returns messages from just those servers. This means that a particular receive request might not return all your messages. Or, if you have a small number of messages in your queue (less than 1000), it means a particular request might not return any of your messages, whereas a subsequent request will. If you keep retrieving from your queues, SQS will sample all of the servers, and you will receive all of your messages.

The following figure shows messages being returned after one of your system components makes a receive request. SQS samples several of the servers (in gray) and returns the messages from those servers (Message A, C, D, and B). Message E is not returned to this particular request, but it would be returned to a subsequent request.



Queue and Message Identifiers

SQS uses the following three identifiers that you need to be familiar with:

- Queue URL
- Message ID
- Receipt handle

Queue URLs

When creating a new queue, you must provide a queue name that is unique within the scope of all your queues. If you create queues using both the latest WSDL and a previous version, you still have a single namespace for all your queues. SQS assigns each queue you create an identifier called a queue URL, which includes the queue name and other components that SQS determines. Whenever you want to perform an action on a queue, you provide its queue URL.

The following is the queue URL for a queue named "queue2" owned by a person with the AWS account number "123456789012".

```
http://sqs.us-east-1.amazonaws.com/123456789012/queue2
```

Important

In your system, always store the entire queue URL as Amazon SQS returned it to you when you created the queue (for example, `http://sqs.us-east-1.amazonaws.com/123456789012/queue2`). 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 could change the components that make up the queue URL.

You can also get the queue URL for a queue by listing your queues. Even though you have a single namespace for all your queues, the list of queues returned depends on the WSDL you use for the request. For more information, see ListQueues.

Message IDs

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, but to delete a message, you need the message's receipt handle instead of the message ID. The maximum length of a message ID is 100 characters.

Receipt Handles

Each time you receive a message from a queue, you receive a *receipt handle* for that message. The handle is associated with the act 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 and not the message ID. This means 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 1024 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 or the message might not be deleted.

Following is an example of a receipt handle.

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

Resources Required to Process Messages

To help you estimate the resources needed to process your queued messages, Amazon SQS can provide you with an approximate number of messages in a queue. You can view the number of messages that are visible or you can view the number of messages that are not visible. Messages that are not visible are messages in a queue that are not timed-out and not deleted. For more information about visibility, see Visibility Timeout (p. 7).

> **Important**
>
> Because of the distributed architecture of SQS, the result is not an exact count of the number of messages in a queue. In most cases it should be close to the actual number of messages in the queue, but you should not rely on the count being precise.

The following table lists the API action to use.

To do this...	Use this action	With AttributeName set to
Get the approximate number of messages in the queue	GetQueueAttributes	`ApproximateNumberOfMessages`
Get the approximate number of messages in the queue that are not visible	GetQueueAttributes	`ApproximateNumberOfMessagesNotVisible`

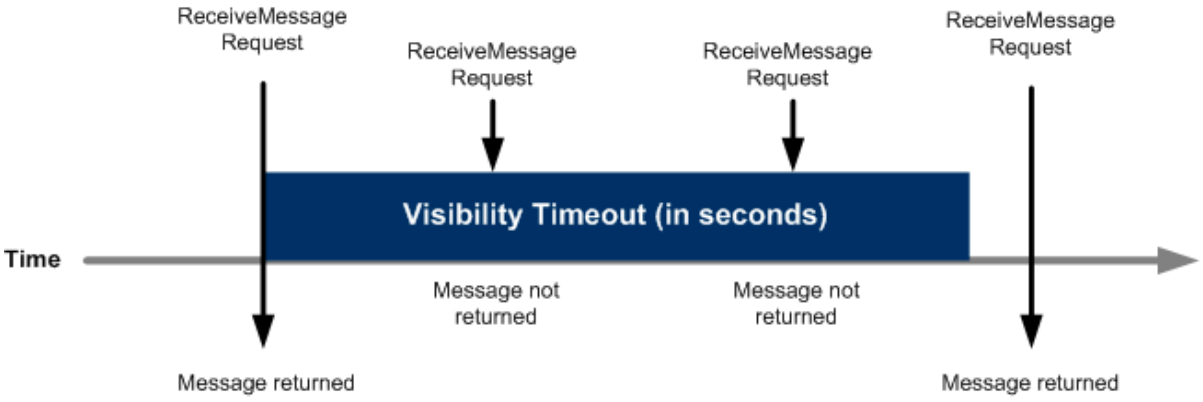
Visibility Timeout

Topics

When a consuming component in your system receives and processes a message from the queue, the message remains in the queue. Why doesn't Amazon SQS automatically delete it?

Because your system is distributed, there's no guarantee that the component will actually receive the message (it's possible the connection could break or the component could fail before receiving the message). Therefore, Amazon SQS does not delete the message, and instead, your consuming component must delete the message from the queue after receiving and processing it.

Immediately after the component receives the message, the message is still in the queue. However, you don't want other components in the system receiving and processing the message again. Therefore, Amazon SQS blocks them with a visibility timeout, which is a period of time during which Amazon SQS prevents other consuming components from receiving and processing that message. The following figure and discussion illustrate the concept.



General Recommendations for Visibility Timeout

The visibility timeout clock starts ticking once Amazon SQS returns the message. During that time, the component processes and deletes the message. But what happens if the component fails before deleting the message? If your system doesn't call DeleteMessage for that message before the visibility timeout expires, the message again becomes visible to the ReceiveMessage calls placed by the components in your system and it will be received again. If a message should only be received once, your system should delete it within the duration of the visibility timeout.

Each queue starts with a default setting of 30 seconds for the visibility timeout. You can change that setting for the entire queue. Typically, you'll set the visibility timeout to the average time it takes 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.

We recommend that if you have a system that produces messages that require varying amounts of time to process and delete, you create multiple queues, each with a different visibility timeout setting. Your system can then send all messages to a single queue that forwards each message to another queue with the appropriate visibility timeout based on the expected processing and deletion time for that message.

Extending a Message's Visibility Timeout

When you receive a message from a queue and begin processing it, you may find the visibility timeout for the queue is insufficient to fully process and delete that message. To give yourself more time to process the message, you can extend its visibility timeout by using the `ChangeMessageVisibility` action to specify a new timeout value. Amazon SQS restarts the timeout period using the new value.

For example, let's say the timeout for the queue is 30 seconds, and you receive a message from that queue. When you're 20 seconds into the timeout for that message (i.e., you have 10 seconds left), you want to give yourself 60 more seconds, so you immediately call `ChangeMessageVisibility` for the message with *`VisibilityTimeout`* set to 60 seconds. This means that you extended the message's visibility timeout from 30 seconds to 80 seconds: 20 seconds from the initial timeout setting plus 60 seconds from when you changed the timeout.

When you extend a message's visibility timeout, the new timeout applies only to that particular receipt of the message. `ChangeMessageVisibility` does not affect the timeout for the queue or later receipts of the message. If for some reason you don't delete the message and receive it again, its visibility timeout is the original value set for the queue.

Terminating a Message's Visibility Timeout

When you receive a message from the 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, which immediately makes the message visible to other components in the system to process. To do this, you call ChangeMessageVisibility with *`VisibilityTimeout`*=0 seconds.

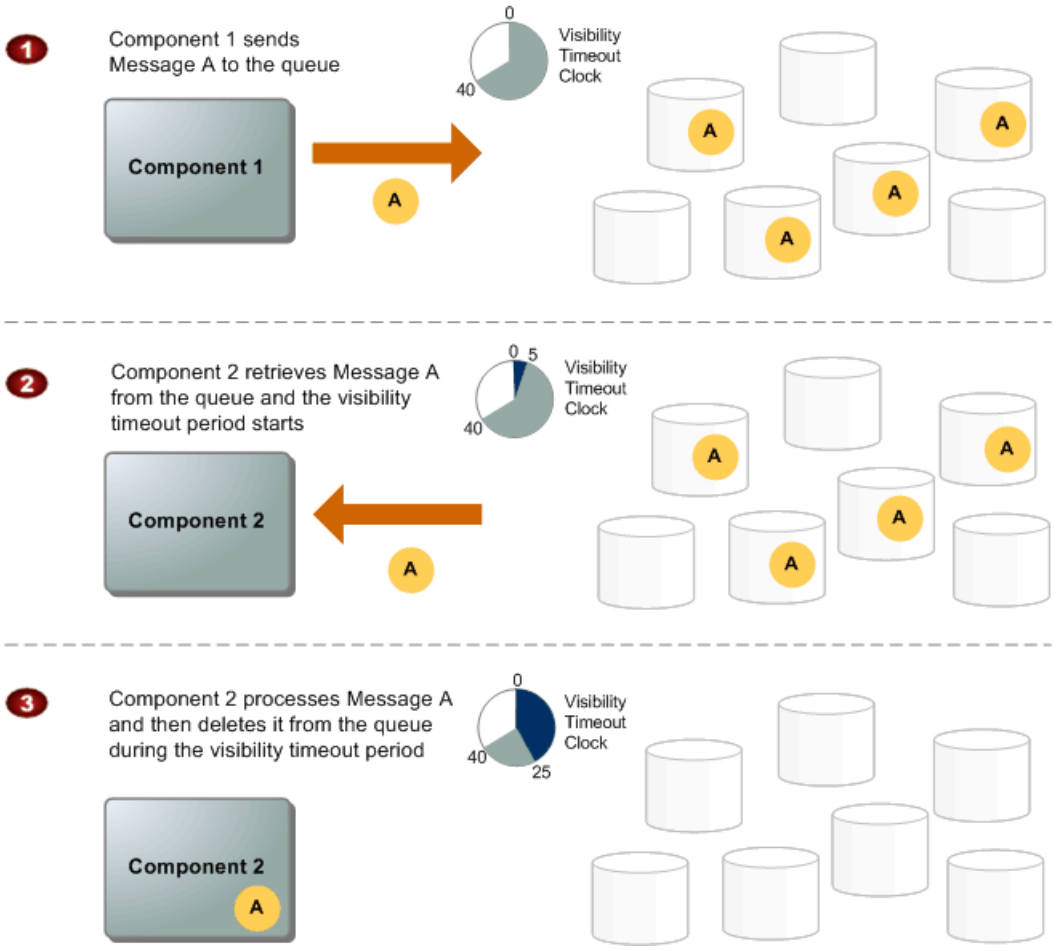
API Actions Related to Visibility Timeout

The following table lists the API actions to use to manipulate the visibility timeout. Use each action's *`VisibilityTimeout`* parameter to set or get the value.

To do this...	Use this action
Set the visibility timeout for a queue	SetQueueAttributes
Get the visibility timeout for a queue	GetQueueAttributes
Set the visibility timeout for the received messages without affecting the queue's visibility timeout	ReceiveMessage
Extending or terminating a message's visibility timeout	ChangeMessageVisibility
Extending or terminating the visibility timeout for up to ten messages.	ChangeMessageVisibilityBatch

Message Lifecycle

The following diagram and process describe the lifecycle of an Amazon SQS message, called *Message A*, from creation to deletion. Assume that a queue already exists.



Message Lifecycle

1	Component 1 sends Message A to a queue and the message is redundantly distributed across the SQS servers.
2	When Component 2 is ready to process a message, it retrieves messages from the queue, and Message A is returned. While Message A is being processed, it remains in the queue and is not returned to subsequent receive requests for the duration of the visibility timeout.
3	Component 2 deletes Message A from the queue to avoid the message being received and processed again once the visibility timeout expires.

Note

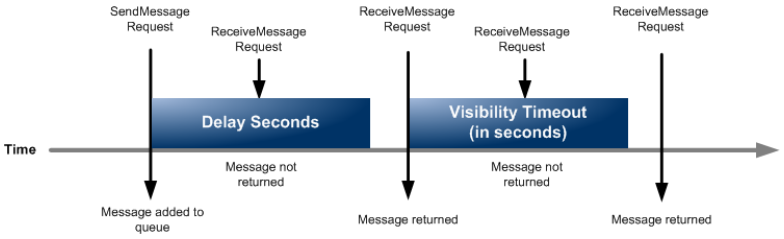
SQS automatically deletes messages that have been in a queue for more than maximum message retention period. The default message retention period is 4 days. However, you can set the message retention period to a value from 60 seconds to 1209600 seconds (14 days) with SetQueueAttributes.

Amazon SQS Delay Queues

Topics

Delay queues allow you to postpone the delivery of new messages in a queue for a specific number of seconds. If you create a delay queue, any message that you send to that queue will be invisible to consumers for the duration of the delay period. You can use `CreateQueue` to create a delay queue by setting the *`DelaySeconds`* attribute to any value between 0 and 900 (15 minutes). You can also turn an existing queue into a delay queue by using `SetQueueAttributes` to set the queue's *`DelaySeconds`* attribute.

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



To set delay seconds on individual messages, rather than for an entire queue, use message timers. If you send a message with a message timer, Amazon SQS uses the message timer's delay seconds value instead of the delay queue's delay seconds value. For more information, see Amazon SQS Message Timers (p. 14).

Important

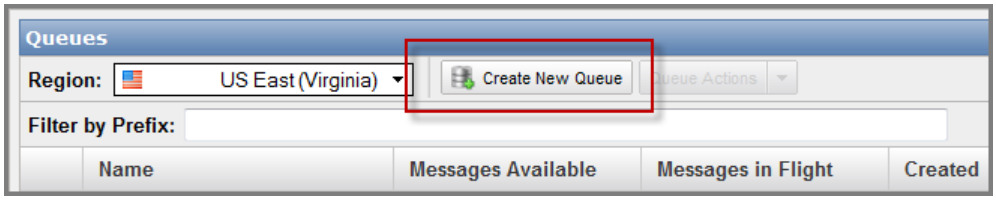
All Amazon SQS queues have delay functionality enabled. You can turn any queue that you create using the 2008-01-01 or 2009-02-01 API versions into a delay queue. However, message timers are available only in the 2011-10-01 API version. If you want to apply specific delay values to individual messages, you must use the 2011-10-01 API version.

Creating Delay Queues with the AWS Management Console

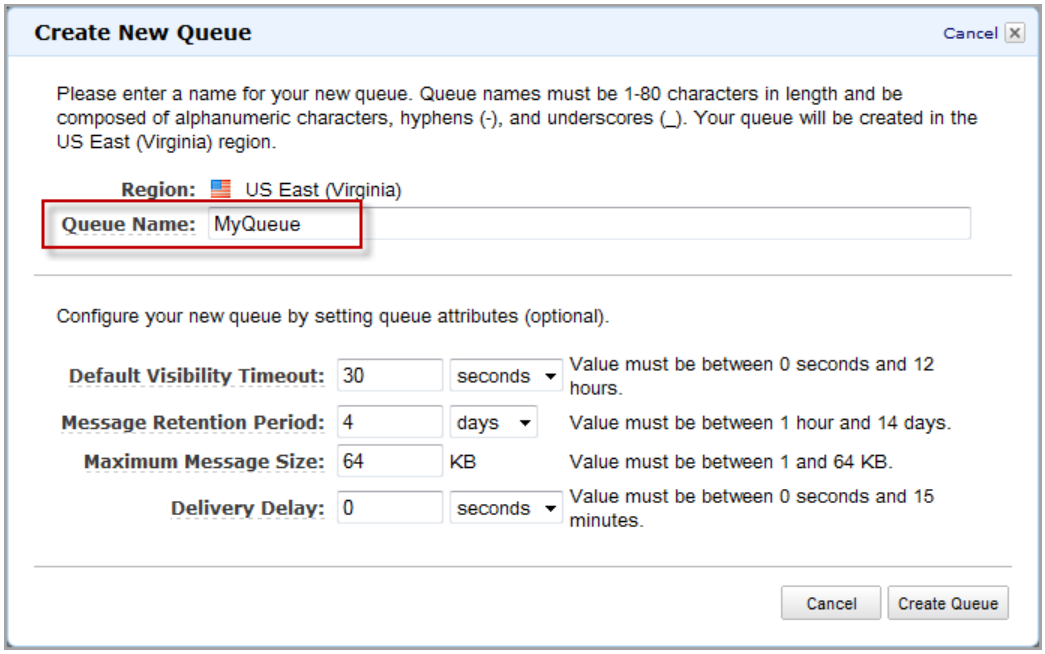
You can create a delay queue using the AWS Management Console by setting a **Delivery Delay** value that is greater than 0.

To create a delay queue with the AWS Management Console

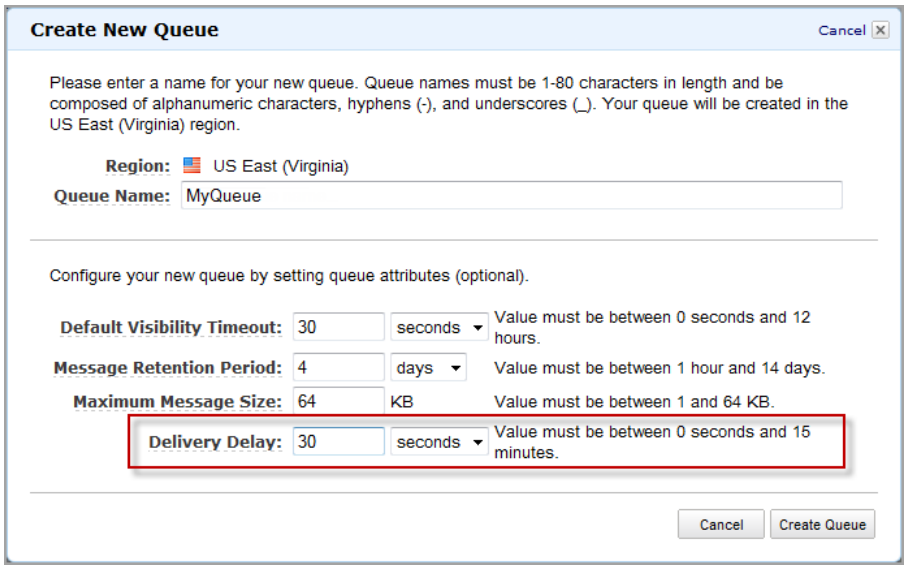
1. Sign in to the AWS Management Console and open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Click **Create New Queue**.



3. In the **Create New Queue** dialog box, enter a name for your queue (e.g., `MyQueue`) in the **Queue Name** field.



4. Enter a positive integer value (e.g., 30) for the Delivery Delay attribute.
   You can leave the default value settings for the remaining fields or enter new values.

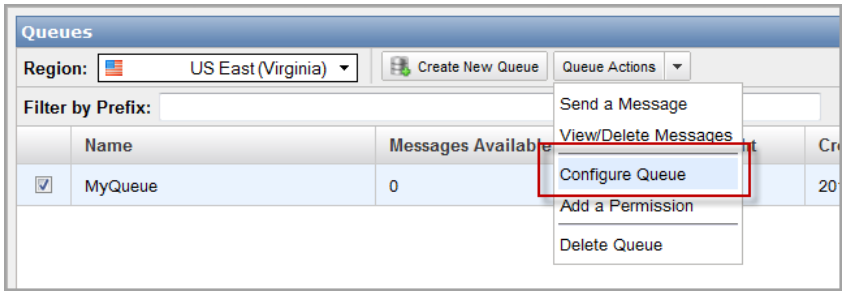


5. Click **Create Queue**.

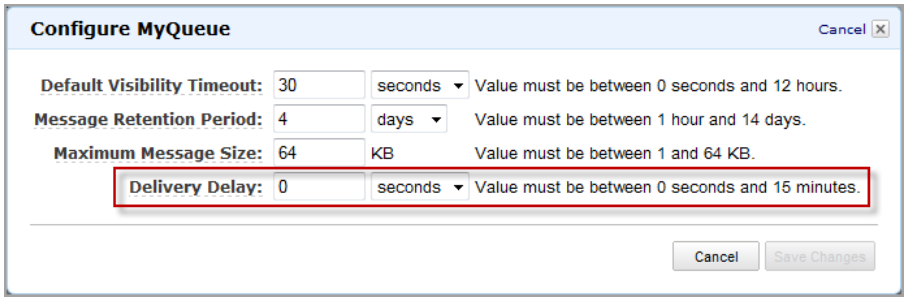
You can use the AWS Management Console to change the **Delivery Delay** setting for an existing queue by selecting the **Configure Queue** action with an existing queue highlighted.

To set a new delivery delay value for an existing queue

1. Select the **Configure Queue** action with an existing queue highlighted.



2. Change the value of **Delivery Delay** to a positive integer value.



3. Click **Save Changes**.

Creating Delay Queues with the Query API

The following Query API example calls the `CreateQueue` action to create a delay queue that hides each message from consumers for the first 45 seconds that the message is in the queue.

```
http://sqs.us-east-1.amazonaws.com/
?Action=CreateQueue
&QueueName=testQueue
&Attribute.1.Name=DelaySeconds
&Attribute.1.Value=45
&Version=2011-10-01
&SignatureMethod=HmacSHA256
&Expires=2011-10-20T22%3A52%3A43PST
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&SignatureVersion=2
&Signature=Dqlp3Sd6ljTUA9Uf6SGtEExwUQEXAMPLE
```

You can also change an existing queue into a delay queue by changing the DelaySeconds attribute from its default value of 0 to a positive integer value that is less than or equal to 900. The following example calls `SetQueueAttributes` to set the *`DelaySeconds`* attribute of a queue named `testQueue` to 45 seconds.

```
http://sqs.us-east-1.amazonaws.com/123456789012/testQueue/
?Action=SetQueueAttributes
&Attribute.Name=DelaySeconds
&Attribute.Value=45
&Version=2011-10-01
&SignatureMethod=HmacSHA256
&Expires=2011-10-20T22%3A52%3A43PST
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&SignatureVersion=2
&Signature=Dqlp3Sd6ljTUA9Uf6SGtEExwUQEXAMPLE
```

Amazon SQS Message Timers

Topics

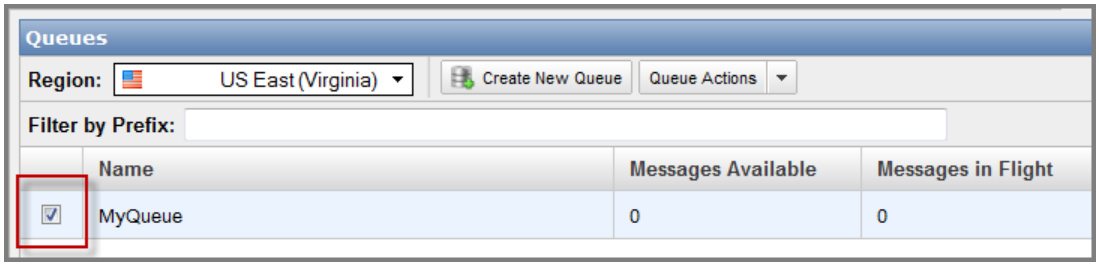
Amazon SQS message timers allow you to specify an initial invisibility period for a message that you are adding to a queue. For example, if you send a message with the *DelaySeconds* parameter set to 45, the message will not be visible to consumers for the first 45 seconds that the message resides in the queue. The default value for `DelaySeconds` is 0.

To set a delay period that applies to all messages in a queue, use delay queues. For more information, see Amazon SQS Delay Queues (p. 10). A message timer setting for an individual message overrides any `DelaySeconds` value that applies to the entire delay queue.

Creating Message Timers Using the Console

To send a message with a message timer using the AWS Management Console

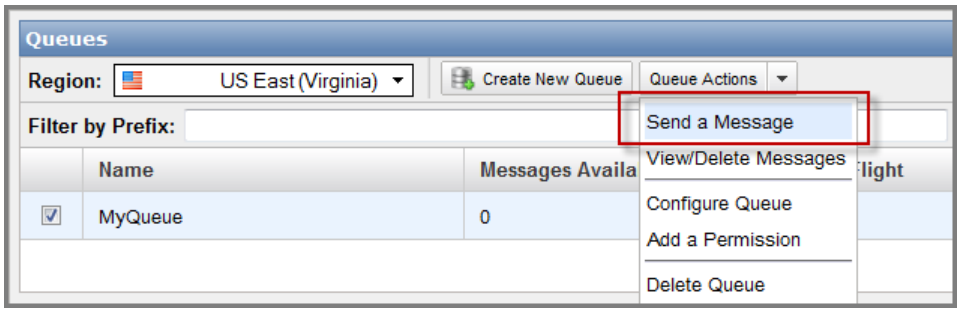
1. Sign in to the AWS Management Console and open the Amazon SQS console at https://console.aws.amazon.com/sqs/.
2. Select a queue.



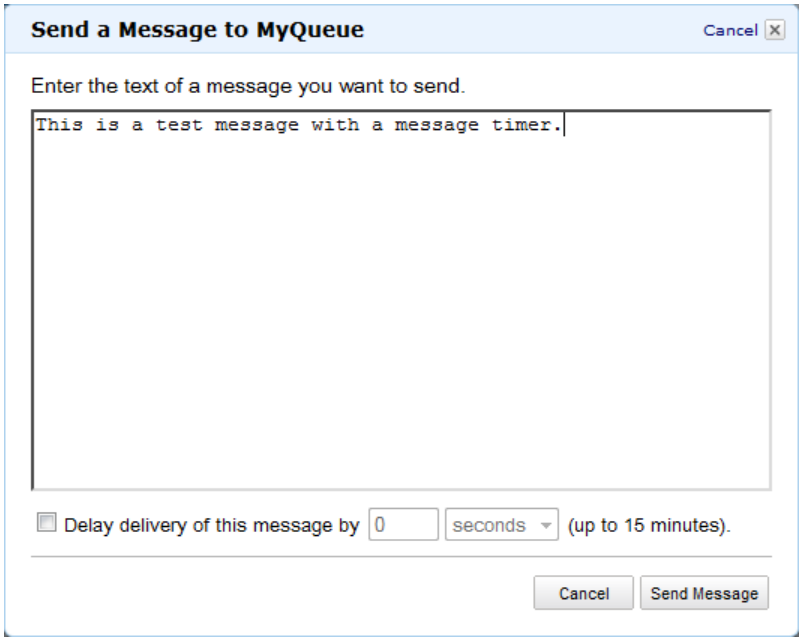
3. Select **Send a Message** from the **Queue Actions** drop-down list.

   Note

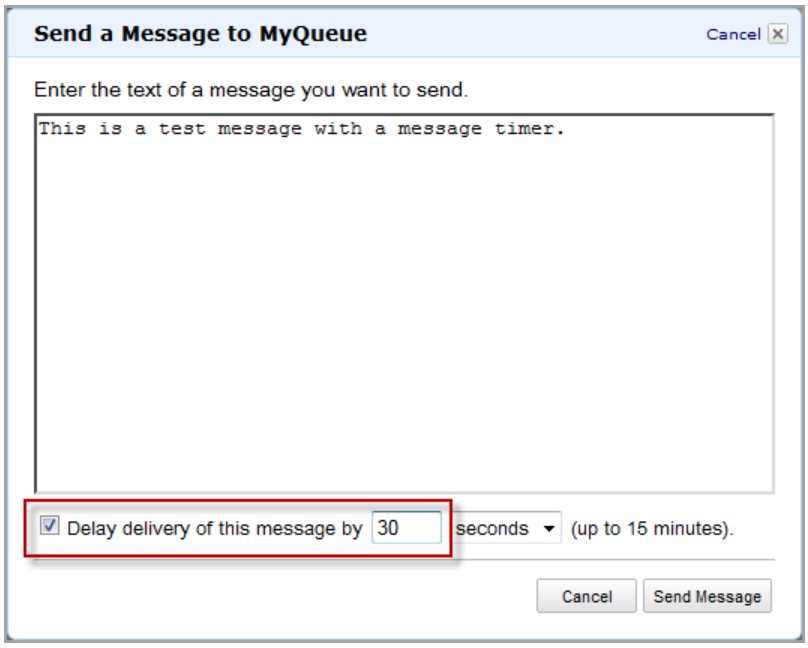
   The **Queue Actions** drop-down list is available only if a queue is selected.



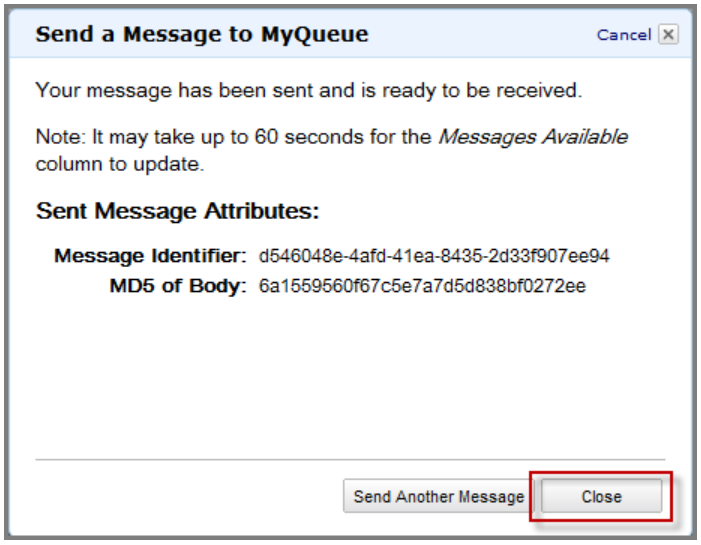
4. In the **Send a Message to MyQueue** dialog box, enter a message (e.g., `This is a test message with a message timer.`).



5. Enter a delay value (e.g., `30`) in the **Delay delivery of this message by** text box.



6. Click **Send Message**.
7. In the **Send a Message to MyQueue** confirmation box click **Close**.



Creating Message Timers Using the Query API

The following Query API example applies a 45 second initial visibility delay for a single message sent with `SendMessage`.

```
http://sqs.us-east-1.amazonaws.com/123456789012/testQueue/
?Action=SendMessage
&MessageBody=This+is+a+test+message
```

```
&Attribute.Name=DelaySeconds
&Attribute.Value=45
&Version=2011-10-01
&SignatureMethod=HmacSHA256
&Expires=2011-10-18T22%3A52%3A43PST
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&SignatureVersion=2
&Signature=Dqlp3Sd6ljTUA9Uf6SGtEExwUQEXAMPLE
```

You can also use the Query API `SendMessageBatch` action to send up to ten messages with message timers. You can assign a different *`DelaySeconds`* value to each message or assign no value at all. If you do not set a value for DelaySeconds, the message might still be subject to a delay if you are adding the message to a delay queue. For more information about delay queues, see Amazon SQS Delay Queues (p. 10). The following example uses `SendMessageBatch` to send three messages: one message without a message timer and two messages with different values for *`DelaySeconds`*.

```
http://sqs.us-east-1.amazonaws.com/123456789012/testQueue/
?Action=SendMessageBatch
&SendMessageBatchRequestEntry.1.Id=test_msg_no_message_timer
&SendMessageBatchRequestEntry.1.MessageBody=test%20message%20body%201
&SendMessageBatchRequestEntry.2.Id=test_msg_delay_45_seconds
&SendMessageBatchRequestEntry.2.MessageBody=test%20message%20body%202
&SendMessageBatchRequestEntry.2.DelaySeconds=45
&SendMessageBatchRequestEntry.3.Id=test_msg_delay_2_minutes
&SendMessageBatchRequestEntry.3.MessageBody=test%20message%20body%203
&SendMessageBatchRequestEntry.3.DelaySeconds=120
&Version=2011-10-01
&SignatureMethod=HmacSHA256
&Expires=2011-10-18T22%3A52%3A43PST
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&SignatureVersion=2
&Signature=Dqlp3Sd6ljTUA9Uf6SGtEExwUQEXAMPLE
```

Amazon SQS Batch API actions

Topics

In the 2009-02-01 API version of Amazon SQS, only one action—`ReceiveMessage`—supports batch processing, i.e., processing more than one message with a single call. With the 2011-10-01 API version, Amazon SQS adds batch functionality for sending messages, deleting messages, and changing message visibility timeout values. To send up to ten messages at once, use the `SendMessageBatch` action. To delete up to ten messages with one API call, use the `DeleteMessageBatch` action. To change the visibility timeout value for up to ten messages, use the `ChangeMessageVisibilityBatch` action.

To use the new batch actions, you must use either the Query API or a Software Development Kit (SDK) that supports the new batch actions. Check your specific SDK's documentation to see whether it supports the new Amazon SQS batch actions. The Amazon SQS console does not currently support the batch API actions.

For details and examples of the three batch API actions, go to the *Amazon Simple Queue Service API Reference*:

- ChangeMessageVisibilityBatch
- DeleteMessageBatch
- SendMessageBatch

Maximum Message Size for SendMessageBatch

You can send a message as large as 65,536 bytes (64 KiB) with `SendMessageBatch`. However, the total size of all the messages that you send in a single call to `SendMessageBatch` cannot exceed 65,536 bytes (64 KiB).

Making API Requests

Topics

This section describes how to make requests to Amazon SQS. The various topics acquaint you with the basic differences between the interfaces, the components of a request, how to authenticate a request, and the content of responses.

Endpoints

For information about this product's regions and endpoints, go to Regions and Endpoints in the Amazon Web Services General Reference.

For example, to create a queue in Europe, you would generate a Query request similar to the following:

```
http://sqs.eu-west-1.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=testQueue
&Version=2009-02-01
&SignatureMethod=HmacSHA256
&Expires=2009-04-18T22%3A52%3A43PST
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&SignatureVersion=2
&Signature=Dqlp3Sd6ljTUA9Uf6SGtEExwUQEXAMPLE
```

Each Amazon SQS endpoint is entirely independent. For example, if you have two queues called "MyQueue," one in sqs.us-east-1.amazonaws.com and one in sqs.eu-west-1.amazonaws.com, they are completely independent and do not share any data.

Making Query Requests

Topics

Amazon SQS supports Query requests for calling service actions. Query requests are simple HTTP or HTTPS requests, using the GET or POST method. Query requests must contain an *Action* parameter to indicate the action to be performed. The response is an XML document that conforms to a schema.

Structure of a GET Request

This guide presents the Amazon SQS GET requests as URLs, which can be used directly in a browser. The URL consists of:

- **Endpoint—**The resource the request is acting on (in the case of SQS, the endpoint is a queue)
- **Action—**The action you want to perform on the endpoint; for example: sending a message
- **Parameters—**Any request parameters

The following is an example GET request to send a message to an SQS queue.

```
http://sqs.us-east-1.amazonaws.com/123456789012/queue1?Action=SendMessage&Mes
sageBody=Your%20Message%20Text&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE&Version=2011-
10-01&Expires=2008-02-10T12:00:00Z&Signature=lBP67vCvGlDMBQ1do
fZxg8E8SUEXAMPLE&SignatureVersion=2&SignatureMethod=HmacSHA256
```

Important

Because the GET requests are URLs, you must URL encode the parameter values. For example, in the preceding example request, the value for the *MessageBody* parameter is actually `Your Message Text`. However, spaces are not allowed in URLs, so each space is URL encoded as "`%20`". The rest of the example has not been URL encoded to make it easier for you to read.

To make the GET examples even easier to read, this guide presents them in the following parsed format.

```
http://sqs.us-east-1.amazonaws.com/123456789012/queue1
?Action=SendMessage
&MessageBody=Your%20Message%20Text
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2011-10-01
&Expires=2011-10-15T12:00:00Z
&Signature=lBP67vCvGlDMBQ1dofZxg8E8SUEXAMPLE
&SignatureVersion=2
&SignatureMethod=HmacSHA256
```

Note

In the example Query requests we present in this guide, we use a false AWS Access Key ID and false signature, each with `EXAMPLE` appended. We do this to indicate that you shouldn't expect the signature in the example to be accurate based on the request parameters presented in the example. The one exception to this is in the instructions for creating Query request

signatures. The example there shows a real signature based on a particular AWS Access Key ID we specify and the request parameters in the example (for more information, see Query Request Authentication (p. 31)).

In SQS, all parameters except *MessageBody* always have values that have no spaces. The value you provide for *MessageBody* in SendMessage requests can have spaces. In this guide, any example SendMessage Query requests with a *MessageBody* that includes spaces is displayed with the spaces URL encoded (as %20). For clarity, the rest of the URL is not displayed in a URL encoded format.

The first line represents the *endpoint* of the request. This is the resource the request acts on. The preceding example acts on a queue, so the request's endpoint is the queue's identifier, known as the queue URL. For more details about the queue URL, see Queue URLs (p. 5).

After the endpoint is a question mark (?), which separates the endpoint from the parameters. Each parameter is separated by an ampersand (&).

The *Action* parameter indicates the action to perform (for a list of the actions, see API Actions in the Amazon SQS API Reference). For a list of the other parameters that are common to all Query requests, see Request Parameters Common to All Actions in the Amazon SQS API Reference.

Structure of a POST Request

SQS also accepts POST requests. With a POST request, you send the query parameters as a form in the HTTP request body as described in the following procedure.

To create a POST request

1. Assemble the query parameter names and values into a form.

   This means you put the parameters and values together like you would for a GET request (with an ampersand separating each name-value pair). The following example shows a SendMessage request with the line breaks we use in this guide to make the information easier to read.

   ```
   Action=SendMessage
   &MessageBody=Your Message Text
   &AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
   &Version=2011-10-01
   &Expires=2011-10-15T12:00:00Z
   &SignatureVersion=2
   &SignatureMethod=HmacSHA256
   ```

2. Form-URL-encode the form according to the *Form Submission* section of the HTML specification (for more information, go to http://www.w3.org/MarkUp/html-spec/html-spec_toc.html#SEC8.2.1).

   ```
   Action=SendMessage
   &MessageBody=Your+Message+Text
   &AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
   &Version=2011-10-01
   &Expires=2011-10-15T12%3A00%3A00Z
   &SignatureVersion=2
   &SignatureMethod=HmacSHA256
   ```

3. Add the request signature to the form (for more information, see Query Request Authentication (p. 31)).

```
Action=SendMessage
&MessageBody=Your+Message+Text
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2011-10-01
&Expires=2011-10-15T12%3A00%3A00Z
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Signature=lBP67vCvGlDMBQ1dofZxg8E8SUEXAMPLE
```

4. Provide the resulting form as the body of the POST request.
5. Include the `Content-Type` HTTP header with the value set to `application/x-www-form-urlencoded`.

The following example shows the final POST request.

```
POST /queue1 HTTP/1.1
Host: sqs.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded

Action=SendMessage
&MessageBody=Your+Message+Text
&AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
&Version=2011-10-01
&Expires=2011-10-15T12%3A00%3A00Z
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Signature=lBP67vCvGlDMBQ1dofZxg8E8SUEXAMPLE
```

SQS requires no other HTTP headers in the request besides `Content-Type`. The authentication signature you provide is the same signature you would provide if you sent a GET request (for information about the signature, see Query Request Authentication (p. 31)).

Note

Your HTTP client typically adds other items to the HTTP request as required by the version of HTTP the client uses. We don't include those additional items in the examples in this guide.

Related Topics

- Query Request Authentication (p. 31)
- Responses (p. 35)

Making SOAP Requests

Important

As of August 8, 2011, Amazon SQS no longer supports SOAP requests.

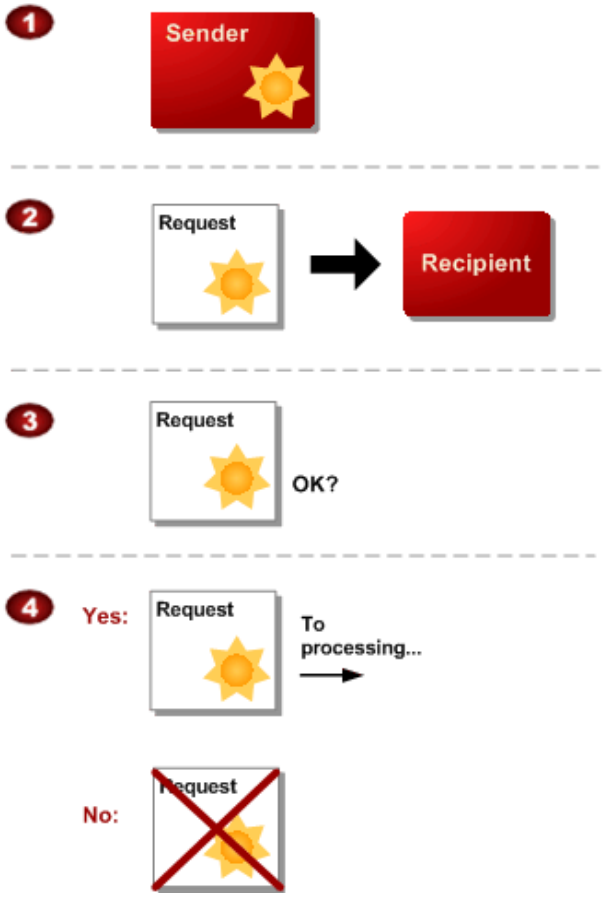
Request Authentication

Topics

The topics in this section describe how Amazon SQS authenticates your requests. In this section you can learn about the basics of authentication, how your AWS account and identifiers are used to support authentication, and how to create an HMAC-SHA1 signature. This section also covers the request authentication requirements for Query requests.

What Is Authentication?

Authentication is a process for identifying and verifying who is sending a request. The following diagram shows a simplified version of an authentication process.



General Process of Authentication

1	The sender obtains the necessary credential.
2	The sender sends a request with the credential to the recipient.
3	The recipient uses the credential to verify the sender truly sent the request.
4	If yes, the recipient processes the request. If no, the recipient rejects the request and responds accordingly.

During authentication, AWS verifies both the identity of the sender and whether the sender is registered to use services offered by AWS. If either test fails, the request is not processed further.

For further discussion of authentication, go to the techencylopedia.com entry for authentication. For definitions of common industry terms related to authentication, go to the RSA Laboratories Glossary.

The subsequent sections describe how SQS implements authentication to protect your data.

Your AWS Account

To access any web services offered by AWS, you must first create an AWS account at http://aws.amazon.com. An AWS account is simply an Amazon.com account that is enabled to use AWS products; you can use an existing Amazon.com account login and password when creating the AWS account.

Alternately, you could create a new AWS-enabled Amazon.com account by using a new login and password. The e-mail address you provide as the account login must be valid. You'll be asked to provide a credit card or other payment method to cover the charges for any AWS products you use.

From your AWS account you can view your AWS account activity, view usage reports, and manage your AWS account access identifiers.

Related Topics

- Your AWS Identifiers (p. 25)

Your AWS Identifiers

When you create an AWS account, AWS assigns you a pair of related identifiers:

- Access Key ID (a 20-character, alphanumeric sequence)
  For example: AKIAIOSFODNN7EXAMPLE
- Secret Access Key (a 40-character sequence)
  For example: # wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY

These are your *AWS access key identifiers*.

Caution

Your Secret Access Key is a secret and only you and AWS should know it. It is important to keep it confidential to protect your account. Never include it in your requests to AWS, and never e-mail it to anyone. Do not share it 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 Access Key.

The Access Key ID is associated with your AWS account. You include it in AWS service requests to identify yourself as the sender of the request.

The Access Key ID is not a secret, and anyone could use your Access Key ID in requests to AWS. To provide proof that you truly are the sender of the request, you must also include a digital signature. For all requests, you calculate the signature using your Secret Access Key. AWS uses the Access Key ID in the request to look up your Secret Access Key and then calculates a digital signature with the key. If the signature AWS calculates matches the signature you sent, the request is considered authentic. Otherwise, the request fails authentication and is not processed.

Related Topics

- HMAC-SHA Signatures (p. 26)
- Query Request Authentication (p. 31)

Viewing Your AWS Identifiers

Your Access Key ID and Secret Access Key are displayed to you when you create your AWS account. They are not e-mailed to you. If you need to see them again, you can view them at any time from your AWS account.

To view your AWS access identifiers

1. Log in to the AWS security credentials web site.
2. Scroll down to the **Access Credentials** section and select the **Access Keys** tab.
3. Locate an active Access Key in the **Your Access Keys** list.
4. To display the Secret Access Key, click **Show** in the **Secret Access Key** column.

Related Topics

- Your AWS Account (p. 25)
- Your AWS Identifiers (p. 25)

HMAC-SHA Signatures

Topics

The topics in this section describe how Amazon SQS uses HMAC-SHA signatures to authenticate Query requests.

Required Authentication Information

When accessing Amazon SQS using the Query API, you must provide the following items so the request can be authenticated:

- **AWS Access Key ID—**Your AWS account is identified by your Access Key ID, which AWS uses to look up your Secret Access Key.
- **Signature—**Each request must contain a valid HMAC-SHA request signature, or the request is rejected. You calculate the request signature by using your Secret Access Key, which is a shared secret known only to you and AWS.
- **Date—**Each request must contain the time stamp of the request. You can provide an expiration date and time for the request instead of or in addition to the time stamp.

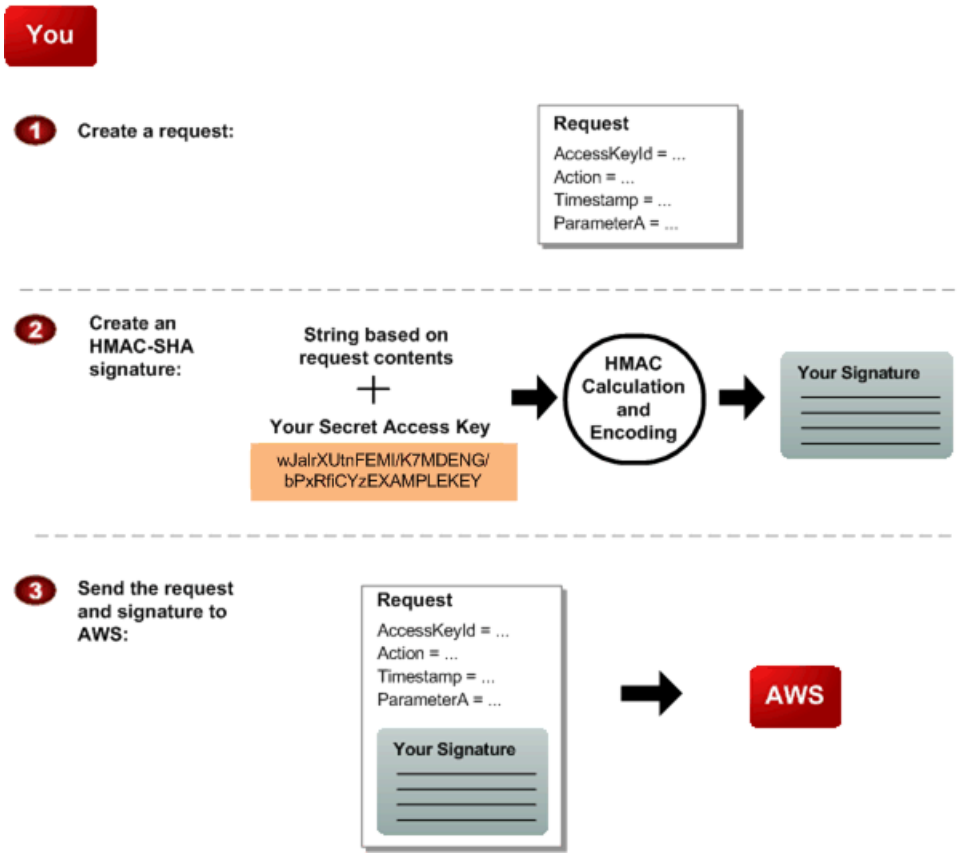
Related Topics

- Your AWS Identifiers (p. 25)

Basic Authentication Process

Following is the series of tasks required to authenticate requests to AWS using an HMAC-SHA request signature. It is assumed you have already created an AWS account and received an Access Key ID and Secret Access Key. For more information about those, see Your AWS Account (p. 25) and Your AWS Identifiers (p. 25).

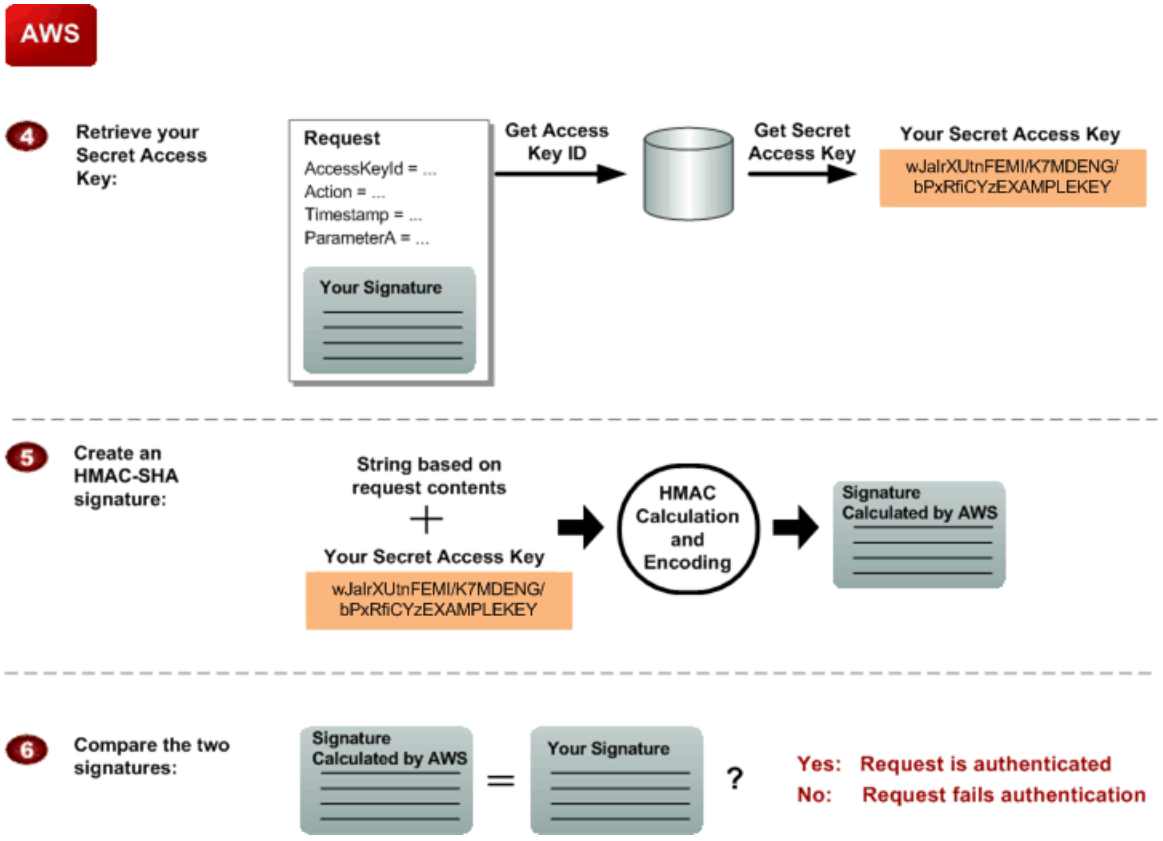
You perform the first three tasks.



Process for Authentication: Tasks You Perform

1	You construct a request to AWS.
2	You calculate a keyed-hash message authentication code (HMAC-SHA) signature using your Secret Access Key (for information about HMAC, go to http://www.faqs.org/rfcs/rfc2104.html)
3	You include the signature and your Access Key ID in the request, and then send the request to AWS.

AWS performs the next three tasks.



Process for Authentication: Tasks AWS Performs

4	AWS uses the Access Key ID to look up your Secret Access Key.
5	AWS generates a signature from the request data and the Secret Access Key using the same algorithm you used to calculate the signature you sent in the request.
6	If the signature generated by AWS matches the one you sent in the request, the request is considered authentic. If the comparison fails, the request is discarded, and AWS returns an error response.

About the String to Sign

Each AWS request you send must include an HMAC-SHA request signature calculated with your Secret Access Key. The details are covered in Query Request Authentication (p. 31).

About the Time Stamp

The time stamp (or expiration time) you use in the request must be a `dateTime` object, with the complete date plus hours, minutes, and seconds (for more information, go to http://www.w3.org/TR/xmlschema-2/#dateTime). For example: 2007-01-31T23:59:59Z. Although it is not required, we recommend you provide the time stamp in the Coordinated Universal Time (Greenwich Mean Time) time zone.

If you specify a time stamp (instead of an expiration time), the request automatically expires 15 minutes after the time stamp (in other words, AWS does not process a request if the request time stamp is more than 15 minutes earlier than the current time on AWS servers). Make sure your server's time is set correctly.

Important

If you are using .NET you must not send overly specific time stamps, due to different interpretations of how extra time precision should be dropped. To avoid overly specific time stamps, manually construct `dateTime` objects with no more than millisecond precision.

Java Sample Code for Base64 Encoding

Request signatures must be base64 encoded. The following Java sample code shows how to perform base64 encoding.

```
package amazon.webservices.common;
/**
* This class defines common routines for encoding data in AWS requests.
*/
public class Encoding {
/**
* Performs base64-encoding of input bytes.
*
* @param rawData * Array of bytes to be encoded.
* @return * The base64 encoded string representation of rawData.
*/
public static String EncodeBase64(byte[] rawData) {
return Base64.encodeBytes(rawData);
}
}
```

Java Sample Code for Calculating HMAC-SHA1 Signatures

The following Java code sample shows how to calculate an HMAC request signature.

```
package amazon.webservices.common;

import java.security.SignatureException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

/**
* This class defines common routines for generating
* authentication signatures for AWS requests.
*/
public class Signature {
private static final String HMAC_SHA1_ALGORITHM = "HmacSHA1";


/**
* Computes RFC 2104-compliant HMAC signature.
* * @param data
* The data to be signed.
* @param key
* The signing key.
```

```
* @return
* The Base64-encoded RFC 2104-compliant HMAC signature.
* @throws
* java.security.SignatureException when signature generation fails
*/
public static String calculateRFC2104HMAC(String data, String key)
throws java.security.SignatureException
{
String result;
try {

// get an hmac_sha1 key from the raw key bytes
SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), HMAC_SHA1_AL
GORITHM);

// get an hmac_sha1 Mac instance and initialize with the signing key
Mac mac = Mac.getInstance(HMAC_SHA1_ALGORITHM);
mac.init(signingKey);

// compute the hmac on input data bytes
byte[] rawHmac = mac.doFinal(data.getBytes());

// base64-encode the hmac
result = Encoding.EncodeBase64(rawHmac);

} catch (Exception e) {
throw new SignatureException("Failed to generate HMAC : " + e.getMessage());
}
return result;
}
}
```

Query Request Authentication

You can send Query requests over either HTTP or HTTPS. Regardless of which protocol you use, you must include a signature in every Query request. This section describes how to create the signature. The method described in the following procedure is known as *signature version 2*.

Caution

If you are currently using signature version 1: Version 1 is deprecated, and you should move to signature version 2 immediately. For information about the deprecation schedule and the differences between signature version 2 and version 1, go to Making Secure Requests to Amazon Web Services.

To create the signature

1. Create the canonicalized query string that you need later in this procedure:

   a. Sort the UTF-8 query string components by parameter name with natural byte ordering. The parameters can come from the GET URI or from the POST body (when `Content-Type` is `application/x-www-form-urlencoded`).

   b. URL encode the parameter name and values according to the following rules:

      - Do not URL encode any of the unreserved characters that RFC 3986 defines. These unreserved characters are A-Z, a-z, 0-9, hyphen (-), underscore (_), period (.), and tilde (~).

- Percent encode all other characters with %XY, where X and Y are hex characters 0-9 and uppercase A-F.
- Percent encode extended UTF-8 characters in the form %XY%ZA....
- Percent encode the space character as %20 (and not +, as common encoding schemes do).

Note

Currently all AWS service parameter names use unreserved characters, so you don't need to encode them. However, you might want to include code to handle parameter names that use reserved characters, for possible future use.

c. Separate the encoded parameter names from their encoded values with the equals sign (=) (ASCII character 61), even if the parameter value is empty.

d. Separate the name-value pairs with an ampersand (&) (ASCII code 38).

2. Create the string to sign according to the following pseudo-grammar (the "\n" represents an ASCII newline).

```
StringToSign = HTTPVerb + "\n" +
               ValueOfHostHeaderInLowercase + "\n" +
               HTTPRequestURI + "\n" +
               CanonicalizedQueryString <from the preceding step>
```

The HTTPRequestURI component is the HTTP absolute path component of the URI up to, but not including, the query string. If the HTTPRequestURI is empty, use a forward slash (/).

3. Calculate an RFC 2104-compliant HMAC with the string you just created, your Secret Access Key as the key, and SHA256 or SHA1 as the hash algorithm.
For more information, go to http://www.ietf.org/rfc/rfc2104.txt.
4. Convert the resulting value to base64.
5. Use the resulting value as the value of the *Signature* request parameter.

Important

The final signature you send in the request must be URL encoded as specified in RFC 3986 (for more information, go to http://www.ietf.org/rfc/rfc3986.txt). If your toolkit URL encodes your final request, then it handles the required URL encoding of the signature. If your toolkit doesn't URL encode the final request, then make sure to URL encode the signature before you include it in the request. Most importantly, make sure the signature is URL encoded *only once*. A common mistake is to URL encode it manually during signature formation, and then again when the toolkit URL encodes the entire request.

Example SetQueueAttributes Request

```
https://sqs.us-east-1.amazonaws.com/770098461991/queue2
?Action=SetQueueAttributes
&Attribute.Name=VisibilityTimeout
&Attribute.Value=90
&Version=2009-02-01
&Expires=2008-02-10T12%3A00%3A00Z
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&AWSAccessKeyId=<Your AWS Access Key ID>
```

Following is the string to sign.

```
GET\n
sqs.us-east-1.amazonaws.com\n
/770098461991/queue2\n
AWSAccessKeyId=<Your AWS Access Key ID>
&Action=SetQueueAttributes
&Attribute.Name=VisibilityTimeout
&Attribute.Value=90
&Expires=2008-02-10T12%3A00%3A00Z
&SignatureMethod=HmacSHA256
&SignatureVersion=2
&Version=2009-02-01
```

Following is the signed request.

```
https://sqs.us-east-1.amazonaws.com/770098461991/queue2
?Action=SetQueueAttributes
&Attribute.Name=VisibilityTimeout
&Attribute.Value=35
&Version=2009-02-01
&Expires=2008-02-10T12%3A00%3A00Z
&Signature=<URLEncode(Base64Encode(Signature))>
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&AWSAccessKeyId=<Your AWS Access Key ID>
```

Example SendMessage Request Using POST

```
POST /queue2 HTTP/1.1
Host: sqs.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded

Action=SendMessage
&MessageBody=Your+Message+Text
&Version=2009-02-01
&Expires=2008-02-10T12%3A00%3A00Z
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&AWSAccessKeyId=<Your AWS Access Key ID>
```

Following is the string to sign. Notice that you encode the spaces as %20 (and not plus signs) when you form the string to sign.

```
POST\n
sqs.us-east-1.amazonaws.com\n
/queue2\n
AWSAccessKeyId=<Your AWS Access Key ID>
&Action=SendMessage
&Expires=2008-02-10T12%3A00%3A00Z
&MessageBody=Your%20Message%20Text
&SignatureMethod=HmacSHA256
&SignatureVersion=2
&Version=2009-02-01
```

Following is the signed request.

```
POST /queue2 HTTP/1.1
Host: sqs.us-east-1.amazonaws.com
Content-Type: application/x-www-form-urlencoded

Action=SendMessage
&MessageBody=Your+Message+Text
&Version=2009-02-01
&Expires=2008-02-10T12%3A00%3A00Z
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&AWSAccessKeyId=<Your AWS Access Key ID>
&Signature=URLEncode(Base64Encode(Signature))
```

Responses

Topics

In response to an action request, SQS returns an XML data structure that contains the results of the request. This data conforms to the SQS schema. For more information, see WSDL Location and API Version in the Amazon SQS API Reference.

Structure of a Successful Response

If the request succeeded, the main response element is named after the action, but with "Response" appended. For example, `CreateQueueResponse` is the response element returned for a successful `CreateQueue` request. This element contains the following child elements:

- `ResponseMetadata`, which contains the `RequestId` child element
- An optional element containing action-specific results; for example, the `CreateQueueResponse` element includes an element called `CreateQueueResult`

The XML schema describes the XML response message for each SQS action.

The following is an example of a successful response.

```
<CreateQueueResponse
  xmlns=http://sqs.us-east-1.amazonaws.com/doc/2011-10-01/
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:type=CreateQueueResponse>
     <CreateQueueResult>
        <QueueUrl>
        http://sqs.us-east-1.amazonaws.com/770098461991/queue2
        </QueueUrl>
     </CreateQueueResult>
     <ResponseMetadata>
        <RequestId>cb919c0a-9bce-4afe-9b48-9bdf2412bb67</RequestId>
     </ResponseMetadata>
</CreateQueueResponse>
```

Structure of an Error Response

If a request is unsuccessful, the main response element is called `ErrorResponse` regardless of the action that was called. This element contains an `Error` element and a `RequestId` element. Each `Error` includes:

- A `Type` element that identifies whether the error was a receiver or sender error
- A `Code` element that identifies the type of error that occurred
- A `Message` element that describes the error condition in a human-readable form
- A `Detail` element that might give additional details about the error or might be empty

The following is an example of an error response.

```
<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>
```

Related Topics

- Making Query Requests (p. 21)

Shared Queues

Topics

Amazon SQS includes methods to share your queues so others can use them, using permissions set in an access control policy. A permission gives access to another person to use your queue in some particular way. A policy is the actual document that contains the permissions you've granted.

Amazon SQS offers two methods for setting a policy: a simple API and an advanced API. In the simple API, SQS generates an access control policy for you. In the advanced API, you create the access control policy.

Simple API for Shared Queues

The simple API for sharing a queue has two operations:

- AddPermission
- RemovePermission

With the Simple API, Amazon SQS writes the policy in the required language for you based on the information you include in the `AddPermission` operation. However, the policy that Amazon SQS generates is limited in scope. You can grant permissions to principals, but you can't specify restrictions.

Advanced API for Shared Queues

With the advanced API, you write the policy yourself directly in the access policy language and upload the policy with the SetQueueAttributes operation. The advanced API allows you to deny access or to apply finer access restrictions (for example, based on time or based on IP address).

If you choose to write your own policies, you need to understand how policies are structured. For complete reference information about policies, see Using The Access Policy Language (p. 41). For examples of policies, see Amazon SQS Policy Examples (p. 66).

Understanding Permissions

A permission is the type of access you give to a principal (the user receiving the permission). You give each permission a label that identifies that permission. If you want to delete that permission in the future, you use that label to identify the permission. If you want to see what permissions are on a queue, use the GetQueueAttributes operation. Amazon SQS returns the entire policy (containing all the permissions).

Amazon SQS supports the permission types shown in the following table.

Permission	Description
*	This permission type grants the following actions to a principal on a shared queue: receive messages, send messages, delete messages, change a message's visibility, get a queue's attributes.
ReceiveMessage	This grants permission to receive messages in the queue.

Permission	Description
SendMessage	This grants permission to send messages to the queue. `SendMessageBatch` inherits permissions associated with `SendMessage`.
DeleteMessage	This grants permission to delete messages from the queue. `DeleteMessageBatch` inherits permissions associated with `DeleteMessage`.
ChangeMessageVisibility	This grants permission to extend or terminate the read lock timeout of a specified message. `ChangeMessageVisibilityBatch` inherits permissions associated with `ChangeMessageVisibility`. For more information about visibility timeout, see Visibility Timeout (p. 7). For more information about this permission type, see the ChangeMessageVisibility operation.
GetQueueAttributes	This grants permission to receive all of the queue attributes except the policy, which can only be accessed by the queue's owner. For more information, see the GetQueueAttributes operation..

Note

Setting permissions for `SendMessage`, `DeleteMessage`, or `ChangeMessageVisibility` also sets permissions for the corresponding batch versions of those actions: `SendMessageBatch`, `DeleteMessageBatch`, and `ChangeMessageVisibilityBatch`.

Permissions for each of the different permission types are considered separate permissions by Amazon SQS, even though `*` includes the access provided by the other permission types. 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 applies when you remove a permission. If a principal has only a `*` permission, requesting to remove a `SendMessage` permission does not leave the principal with an "everything but" permission. Instead, the request does nothing, because the principal did not previously possess an explicit `SendMessage` permission.

If you want to remove `*` and leave the principal with just the `ReceiveMessage` permission, first add the `ReceiveMessage` permission, then remove the `*` permission.

Tip

You give each permission a label that identifies that permission. If you want to delete that permission in the future, you use that label to identify the permission.

Note

If you want to see what permissions are on a queue, use the GetQueueAttributes operation. The entire policy (containing all the permissions) is returned.

Granting Anonymous Access to a Queue

You can allow shared queue access to anonymous users. Such access requires no signature or Access Key ID.

To allow anonymous access you must write your own policy, setting the `Principal` to `*`. For information about writing your own policies, see Using The Access Policy Language (p. 41).

Caution

Keep in mind that the queue owner is responsible for all costs related to the queue. Therefore you probably want to limit anonymous access in some other way (by time or IP address, for example).

Programming Languages

AWS provides libraries, sample code, tutorials, and other resources for software developers who prefer to build applications using language-specific APIs instead of Amazon SQS's Query API. These libraries provide basic functions (not included in Amazon SQS's Query API), such as request authentication, request retries, and error handling so that it's easier to get started. Libraries and resources are available for the following languages:

- Java
- PHP
- Ruby
- Windows and .NET

For libraries and sample code in all languages, go to Sample Code & Libraries.

Using The Access Policy Language

Topics

This section is for Amazon SQS users who want to write their own access control policies. You don't need to write your own policies if you want to allow access based only on AWS account ID and basic permissions (e.g., SendMessage, ReceiveMessage). In that case, you can just use the Amazon SQS `AddPermission` action. If you want to explicitly deny access or allow it based on finer conditions (such as the time the request comes in or the IP address of the requester), you need to write your own policies and upload them to the AWS system using the SQS `SetQueueAttributes` action.

> **Note**
>
> To write your own policies, you must be familiar with JSON. For more information, go to http://json.org.

The main portion of this section includes basic concepts you need to understand, how to write a policy, and the logic AWS uses to evaluate policies and decide whether to give the requester access to the resource. Although most of the information in this section is service-agnostic, there are some SQS-specific details you need to know. For more information, see Special Information for SQS Policies (p. 70).

Overview

Topics

This section describes basic concepts you need to understand to use the access policy language to write policies. It also describes the general process for how access control works with the access policy language, and how policies are evaluated.

When to Use Access Control

You have a great deal of flexibility in how you grant or deny access to a resource. However, the typical use cases are fairly simple:

- You want to grant another AWS account a particular type of access to your queue (e.g., SendMessage). For more information, see Use Case 1 (p. 52).
- You want to grant another AWS account access to your queue for a specific period of time. For more information, see Use Case 2 (p. 52).
- You want to grant another AWS account access to your queue only if the requests come from your EC2 instances. For more information, see Use Case 3 (p. 53).
- You want to *deny* another AWS account access to your queue. For more information, see Use Case 4 (p. 53).

Key Concepts

The following sections describe the concepts you need to understand to use the access policy language. They're presented in a logical order, with the first terms you need to know at the top of the list.

Permission

A *permission* is the concept of allowing or disallowing some kind of access to a particular resource. Permissions essentially follow this form: "A is/isn't allowed to do B to C where D applies." For example, *Jane* (A) has permission to *receive messages* (B) from *John's Amazon SQS queue* (C), as long as *she asks to receive them before midnight on May 30, 2009* (D). Whenever Jane sends a request to Amazon SQS to use John's queue, the service checks to see if she has permission and if the request satisfies the conditions John set forth in the permission.

Statement

A *statement* is the formal description of a single permission, written in the access policy language. You always write a statement as part of a broader container document known as a *policy* (see the next concept).

Policy

A *policy* is a document (written in the access policy language) that acts as a container for one or more statements. For example, a policy could have two statements in it: one that states that Jane can use

John's queue, and another that states that Bob cannot use John's queue. As shown in the following figure, an equivalent scenario would be to have two policies, one containing the statement that Jane can use John's queue, and another containing the statement that Bob cannot use John's queue.

The AWS service implementing access control (e.g., Amazon SQS) uses the information in the statements (whether they're contained in a single policy or multiple) to determine if someone requesting access to a resource should be granted that access. We often use the term *policy* interchangeably with *statement*, as they generally represent the same concept (an entity that represents a permission).

Issuer

The *issuer* is the person who writes a policy to grant permissions for a resource. The issuer (by definition) is always the resource owner. AWS does not permit AWS service users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.

Principal

The *principal* is the person or persons who receive the permission in the policy. The principal is A in the statement "A has permission to do B to C where D applies." In a policy, you can set the principal to "anyone" (i.e., you can specify a wildcard to represent all people). You might do this, for example, if you don't want to restrict access based on the actual identity of the requester, but instead on some other identifying characteristic such as the requester's IP address.

Action

The *action* is the activity the principal has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." Typically, the action is just the operation in the request to AWS. For example, Jane sends a request to Amazon SQS with `Action=ReceiveMessage`. You can specify one or multiple actions in a policy.

Resource

The *resource* is the object the principal is requesting access to. The resource is C in the statement "A has permission to do B to C where D applies."

Conditions and Keys

The *conditions* are any restrictions or details about the permission. The condition is D in the statement "A has permission to do B to C where D applies." The part of the policy that specifies the conditions can be the most detailed and complex of all the parts. Typical conditions are related to:

- Date and time (e.g., the request must arrive before a specific day)
- IP address (e.g., the requester's IP address must be part of a particular CIDR range)

A *key* is the specific characteristic that is the basis for access restriction. For example, the date and time of request.

You use both *conditions* and *keys* together to express the restriction. The easiest way to understand how you actually implement a restriction is with an example: If you want to restrict access to before May 30, 2010, you use the condition called `DateLessThan`. You use the key called `AWS:CurrentTime` and set it to the value `2010-05-30T00:00:00Z`. AWS defines the conditions and keys you can use. The AWS service itself (e.g., Amazon SQS) might also define service-specific keys. For more information about conditions, see Condition (p. 59). For more information about the available keys, see Available Keys (p. 61).

Requester

The *requester* is the person who sends a request to an AWS service and asks for access to a particular resource. The requester sends a request to AWS that essentially says: "Will you allow me to do B to C where D applies?"

Evaluation

Evaluation is the process the AWS service uses to determine if an incoming request should be denied or allowed based on the applicable policies. For information about the evaluation logic, see Evaluation Logic (p. 48).

Effect

The *effect* is the result that you want a policy statement to return at evaluation time. You specify this value when you write the statements in a policy, and the possible values are *deny* and *allow*.

For example, you could write a policy that has a statement that *denies* all requests that come from Antarctica (effect=deny given that the request uses an IP address allocated to Antarctica). Alternately, you could write a policy that has a statement that *allows* all requests that *don't* come from Antarctica (effect=allow, given that the request doesn't come from Antarctica). Although the two statements sound like they do the same thing, in the access policy language logic, they are different. For more information, see Evaluation Logic (p. 48).

Although there are only two possible values you can specify for the effect (allow or deny), there can be three different results at policy evaluation time: *default deny*, *allow*, or *explicit deny*. For more information, see the following concepts and Evaluation Logic (p. 48).

Default Deny

A *default deny* is the default result from a policy in the absence of an allow or explicit deny.

Allow

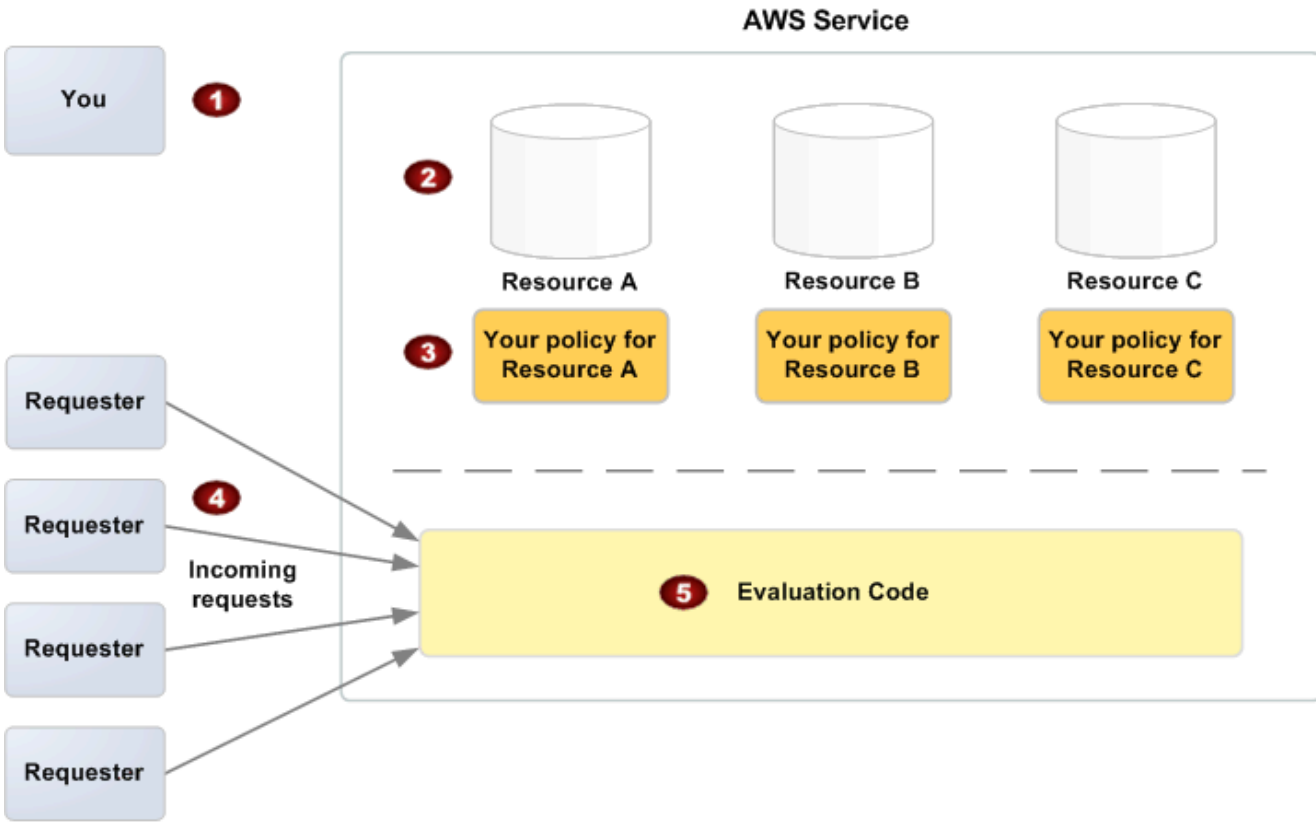
An *allow* results from a statement that has effect=allow, assuming any stated conditions are met. Example: Allow requests if they are received before 1:00 p.m. on April 30, 2010. An allow overrides all default denies, but never an explicit deny.

Explicit Deny

An *explicit deny* results from a statement that has effect=deny, assuming any stated conditions are met. Example: Deny all requests if they are from Antarctica. Any request that comes from Antarctica will always be denied no matter what any other policies might allow.

Architectural Overview

The following figure and table describe the main components that interact to provide access control for your resources.



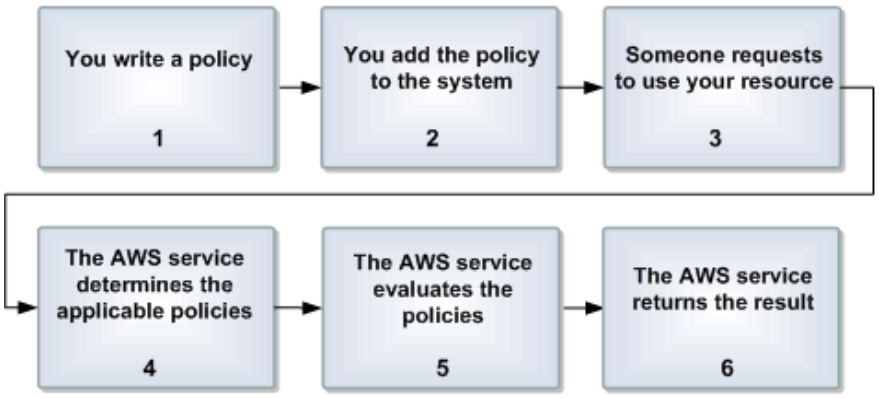
1	You, the resource owner.
2	Your resources (contained within the AWS service; e.g., SQS queues).
3	Your policies. Typically you have one policy per resource, although you could have multiple. The AWS service itself provides an API you use to upload and manage your policies. For information about the content of the policies, see How to Write a Policy (p. 55).
4	Requesters and their incoming requests to the AWS service.

5	The access policy language evaluation code. This is the set of code within the AWS service that evaluates incoming requests against the applicable policies and determines whether the requester is allowed access to the resource. For information about how the service makes the decision, see Evaluation Logic (p. 48).

For the typical process of how the components work together, see Using the Access Policy Language (p. 47).

Using the Access Policy Language

The following figure and table describe the general process of how access control works with the access policy language.



Process for Using Access Control with the Access Policy Language

1	You write a policy for your resource. For example, you write a policy to specify permissions for your Amazon SQS queues. For more information, see How to Write a Policy (p. 55).
2	You upload your policy to AWS. The AWS service itself provides an API you use to upload your policies. For example, you use the Amazon SQS `SetQueueAttributes` action to upload a policy for a particular Amazon SQS queue.
3	Someone sends a request to use your resource. For example, a user sends a request to SQS to use one of your queues.
4	The AWS service determines which policies are applicable to the request. For example, SQS looks at all the available SQS policies and determines which ones are applicable (based on what the resource is, who the requester is, etc.).
5	The AWS service evaluates the policies. For example, SQS evaluates the policies and determines if the requester is allowed to use your queue or not. For information about the decision logic, see Evaluation Logic (p. 48).
6	The AWS service either denies the request or continues to process it. For example, based on the policy evaluation result, the service either returns an "Access denied" error to the requester or continues to process the request.

Related Topics

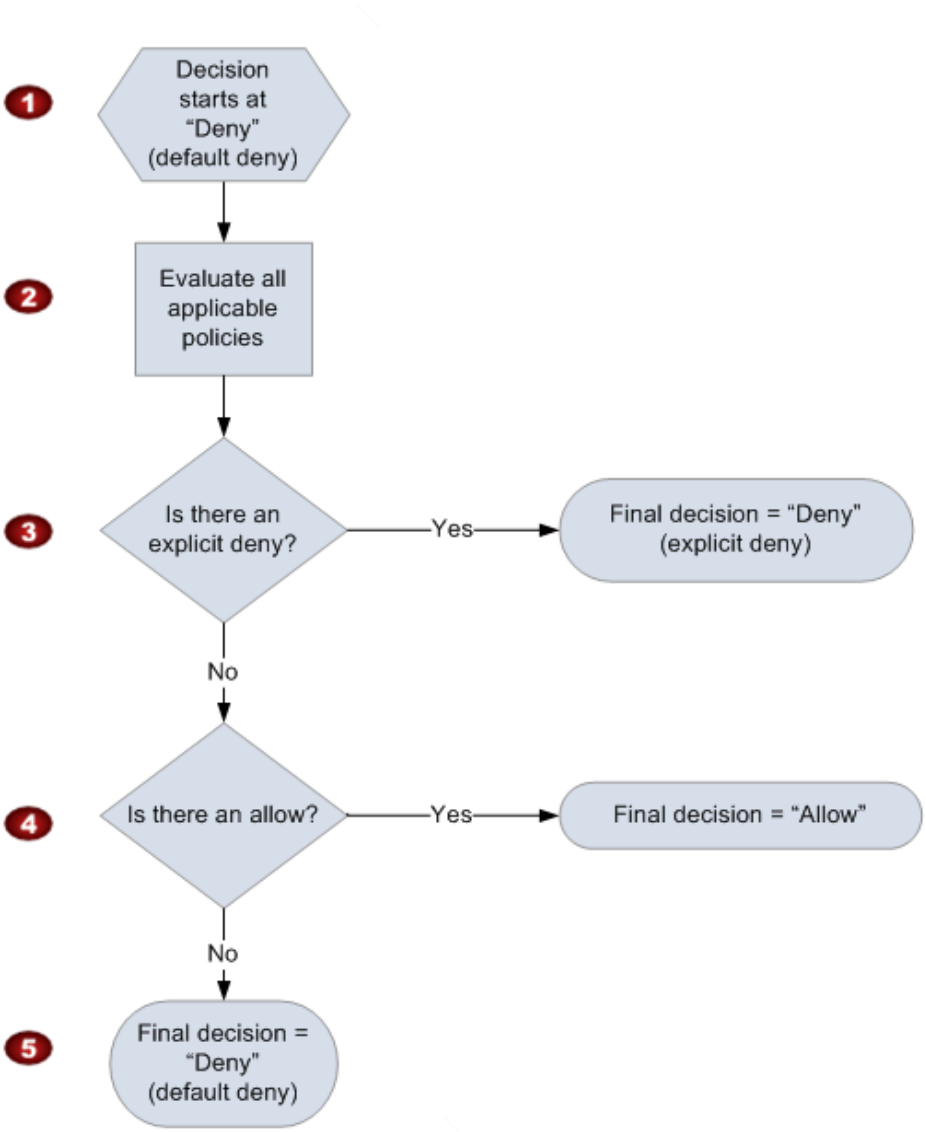
- Architectural Overview (p. 45)

Evaluation Logic

The goal at evaluation time is to decide whether a given request from someone other than you (the resource owner) should be allowed or denied. The evaluation logic follows several basic rules:

- By default, all requests to use your resource coming from anyone but you are denied
- An allow overrides any default denies
- An explicit deny overrides any allows
- The order in which the policies are evaluated is not important

The following flow chart and discussion describe in more detail how the decision is made.



1	The decision starts with a default deny.

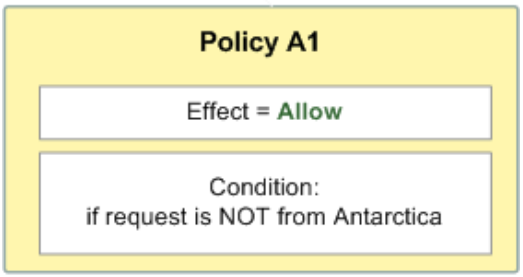
2	The enforcement code then evaluates all the policies that are applicable to the request (based on the resource, principal, action, and conditions). The order in which the enforcement code evaluates the policies is not important.
3	In all those policies, the enforcement code looks for an explicit deny instruction that would apply to the request. If it finds even one, the enforcement code returns a decision of "deny" and the process is finished (this is an explicit deny; for more information, see Explicit Deny (p. 45)).
4	If no explicit deny is found, the enforcement code looks for any "allow" instructions that would apply to the request. If it finds even one, the enforcement code returns a decision of "allow" and the process is done (the service continues to process the request).
5	If no allow is found, then the final decision is "deny" (because there was no explicit deny or allow, this is considered a *default deny* (for more information, see Default Deny (p. 44)).

The Interplay of Explicit and Default Denials

A policy results in a default deny if it doesn't directly apply to the request. For example, if a user requests to use Amazon SQS, but the only policy that applies to the user states that the user can use Amazon SimpleDB, then that policy results in a default deny.

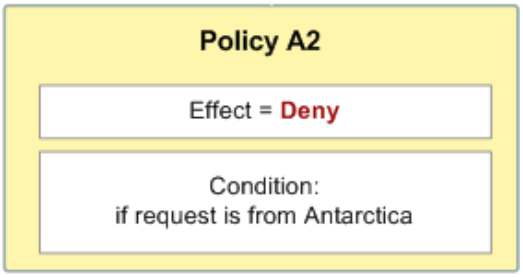
A policy also results in a default deny if a condition in a statement isn't met. If all conditions in the statement are met, then the policy results in either an allow or an explicit deny, based on the value of the Effect element in the policy. Policies don't specify what to do if a condition isn't met, and so the default result in that case is a default deny.

For example, let's say you want to prevent requests coming in from Antarctica. You write a policy (called Policy A1) that allows a request only if it doesn't come from Antarctica. The following diagram illustrates the policy.



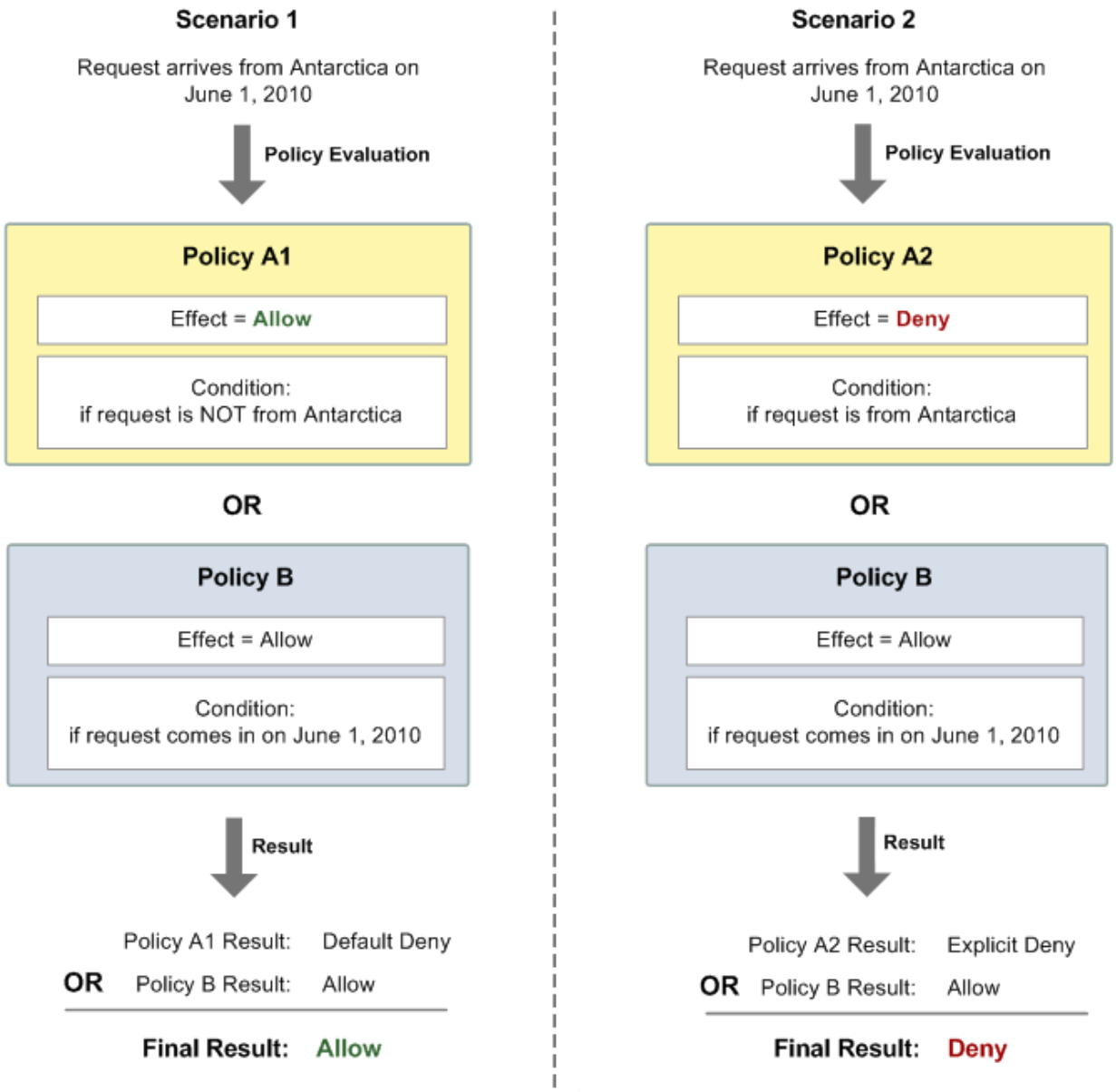
If someone sends a request from the U.S., the condition is met (the request is not from Antarctica). Therefore, the request is allowed. But, if someone sends a request from Antarctica, the condition isn't met, and the policy's result is therefore a default deny.

You could turn the result into an explicit deny by rewriting the policy (named Policy A2) as in the following diagram. Here, the policy explicitly denies a request if it comes from Antarctica.



If someone sends a request from Antarctica, the condition is met, and the policy's result is therefore an explicit deny.

The distinction between a default deny and an explicit deny is important because a default deny can be overridden by an allow, but an explicit deny can't. For example, let's say there's another policy that allows requests if they arrive on June 1, 2010. How does this policy affect the overall outcome when coupled with the policy restricting access from Antarctica? We'll compare the overall outcome when coupling the date-based policy (we'll call Policy B) with the preceding policies A1 and A2. Scenario 1 couples Policy A1 with Policy B, and Scenario 2 couples Policy A2 with Policy B. The following figure and discussion show the results when a request comes in from Antarctica on June 1, 2010.



In Scenario 1, Policy A1 returns a default deny, as described earlier in this section. Policy B returns an allow because the policy (by definition) 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 therefore allowed.

In Scenario 2, Policy B2 returns an explicit deny, as described earlier in this section. Again, Policy B returns an allow. The explicit deny from Policy A2 overrides the allow from Policy B, and the request is therefore denied.

Basic Use Cases for Access Control

This section gives a few examples of typical use cases for access control.

Use Case 1

Let's say you have a set of queues in the Amazon SQS system. In the simplest case, you want to allow one or more AWS accounts a particular type of access to a queue (e.g., SendMessage, ReceiveMessage).

You can do this by simply using the Amazon SQS API action `AddPermission`. It takes a few input parameters and automatically creates a policy in the SQS system for that queue. For this use case, you don't need to read this appendix or learn how to write a policy yourself, because SQS can automatically create the policy for you.

The following example shows a policy that gives AWS account ID 1111-2222-3333 permission to send and receive from a queue you own named queue2. In this example, your AWS account ID is 4444-5555-6666.

```
{
   "Version":"2008-10-17",
   "Id":"UseCase1",
   "Statement" : [
      {
         "Sid":"1",
         "Effect":"Allow",
         "Principal" : {
            "AWS": "111122223333"
         },
         "Action":["sqs:SendMessage","sqs:ReceiveMessage"],
         "Resource": "arn:aws:sqs:us-east-1:444455556666:queue2",
      }
   ]
}
```

Use Case 2

In this use case, you want to allow one or more AWS accounts access to your queues *only for a specific time period*.

You need to know how to write your own policy for the queue because the SQS `AddPermission` action doesn't let you specify a time restriction when granting someone access to your queue. In this case, you would write your own policy and then upload it to the AWS system with the `SetQueueAttributes` action. Effectively the action sets your policy as an attribute of the queue.

The following example is the same as in use case 1, except it also includes a condition that restricts access to before June 30, 2009, at noon (UTC).

```
{
   "Version":"2008-10-17",
   "Id":"UseCase2",
   "Statement" : [
      {
         "Sid":"1",
         "Effect":"Allow",
         "Principal" : {
            "AWS": "111122223333"
         },
         "Action":["sqs:SendMessage","sqs:ReceiveMessage"],
         "Resource": "arn:aws:sqs:us-east-1:444455556666:queue2",
         "Condition" : {
```

```
            "DateLessThan" : {
               "AWS:CurrentTime":"2009-06-30T12:00Z"
            }
         }
      }
   ]
}
```

Use Case 3

In this use case, you want to allow access to your queues *only if the requests come from your Amazon EC2 instances*.

Again, you need to know how to write your own policy because the SQS `AddPermission` action doesn't let you specify an IP address restriction when granting access to your queue.

The following example builds on the example in use case 2, and also includes a condition that restricts access to the IP address range 10.52.176.0/24. So in this example, a request from AWS account 1234-5678-9012 to send or receive messages from queue2 would be allowed only if it came in before noon on June 30, 2009, *and* it came from the 10.52.176.0/24 address range.

```
{
   "Version":"2008-10-17",
   "Id":"UseCase3",
   "Statement" : [
      {
         "Sid":"1",
         "Effect":"Allow",
         "Principal" : {
            "AWS": "111122223333"
         },
         "Action":["sqs:SendMessage","sqs:ReceiveMessage"],
         "Resource": "arn:aws:sqs:us-east-1:444455556666:queue2",
         "Condition" : {
            "DateLessThan" : {
               "AWS:CurrentTime":"2009-06-30T12:00Z"
            },
            "IpAddress" : {
               "AWS:SourceIp":"10.52.176.0/24"
            }
         }
      }
   ]
}
```

Use Case 4

In this use case, you want to specifically *deny* a certain AWS account access to your queues.

Again, you need to know how to write your own policy because the SQS `AddPermission` action doesn't let you *deny* access to a queue; it only lets you *grant* access.

The following example is the same as in the original use case (#1), except it *denies* access to the specified AWS account.

```
{
   "Version":"2008-10-17",
   "Id":"UseCase4",
   "Statement" : [
      {
         "Sid":"1",
         "Effect":"Deny",
         "Principal" : {
            "AWS": "111122223333"
         },
         "Action":["sqs:SendMessage","sqs:ReceiveMessage"],
         "Resource": "arn:aws:sqs:us-east-1:444455556666:queue2",
      }
   ]
}
```

From these use cases, you can see that if you want to restrict access based on special conditions or deny someone access entirely, you need to read this appendix and learn how to write your own policies. You can also see that the policies themselves are not that complex and the access policy language is straightforward.

How to Write a Policy

Topics

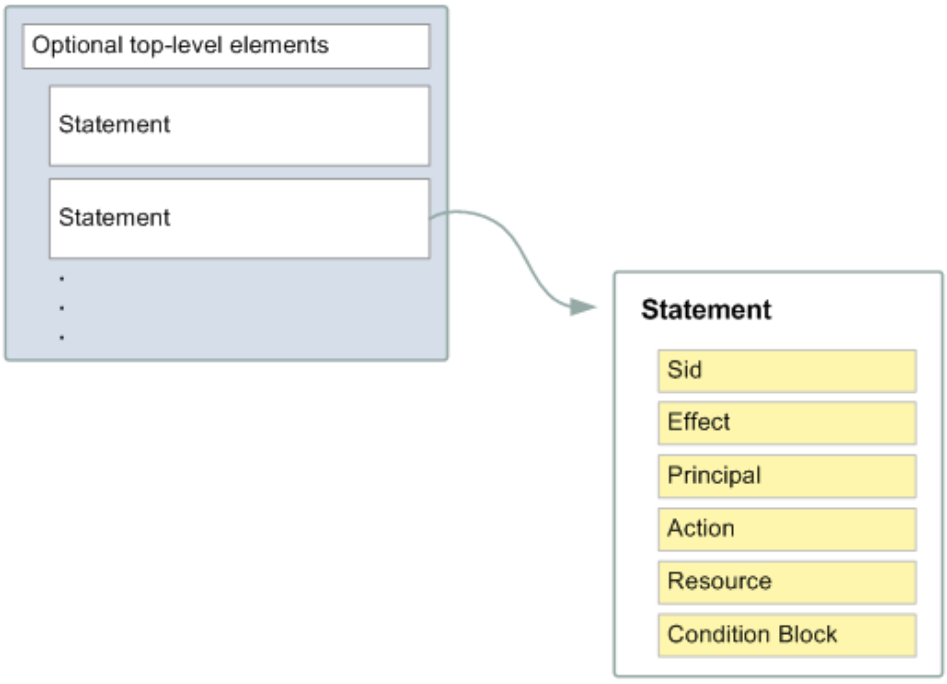
This section describes how to write policies and gives reference information about each policy element.

Basic Policy Structure

Each policy is a JSON document. As illustrated in the following figure, a policy includes:

- Optional policy-wide information (at the top of the document)
- One or more individual *statements*

Each statement includes the core information about a single permission. If a policy includes multiple statements, we apply a logical `OR` across the statements at evaluation time. If multiple policies are applicable to a request, we apply a logical `OR` across the policies at evaluation time.



The information in a statement is contained within a series of *elements*. For information about these elements, see Element Descriptions (p. 56).

Example

The following simple policy allows an AWS developer with account ID 111122223333 to send and read from the Amazon SQS queue named queue2 (owned by the developer with account ID 444455556666) in the US East (Northern Virginia) Region, given that the request comes from the 10.52.176.0/24 address range, and the request comes in before noon on June 30, 2009 (UTC).

```
{
   "Version":"2008-10-17",
   "Id":"cd3ad3d9-2776-4ef1-a904-4c229d1642ee",
   "Statement" : [
      {
         "Sid":"1",
         "Effect":"Allow",
         "Principal" : {
            "aws": "111122223333"
         },
         "Action":["sqs:SendMessage","sqs:ReceiveMessage"],
         "Resource": "arn:aws:sqs:us-east-1:444455556666:queue2",
         "Condition" : {
            "IpAddress" : {
               "aws:SourceIp":"10.52.176.0/24"
            },
             "DateLessThan" : {
               "aws:CurrentTime":"2009-06-30T12:00Z"
            }
         }
      }
   ]
}
```

Element Descriptions

Topics

This section describes the elements you can use in a policy and its statements. The elements are listed here in the general order you use them in a policy. The `Id`, `Version`, and `Statement` are top-level policy elements; the rest are statement-level elements. JSON examples are provided.

All elements are optional for the purposes of parsing the policy document itself. The order of the elements doesn't matter (e.g., the `Resource` element can come before the `Action` element). You're not required to specify any Conditions in the policy.

Version

The `Version` is the access policy language version. This is an optional element, and currently the only allowed value is `2008-10-17`.

```
"Version":"2008-10-17"
```

Id

The `Id` is an optional identifier for the policy. We recommend you use a UUID for the value, or incorporate a UUID as part of the ID to ensure uniqueness.

Important

The AWS service (e.g., SQS or Amazon SNS) implementing the access policy language might require this element and have uniqueness requirements for it. For service-specific information about writing policies, see Special Information for SQS Policies (p. 70).

```
"Id":"cd3ad3d9-2776-4ef1-a904-4c229d1642ee"
```

Statement

The `Statement` is the main element for a statement. It can include multiple elements (see the subsequent sections in this guide).

The `Statement` element contains an array of individual statements. Each individual statement is a distinct JSON block enclosed in curly brackets { }.

```
"Statement":[{...},{...},{...}]
```

Sid

The `Sid` (statement ID) is an optional identifier you provide for the policy statement. Essentially it is just a sub-ID of the policy document's ID.

Important

The AWS service (e.g., SQS or Amazon SNS) implementing the access policy language might require this element and have uniqueness requirements for it. For service-specific information about writing policies, see Special Information for SQS Policies (p. 70).

```
"Sid" : "1"
```

Effect

The `Effect` is a required element that indicates whether you want the statement to result in an allow or an explicit deny (for more information, see Explicit Deny (p. 45)).

Valid values for `Effect` are `Allow` and `Deny`.

```
"Effect":"Allow"
```

Principal

The `Principal` is the person or persons who receive or are denied permission according to the policy. You must specify the principal by using the principal's AWS account ID (e.g., 1234-5678-9012, with or without the hyphens). You can specify multiple principals, or a wildcard (*) to indicate all possible users. You can view your account ID by logging in to your AWS account at http://aws.amazon.com and clicking **Account Activity**.

In JSON, you use `"AWS":` as a prefix for the principal's AWS account ID. In the following example, two principals are included in the statement.

```
"Principal":[
   "AWS": "123456789012",
   "AWS": "999999999999"
]
```

NotPrincipal

The `NotPrincipal` element is useful if you want to make an exception to a list of principals. You could use this, for example, if you want to prevent all AWS accounts except a certain one. The `Principal` is the person or persons who receive or are denied permission according to the policy. You must specify the principal by using the principal's AWS account ID (e.g., 1234-5678-9012, with or without the hyphens). You can specify multiple principals, or a wildcard (*) to indicate all possible users. You can view your account ID by logging in to your AWS account at http://aws.amazon.com and clicking **Account Activity**.

In JSON, you use `"AWS":` as a prefix for the principal's AWS account ID. In the following example, two principals are included in the statement.

```
"Principal":[
   "AWS": "123456789012",
   "AWS": "999999999999"
   ]
```

Action

The `Action` is the specific type or types of access allowed or denied (for example, read or write). You can specify multiple values for this element. The values are free-form but must match values the AWS service expects (for more information, see Special Information for SQS Policies (p. 70)). You can use a wildcard (*) to give the principal access to all the actions the specific AWS service lets you share with other developers. For example, Amazon SQS lets you share only a particular subset of all the possible SQS actions. So, using the wildcard doesn't give someone full control of the queue; it only gives access to that particular subset of actions.

```
"Action":["sqs:SendMessage","sqs:ReceiveMessage"]
```

The prefix and the action name are case insensitive. For example, `sqs:SendMessage` is equivalent to `SQS:sendmessage`.

NotAction

The `NotAction` element is useful if you want to make an exception to a list of actions. You could use this, for example, if you want your users to be able to use only the SQS `SendMessage`.

The following example refers to all actions *other* than the SQS `SendMessage`. You would use this in a policy with `"Effect":"Deny"` to keep users from accessing any other actions.

The `NotAction` element matches everything except the specified action. This is useful if you want to make an exception to a list of actions being allowed or denied. The example below matches any action, except `Publish`.

```
"NotAction":"sqs:SendMessage"
```

Resource

The `Resource` is the object or objects the policy covers. The value can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. The values are free-form, but must follow the format the AWS service expects. For example, for Amazon SQS, you specify a queue in the following format: `<account ID of queue owner>:<queue name>`. For example: `111122223333:queue1`.

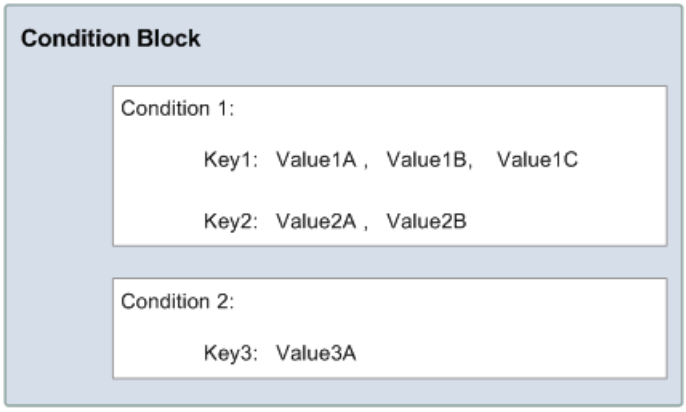
```
"Resource":"arn:aws:sqs:us-east-1:111122223333:queue1;"
```

Condition

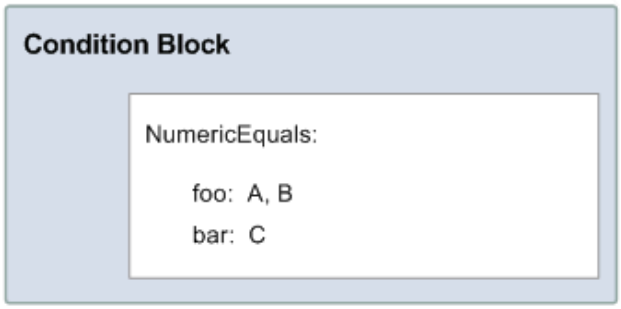
This section describes the `Condition` element and the information you can use inside the element.

The Condition Block

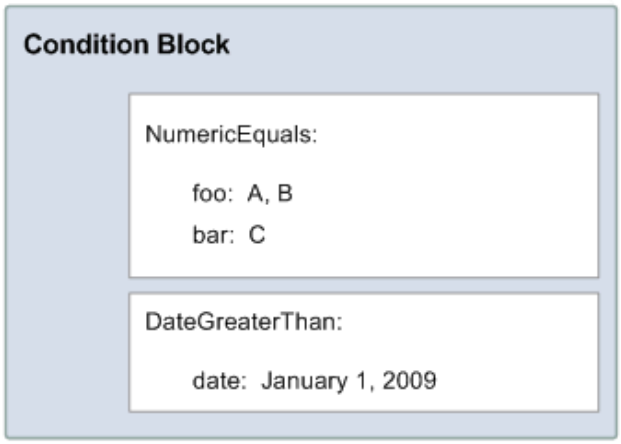
The `Condition` element is the most complex part of the policy statement. We refer to it as the *condition block*, because although it has a single `Condition` element, it can contain multiple conditions, and each condition can contain multiple key-value pairs. The following figure illustrates this. Unless otherwise specified for a particular key, all keys can have multiple values.



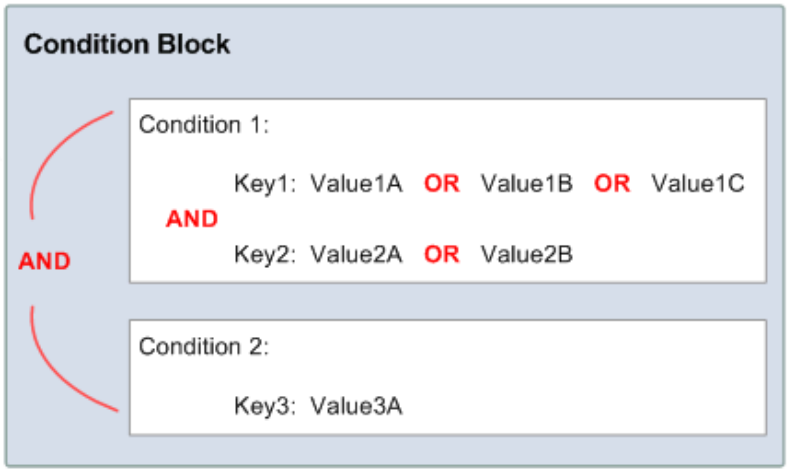
When creating a condition block, you specify the name of each condition, and at least one key-value pair for each condition. AWS defines the conditions and keys you can use (they're listed in the subsequent sections). An example of a condition is `NumericEquals`. Let's say you have a fictional resource, and you want to let John use it only if some particular numeric value *foo* equals either A or B, and another numeric value *bar* equals C. Then you would create a condition block that looks like the following figure.



Let's say you also want to restrict John's access to after January 1, 2009. Then you would add another condition, `DateGreaterThan`, with a date equal to January 1, 2009. The condition block would then look like the following figure.



As illustrated in the following figure, we always apply a logical `AND` to the conditions within a condition block, and to the keys within a condition. We always apply a logical `OR` to the values for a single key. All conditions must be met to return an allow or an explicit deny decision. If a condition isn't met, the result is a default deny.



As mentioned, AWS defines the conditions and keys you can use (for example, one of the keys is `aws:CurrentTime`, which lets you restrict access based on the date and time). The AWS service itself can also define its own service-specific keys. For a list of available keys, see Available Keys (p. 61).

For a concrete example that uses real keys, let's say you want to let John access your Amazon SQS queue under the following three conditions:

- The time is after 12:00 noon on 8/16/2010
- The time is before 3:00 p.m. on 8/16/2010
- The request comes from an IP address within the 192.168.176.0/24 range or the 192.168.143.0/24 range

Your condition block has three separate conditions, and all three of them must be met for John to have access to your queue.

The following shows what the condition block looks like in your policy.

```
"Condition" :  {
      "DateGreaterThan" : {
         "aws:CurrentTime" : "2009-04-16T12:00:00Z"
       },
      "DateLessThan": {
         "aws:CurrentTime" : "2009-04-16T15:00:00Z"
       },
       "IpAddress" : {
          "aws:SourceIp" : ["192.168.176.0/24","192.168.143.0/24"]
      }
}
```

Available Keys

AWS provides a set of common keys supported by all AWS services that adopt the access policy language for access control. These keys are:

- `aws:CurrentTime`–For date/time conditions (see Date Conditions (p. 63))
- `aws:EpochTime`–The date in epoch or UNIX time, for use with date/time conditions (see Date Conditions (p. 63))
- `aws:MultiFactorAuthAge`–Key that provides a numeric value indicating how long ago (in seconds) the MFA-validated security credentials making the request were issued using Multi-Factor Authentication (MFA). Unlike other keys, if MFA is not used successfully, this key is not present (see Existence of Condition Keys (p. 64), Numeric Conditions (p. 62) and Using Multi-Factor Authentication (MFA) Devices with AWS).
- `aws:SecureTransport`–Boolean representing whether the request was sent using SSL (see Boolean Conditions (p. 64))
- `aws:SourceArn`–The Amazon Resource Name (ARN) of the source (see Amazon Resource Name (ARN) (p. 64))
- `aws:SourceIp`–The requester's IP address, for use with IP address conditions (see IP Address (p. 64))
- `aws:UserAgent`–Information about the requester's client application, for use with string conditions (see String Conditions (p. 62))

The key names are case insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

Note

If you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, we evaluate the instance's public IP address to determine if access is allowed.

Each AWS service that uses the access policy language might also provide service-specific keys. For a list of any service-specific keys you can use, see Special Information for SQS Policies (p. 70).

Condition Types

These are the general types of conditions you can specify:

- String
- Numeric
- Date and time
- Boolean
- IP address
- Amazon Resource Name (ARN)
- Existence of condition keys

String Conditions

String conditions let you constrain using string matching rules. The actual data type you use is a string.

Condition	Description
`StringEquals`	Strict matching Short version: `streq`
`StringNotEquals`	Strict negated matching Short version: `strneq`
`StringEqualsIgnoreCase`	Strict matching, ignoring case Short version: `streqi`
`StringNotEqualsIgnoreCase`	Strict negated matching, ignoring case Short version: `strneqi`
`StringLike`	Loose case-sensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: `strl`
`StringNotLike`	Negated loose case-insensitive matching. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string. Short version: `strnl`

Numeric Conditions

Numeric conditions let you constrain using numeric matching rules. You can use both whole integers or decimal numbers. Fractional or irrational syntax is not supported.

Condition	Description
`NumericEquals`	Strict matching Short version: `numeq`
`NumericNotEquals`	Strict negated matching Short version: `numneq`
`NumericLessThan`	"Less than" matching Short version: `numlt`
`NumericLessThanEquals`	"Less than or equals" matching Short version: `numlteq`
`NumericGreaterThan`	"Greater than" matching Short version: `numgt`
`NumericGreaterThanEquals`	"Greater than or equals" matching Short version: `numgteq`

Date Conditions

Date conditions let you constrain using date and time matching rules. You must specify all date/time values with one of the W3C implementations of the ISO 8601 date formats (for more information, go to http://www.w3.org/TR/NOTE-datetime). You use these conditions with the `aws:CurrentTime` key to restrict access based on request time.

Note

Wildcards are not permitted for date conditions.

Condition	Description
`DateEquals`	Strict matching Short version: `dateeq`
`DateNotEquals`	Strict negated matching Short version: `dateneq`
`DateLessThan`	A point in time at which a key stops taking effect Short version: `datelt`
`DateLessThanEquals`	A point in time at which a key stops taking effect Short version: `datelteq`
`DateGreaterThan`	A point in time at which a key starts taking effect Short version: `dategt`
`DateGreaterThanEquals`	A point in time at which a key starts taking effect Short version: `dategteq`

Boolean Conditions

Condition	Description
`Bool`	Strict Boolean matching

IP Address

IP address conditions let you constrain based on IP address matching rules. You use these with the `aws:SourceIp` key. The value must be in the standard CIDR format (for example, 10.52.176.0/24). For more information, go to RFC 4632.

Condition	Description
`IpAddress`	Approval based on the IP address or range
`NotIpAddress`	Denial based on the IP address or range

Amazon Resource Name (ARN)

Amazon Resource Name (ARN) conditions let you constrain based on ARN matching rules. The actual data type you use is a string.

Condition	Description
`ArnEquals`	Strict matching for ARN
`ArnNotEquals`	Strict negated matching for ARN
`ArnLike`	Loose case-insensitive matching of the ARN. Each of the six colon-delimited components of the ARN is checked separately and each can include a multi-character match wildcard (*) or a single-character match wildcard (?).
`ArnNotLike`	Negated loose case-insensitive matching of the ARN. The values can include a multi-character match wildcard (*) or a single-character match wildcard (?) anywhere in the string.

Existence of Condition Keys

Use a `Null` condition to check if a condition key is present at the time of authorization. In the policy statement, use either `true` (the key doesn't exist) or `false` (the key exists and its value is not null). You can use this condition to determine if a user has authenticated with MFA (Multi-Factor Authentication). For example, the following condition states that MFA authentication must exist (be *not null*) for the user to use the Amazon EC2 API.

```
{
  "Statement":[{
      "Action":["ec2:*"],
      "Effect":"Allow",
      "Resource":["*"],
      "Condition":{
         "Null":{"aws:MultiFactorAuthAge":"false"}
      }
```

```
    }
  ]
}
```

Supported Data Types

This section lists the set of data types the access policy language supports. The language doesn't support all types for each policy element (for the supported data types for each element, see Element Descriptions (p. 56)).

The access policy language supports the following data types:

- Strings
- Numbers (Ints and Floats)
- Boolean
- Null
- Lists
- Maps
- Structs (which are just nested Maps)

The following table maps each data type to the serialization. Note that all policies must be in UTF-8. For information about the JSON data types, go to RFC 4627.

Type	JSON
String	String
Integer	Number
Float	Number
Boolean	true false
Null	null
Date	String adhering to the W3C Profile of ISO 8601
IpAddress	String adhering to RFC 4632
List	Array
Object	Object

Amazon SQS Policy Examples

This section shows example policies for common Amazon SQS use cases.

The following example policy gives the developer with AWS account number 111122223333 the SendMessage permission for the queue named 444455556666/queue1 in the US East (Northern Virginia) Region.

```
{
  "Version": "2008-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement":
    {
       "Sid":"Queue1_SendMessage",
       "Effect": "Allow",
       "Principal": {
            "AWS": "111122223333"
        },
        "Action": "sqs:SendMessage",
        "Resource": "arn:aws:sqs:us-east-1:444455556666:queue1"
    }
}
```

The following example policy gives the developer with AWS account number 111122223333 both the SendMessage and ReceiveMessage permission for the queue named 444455556666/queue1 in the US East (Northern Virginia) Region.

```
{
  "Version": "2008-10-17",
  "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"
    }
}
```

The following example policy gives two different developers (with AWS account numbers 111122223333 and 444455556666) permission to use all actions that SQS allows shared access for the queue named 123456789012/queue1 in the US East (Northern Virginia) Region.

```
{
  "Version": "2008-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement":
    {
       "Sid":"Queue1_AllActions",
       "Effect": "Allow",
       "Principal": {
          "AWS": ["111122223333","444455556666"]
```

```
        },
       "Action": "sqs:*",
       "Resource": "arn:aws:sqs:us-east-1:123456789012:queue1"
    }
}
```

The following example policy gives all users ReceiveMessage permission for the queue named 111122223333/queue1.

```
{
  "Version": "2008-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement":
    {
       "Sid":"Queue1_AnonymousAccess_ReceiveMessage",
       "Effect": "Allow",
       "Principal": {
          "AWS": "*"
        },
       "Action": "sqs:ReceiveMessage",
       "Resource": "arn:aws:sqs:*:111122223333:queue1"
    }
}
```

The following example policy gives all users ReceiveMessage permission for the queue named 111122223333/queue1, but only between noon and 3:00 p.m. on January 31, 2009.

```
{
  "Version": "2008-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement":
     {
        "Sid":"Queue1_AnonymousAccess_ReceiveMessage_TimeLimit",
        "Effect": "Allow",
        "Principal": {
           "AWS": "*"
        },
        "Action": "sqs:ReceiveMessage",
        "Resource": "arn:aws:sqs:*:111122223333:queue1",
        "Condition" : {
           "DateGreaterThan" : {
              "aws:CurrentTime":"2009-01-31T12:00Z"
           },
           "DateLessThan" : {
              "aws:CurrentTime":"2009-01-31T15:00Z"
           }
        }
     }
}
```

The following example policy gives all users permission to use all possible 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 range.

```
{
  "Version": "2008-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement":
     {
        "Sid":"Queue1_AnonymousAccess_AllActions_WhitelistIP",
        "Effect": "Allow",
        "Principal": {
           "AWS": "*"
         },
        "Action": "sqs:*",
        "Resource": "arn:aws:sqs:*:111122223333:queue1",
        "Condition" : {
            "IpAddress" : {
            "aws:SourceIp":"192.168.143.0/24"
            }
        }
     }
}
```

The following example policy has two statements:

- One that gives all users in the 192.168.143.0/24 range (except for 192.168.143.188) permission to use the SendMessage action for the queue named 111122223333/queue1.
- One that blacklists all users in the 10.1.2.0/24 range from using the queue.

```
{
  "Version": "2008-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [
     {
         "Sid":"Queue1_AnonymousAccess_SendMessage_IPLimit",
         "Effect": "Allow",
         "Principal": {
            "AWS": "*"
         },
         "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": {
            "AWS": "*"
         },
         "Action": "sqs:*",
         "Resource": "arn:aws:sqs:*:111122223333:queue1",
         "Condition" : {
```

```
            "IpAddress" : {
            "aws:SourceIp":"10.1.2.0/24"
            }
         }
      }
  ]
  }
```

The following example policy enables a connection between the Amazon Simple Notification Service topic specified by the Amazon Resource Name (ARN) arn:aws:sns:us-east-1:111122223333:test-topic and the queue named arn:aws:sqs:us-east-1:111122223333:test-topic-queue.

```
{
  "Version": "2008-10-17",
  "Id": "SNStoSQS",
  "Statement":
     {
        "Sid":"rule1",
        "Effect": "Allow",
        "Principal": {
           "AWS": "*"
         },
        "Action": "sqs:*",
        "Resource": "arn:aws:sqs:us-east-1:111122223333:test-topic-queue",
        "Condition" : {
            "StringEquals" : {
            "aws:SourceArn":"arn:aws:sns:us-east-1:111122223333:test-topic"
            }
        }
     }
}
```

Special Information for SQS Policies

The following list gives information specific to the SQS implementation of access control.

- SQS allows you to share only certain types of permissions (for more information, see Understanding Permissions (p. 37))
- Each policy must cover only a single queue (when writing a policy, don't include statements that cover different queues)
- Each policy must have a unique policy ID (`Id`)
- Each statement in a policy must have a unique statement ID (`sid`)
- SQS does not implement any special keys to use when you write conditions; the only keys available are the general AWS-wide keys.

The following table lists the maximum limits for policy information.

Name	Maximum Limit
Bytes	8192
Statements	20
Principals	50
Conditions	10

Controlling User Access to Your AWS Account

Topics

Amazon SQS has its own resource-based permissions system that uses policies written in the same language used for AWS Identity and Access Management (AWS IAM) policies. This means that you can achieve the same things with SQS policies that you can with AWS IAM policies. The main difference between using SQS policies versus AWS IAM policies is that you can grant another AWS Account permission to your queues with an SQS policy, and you can't do that with an AWS IAM policy.

> **Note**
>
> When you grant other AWS accounts access to your AWS resources, be aware that all AWS accounts can delegate their permissions to users under their accounts. This is known as cross-account access. Cross-account access enables you to share access to your AWS resources without having to manage additional users. For information about using cross-account access, go to Enabling Cross-Account Access in *Using AWS Identity and Access Management*.

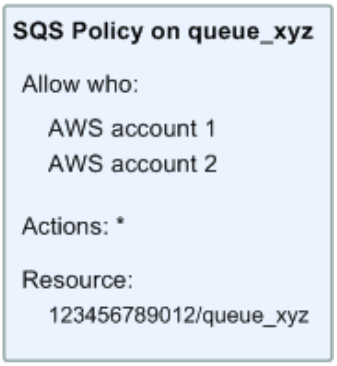
This section describes how the SQS policy system works with AWS IAM.

IAM-Related Features of SQS Policies

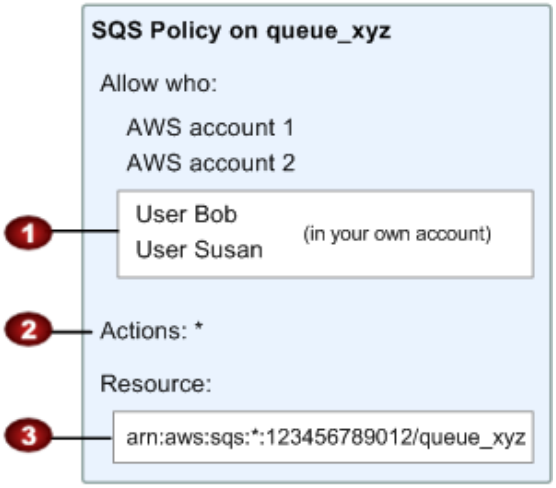
You can use an SQS policy with a queue to specify which AWS Accounts have access to the queue. You can specify the type of access and conditions (e.g., permission to use `SendMessage`, `ReceiveMessage`, if the request is before December 31, 2010). The specific actions you can grant permission for are a

subset of the overall list of SQS actions. When you write an SQS policy and specify * to mean "all the SQS actions", that means all actions in that subset.

The following diagram illustrates the concept of one of these basic SQS policies that covers the subset of actions. The policy is for queue_xyz, and it gives AWS Account 1 and AWS Account 2 permission to use any of the allowed actions with the queue. Notice that the resource in the policy is specified as 123456789012/queue_xyz (where 123456789012 is the AWS Account ID of the account that owns the queue).



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



1	In addition to specifying which AWS Accounts have access to the queue, you can specify which Users *in your own AWS Account* have access to the queue. The Users can't be in another AWS Account.
2	The subset of actions included in "*" has expanded (for a list of allowed actions, see Amazon SQS Actions (p. 76)).

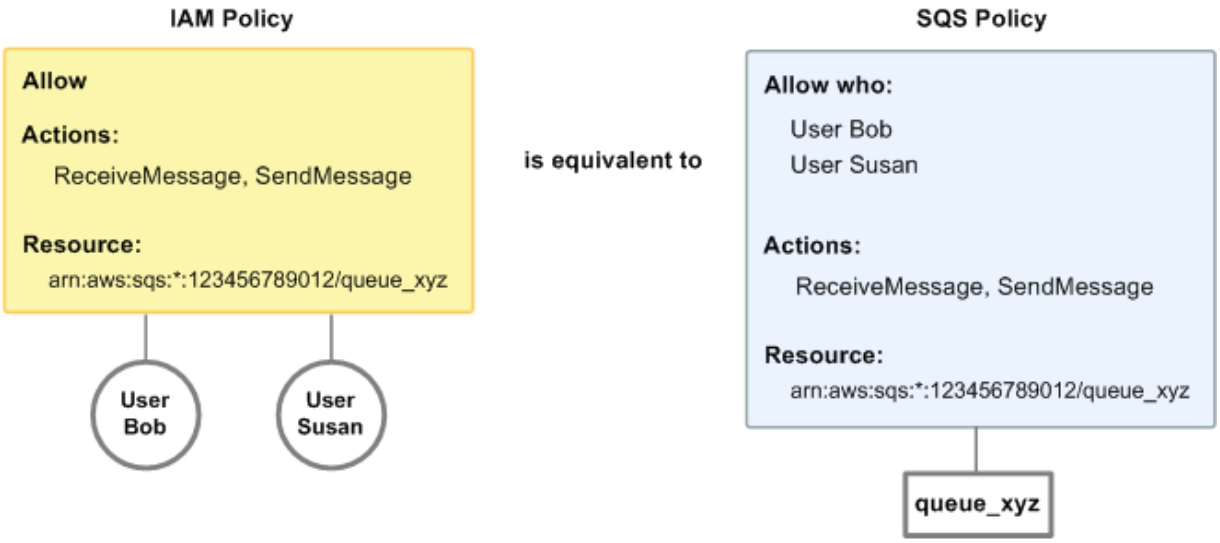
3	You can specify the resource using the *Amazon Resource Name (ARN)*, which is how you must specify resources in AWS IAM policies. For information about the ARN format for SQS queues, see Amazon SQS ARNs (p. 75). You can still use the original format instead (<account_ID>/<queue_name>).

So for example, according to the SQS policy shown in the preceding figure, anyone possessing the root account's security credentials for AWS Account 1 or AWS Account 2 could access queue_xyz. Also, Users Bob and Susan in your own AWS Account (with ID 123456789012) can access the queue.

Also, before the introduction of AWS IAM, SQS automatically gave the creator of a queue full control over the queue (e.g., access to all possible SQS actions with that queue). This is no longer true, unless the creator is using the AWS Account's credentials. Any User who has permission to create a queue must also have permission to use other SQS actions in order to do anything with the queues they create.

AWS IAM and SQS Policies Together

There are two ways you can give your Users permissions for your SQS resources: through the SQS policy system or the AWS IAM policy system. You can use one or the other, or both. For the most part, you can achieve the same results with either. For example, the following diagram shows an IAM policy and an SQS policy that are equivalent. The IAM policy allows the SQS `ReceiveMessage` and `SendMessage` actions for the queue called queue_xyz in your AWS Account, and it's attached to the Users Bob and Susan (which means Bob and Susan have the permissions stated in the policy). The SQS policy also gives Bob and Susan permission to access `ReceiveMessage` and `SendMessage` for the same queue.



Note

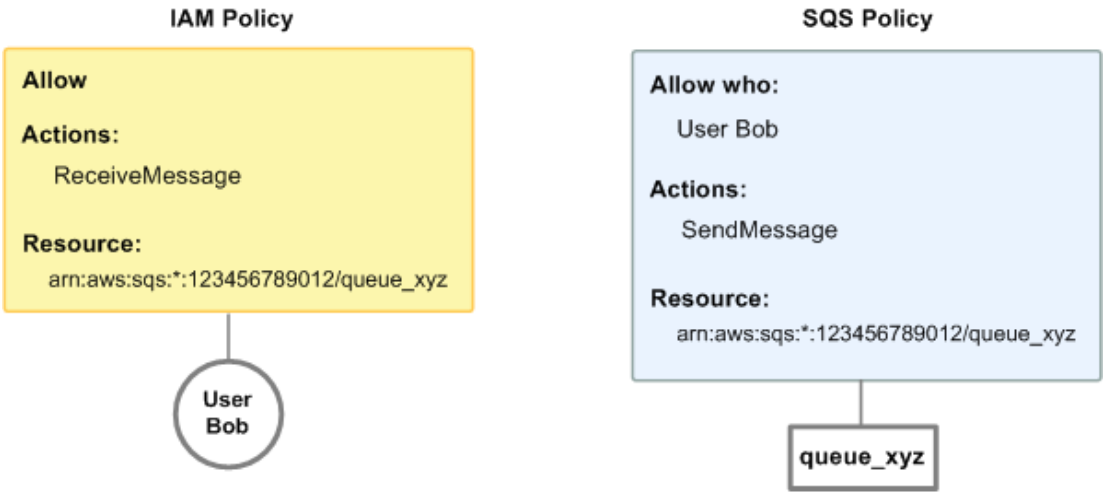
The preceding example shows simple policies with no conditions. You could specify a particular condition in either policy and get the same result.

There is one difference between IAM and SQS policies: the SQS policy system lets you grant permission to other AWS Accounts, whereas AWS IAM doesn't.

It's up to you how you use both of the systems together to manage your permissions, based on your needs. The following examples show how the two policy systems work together.

Example 1

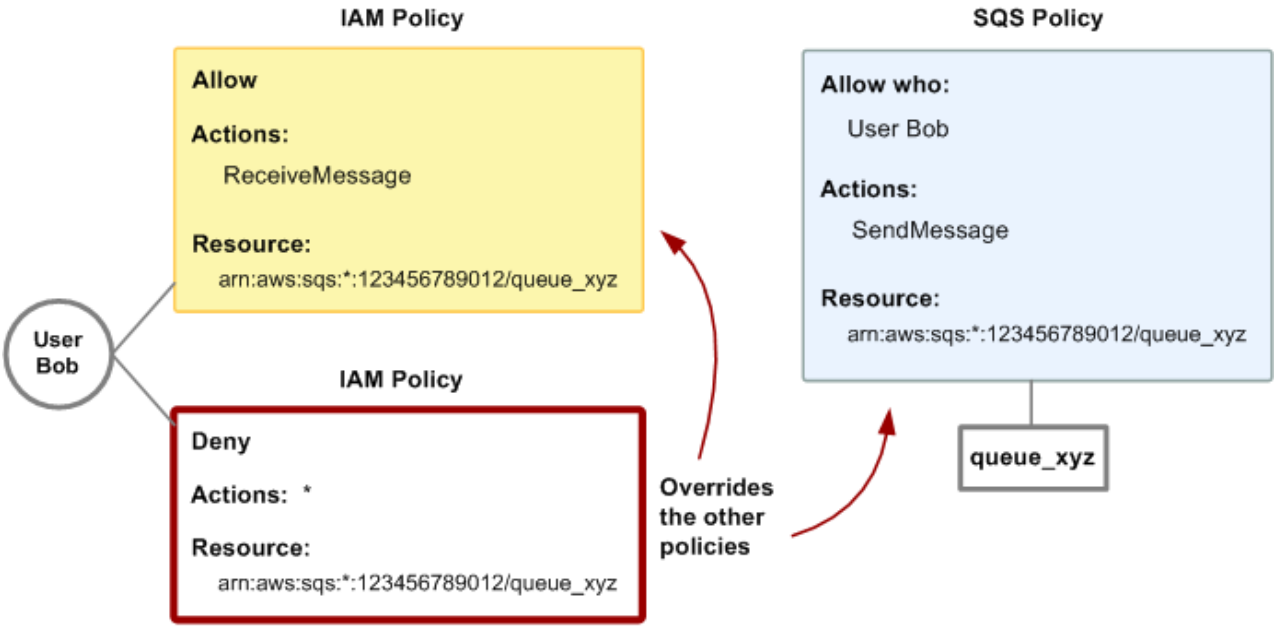
In this example, Bob has both an AWS IAM policy and an SQS policy that apply to him. The AWS IAM policy gives him permission to use `ReceiveMessage` on queue_xyz, whereas the SQS policy gives him permission to use `SendMessage` on the same queue. The following diagram illustrates the concept.



If Bob were to send a request to receive a message from queue_xyz, the AWS IAM policy would allow the action. If Bob were to send a request to send a message to queue_xyz, the SQS policy would allow the action.

Example 2

In this example, we build on example 1 (where Bob has two policies that apply to him). Let's say that Bob abuses his access to queue_xyz, so you want to remove his entire access to that queue. The easiest thing to do is add a policy that denies him access to all actions on the queue. This third policy overrides the other two, because an explicit deny always overrides an allow (for more information about policy evaluation logic, see Evaluation Logic (p. 48)). The following diagram illustrates the concept.



Alternatively, you could add an additional statement to the SQS policy that denies Bob any type of access to the queue. It would have the same effect as adding a AWS IAM policy that denies him access to the queue.

For examples of policies that cover Amazon SQS actions and resources, see Example AWS IAM Policies for Amazon SQS (p. 77). For more information about writing SQS policies, go to the Amazon Simple Queue Service Developer Guide.

Amazon SQS ARNs

For Amazon SQS, queues are the only resource type you can specify in a policy. Following is the Amazon Resource Name (ARN) format for queues:

```
arn:aws:sqs:region:account_ID:queue_name
```

For more information about ARNs, go to ARNs in *Using Identity and Access Management*.

Example

Following is an ARN for a queue named my_queue in the us-east-1 region, belonging to AWS Account 123456789012.

```
arn:aws:sqs:us-east-1:123456789012:my_queue
```

Example

If you had a queue named my_queue in each of the different Regions that Amazon SQS supports, you could specify the queues with the following ARN.

```
arn:aws:sqs:*:123456789012:my_queue
```

You can use * and ? wildcards in the queue name. For example, the following could refer to all the queues Bob has created, which he has prefixed with `bob_`.

```
arn:aws:sqs:*:123456789012:bob_*
```

As a convenience to you, SQS has a queue attribute called `Arn` whose value is the queue's ARN. You can get the value by calling the SQS `GetQueueAttributes` action.

Amazon SQS Actions

All Amazon SQS actions that you specify in a policy must be prefixed with the lowercase string `sqs:`. For example, `sqs:CreateQueue`.

Before the introduction of AWS IAM, you could use an SQS policy with a queue to specify which AWS Accounts have access to the queue. You could also specify the type of access (e.g., `sqs:SendMessage`, `sqs:ReceiveMessage`, etc.). The specific actions you could grant permission for were a subset of the overall set of SQS actions. When you wrote an SQS policy and specified * to mean "all the SQS actions", that meant all actions in that subset. That subset originally included:

- `sqs:SendMessage`
- `sqs:ReceiveMessage`
- `sqs:ChangeMessageVisibility`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes` (for all attributes except `Policy`)

With the introduction of AWS IAM, that list of actions expanded to include the following actions:

- `sqs:CreateQueue`
- `sqs:DeleteQueue`
- `sqs:ListQueues`

The actions related to granting and removing permissions from a queue (`sqs:AddPermission`, etc.) are reserved and so don't appear in the preceding two lists. This means that *Users* in the AWS Account can't use those actions. However, the *AWS Account* can use those actions.

Amazon SQS Keys

Amazon SQS implements the following policy keys, but no others. For more information about policy keys, see Condition (p. 59).

AWS-Wide Policy Keys

- `aws:CurrentTime` (for date/time conditions)
- `aws:EpochTime` (the date in epoch or UNIX time, for use with date/time conditions)
- `aws:SecureTransport` (Boolean representing whether the request was sent using SSL)
- `aws:SourceIp` (the requester's IP address, for use with IP address conditions)
- `aws:UserAgent` (information about the requester's client application, for use with string conditions)

If you use `aws:SourceIp`, and the request comes from an Amazon EC2 instance, we evaluate the instance's public IP address to determine if access is allowed.

For services that use only SSL, such as Amazon RDS and Amazon Route 53, the `aws:SecureTransport` key has no meaning.

The key names are case insensitive. For example, `aws:CurrentTime` is equivalent to `AWS:currenttime`.

Example AWS IAM Policies for Amazon SQS

This section shows several simple AWS IAM policies for controlling User access to Amazon SQS.

> **Note**
>
> In the future, Amazon SQS might add new actions that should logically be included in one of the following policies, based on the policy's stated goals.

Example 1: Allow a User to create and use his or her own queues

In this example, we create a policy for Bob that lets him access all Amazon SQS actions, but only with queues whose names begin with the literal string `bob_queue`.

> **Note**
>
> Amazon SQS doesn't automatically grant the creator of a queue permission to subsequently use the queue. Therefore, in our AWS IAM policy, we must explicitly grant Bob permission to use all the SQS actions in addition to `CreateQueue`.

```
{
   "Statement":[{
      "Effect":"Allow",
      "Action":"sqs:*",
      "Resource":"arn:aws:sqs:*:123456789012:bob_queue*"
      }
   ]
}
```

Example 2: Allow developers to write messages to a shared test queue

In this 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 AWS Account's queue named *CompanyTestQueue*.

```
{
   "Statement":[{
      "Effect":"Allow",
      "Action":"sqs:SendMessage",
      "Resource":"arn:aws:sqs:*:123456789012:CompanyTestQueue"
      }
   ]
}
```

Example 3: Allow managers to get the general size of queues

In this 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 AWS Account's queues.

```
{
   "Statement":[{
      "Effect":"Allow",
      "Action":"sqs:GetQueueAttributes",
      "Resource":"*"
     }
   ]
}
```

Example 4: Allow a partner to send messages to a particular queue

You could do this with an SQS policy or an AWS IAM policy. Using an SQS policy might be easier if the partner has an AWS Account. However, anyone in the partner's company who possesses the AWS Account credentials could send messages to the queue (and not just a particular User). We'll assume you want to limit access to a particular person (or application), so you need to treat the partner like a User within your own company, and use a AWS IAM policy instead of an SQS policy.

In this example, we create a group called WidgetCo that represents the partner company, then create a User for the specific person (or application) at the partner company who needs access, and then put the User in the group.

We then attach a policy that gives the group `SendMessage` access on the specific queue named *WidgetPartnerQueue*.

We also want to prevent the WidgetCo group from doing anything else with queues, so we add a statement that denies permission to any Amazon SQS actions besides `SendMessage` on any queue besides WidgetPartnerQueue. This is only necessary if there's a broad policy elsewhere in the system that gives Users wide access to Amazon SQS.

```
{
   "Statement":[{
         "Effect":"Allow",
         "Action":"sqs:SendMessage",
         "Resource":"arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
      },
      {
         "Effect":"Deny",
         "NotAction":"sqs:SendMessage",
         "NotResource":"arn:aws:sqs:*:123456789012:WidgetPartnerQueue"
      }
   ]
}
```

Using Temporary Security Credentials

In addition to creating IAM users with their own security credentials, IAM also enables you to grant temporary security credentials to any user allowing this user to access your AWS services and resources. You can manage users who have AWS accounts; these users are IAM users. You can also manage users for your system who do not have AWS accounts; these users are called federated users. Additionally, "users" can also be applications that you create to access your AWS resources.

You can use these temporary security credentials in making 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.

First, use IAM to create temporary security credentials, which include a security token, an Access Key ID, and a Secret Access Key. Second, prepare your string to sign with the temporary Access Key ID and the security token. Third, use the temporary Secret Access Key instead of your own Secret Access Key to sign your Query API request. Finally, when you submit the signed Query API request, don't forget to use the temporary Access Key ID instead of your own Access Key ID and include the security token. For more information about IAM support for temporary security credentials, go to Granting Temporary Access to Your AWS Resources in *Using IAM*.

Example

To call an Amazon SQS Query API action using Temporary Security Credentials

1. Request a temporary security token with AWS Identity and Access Management. For more information, go to Creating Temporary Security Credentials to Enable Access for IAM Users in Using AWS Identity and Access Management.
   AWS IAM returns a security token, an Access Key ID, and a Secret Access Key.
2. Prepare your Query as you normally would, but use the temporary Access Key ID in place 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.

```
http://sqs.us-east-1.amazonaws.com/
?Action=CreateQueue
&DefaultVisibilityTimeout=40
&QueueName=testQueue
&Attribute.1.Name=VisibilityTimeout
&Attribute.1.Value=40
&Version=2011-10-01
&Signature=Dqlp3Sd6ljTUA9Uf6SGtEExwUQEXAMPLE
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Expires=2011-10-18T22%3A52%3A43PST
&SecurityToken=SecurityTokenValue
&AWSAccessKeyId= Access Key ID provided by AWS Security Token Service
```

   The following example uses Temporary Security Credentials to send two messages with `SendMessageBatch`.

```
http://sqs.us-east-1.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
&Version=2011-10-01
&Expires=2011-10-18T22%3A52%3A43PST
&Signature=Dqlp3Sd6ljTUA9Uf6SGtEExwUQEXAMPLE
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&SecurityToken=SecurityTokenValue
&AWSAccessKeyId=Access Key ID provided by AWS Security Token Service
```

Appendix A: Increasing Throughput with Horizontal Scaling and Batching

By Marc Levy, July 2012

Amazon SQS queues can deliver very high throughput (many thousands of messages per second). The key to achieving this throughput is to horizontally scale message producers and consumers. In addition, you can use the batching actions in the Amazon SQS API to send, receive, or delete up to 10 messages at a time. In conjunction with horizontal scaling, batching achieves a given throughput with fewer threads, connections, and requests than would be required by individual message requests. Because Amazon SQS charges by the request instead of by the message, batching can also substantially reduce costs.

This appendix discusses horizontal scaling and batching in more detail. It then walks through a simple example that you can try out yourself. It also briefly discusses Amazon SQS throughput metrics that you can monitor by using Amazon CloudWatch.

Horizontal Scaling

Because you access Amazon SQS through an HTTP request-response protocol, the request latency (the time interval between initiating a request and receiving a response) limits the throughput that you can achieve from a single thread over a single connection. For example, if the latency from an Amazon Elastic Compute Cloud (EC2) based client to Amazon SQS in the same region averages around 20 ms, the maximum throughput from a single thread over a single connection will average 50 operations per second.

Horizontal scaling means increasing the number of your message producers (making SendMessage requests) and consumers (making ReceiveMessage and DeleteMessage requests) in order to increase your overall queue throughput. You can scale horizontally by increasing the number of threads on a client, adding clients, or both. You should achieve essentially linear gains in queue throughput as you add more clients. For example, if you double the number of clients, you will get twice the throughput.

Important

As you scale horizontally, you need to ensure that the Amazon SQS that you are using has enough connections or threads to support the number of concurrent message producers and consumers that will be sending requests and receiving responses. For example, by default, instances of the AWS Java SDK AmazonSQSClient class maintain at most 50 connections to Amazon SQS. To create additional concurrent producers and consumers, you'll need to adjust that limit. For example, in the AWS Java SDK, you can adjust the maximum number of allowable producer and consumer threads on an AmazonSQSClient object with this line of code:

```
AmazonSQS sqsClient = new AmazonSQSClient(credentials,
                          new ClientConfiguration().withMaxConnec
tions(producerCount + consumerCount));
```

For the AWS Java SDK asynchronous client AmazonSQSAsyncClient, you'll also need to make sure there are enough threads available. For more information, consult the documentation for the SDK library that you are using.

Batching

The batching actions in the Amazon SQS API (SendMessageBatch and DeleteMessageBatch), which were introduced in October 2011 (WSDL 2011-10-01), can further optimize throughput by processing up to ten messages at a time. ReceiveMessage can process ten messages at a time, so there is no ReceiveMessageBatch action.

The basic idea of batching is to perform more work in each round trip to the service (e.g., sending multiple messages with a single SendMessageBatch request), and to distribute the latency of the batch operation over the multiple messages in the batch request, as opposed to accepting 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 and so improve throughput. Amazon SQS charges by the request, so the cost can be greatly reduced when fewer requests are processing the same number of messages. Moreover, fewer threads and connections reduce client-side resource utilization and can reduce client-side cost by doing the same work with smaller or fewer hosts.

Batching does introduce a bit of complication for the application. For example, the application has to accumulate the messages before sending them and it will sometimes have to wait longer for a response, but batching can be effective in the following circumstances:

- Your application is generating a lot of messages in a short time, so the delay is never very long.
- A message consumer fetches messages from a queue at its discretion, as opposed to typical message producers that need to send messages in response to events they do not control.

Important

A batch request (SendMessageBatch or DeleteMessageBatch) may succeed even though individual messages in the batch have failed. After a batch request, you should always check for individual message failures and retry them if necessary.

Example

The example presented in this section implements a simple producer-consumer pattern. The complete example is available as a free download at https://s3.amazonaws.com/cloudformation-examples/sqs-producer-consumer-sample.tar. The resources that are deployed by each template are described later in this section.

The code for the samples is available on the provisioned instances in /tmp/sqs-producer-consumer-sample/src. The command line for the configured run is in /tmp/sqs-producer-consumer-sample/command.log.

The main thread spawns a number of producer and consumer threads that process 1KB messages for a specified time. The example includes producers and consumers that make single-operation requests and others that make batch requests.

In the program, each producer thread sends messages until the main thread stops the producer thread. The producedCount object tracks the number of messages produced by all producer threads. Error handling is simple: if there is an error, the program exits the run() method. Requests that fail on transient errors are, by default, retried three times by the AmazonSQSClient, so very few such errors are surfaced. The retry count can be configured as necessary to reduce the number of exceptions that are thrown. The run() method on the message producer is implemented as follows:

```
try {
  while (!stop.get()) {
    sqsClient.sendMessage(new SendMessageRequest(queueUrl, theMessage));
    producedCount.incrementAndGet();
  }
} catch (AmazonClientException e) {
  // By default AmazonSQSClient retries calls 3 times before failing,
  // so when this rare condition occurs, simply stop.
  log.error("Producer: " + e.getMessage());
  System.exit(1);
}
```

The batch producer is much the same. One noteworthy difference is the need to retry failed individual batch entries:

```
SendMessageBatchResult batchResult = sqsClient.sendMessageBatch(batchRequest);


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));
}
```

The consumer run() method is as follows:

```
while (!stop.get()) {
     result = sqsClient.receiveMessage(new ReceiveMessageRequest(queueUrl));
```

```
    if (!result.getMessages().isEmpty()) {
      m = result.getMessages().get(0);
      sqsClient.deleteMessage(new DeleteMessageRequest(queueUrl,
                                                       m.getReceiptHandle()));

      consumedCount.incrementAndGet();
    }
}
```

Each consumer thread receives and deletes messages until it is stopped by the main thread. The consumedCount object tracks the number of messages that are consumed by all consumer threads, and the count is periodically logged. The batch consumer is similar, except that up to ten messages are received at a time, and it uses DeleteMessageBatch instead of DeleteMessage.

Running the Example

You can use the AWS CloudFormation templates provided to run the example code in three different configurations: single host with the single operation requests, two hosts with the single operation requests, one host with the batch requests.

Important

The complete sample is available in a single .tar file. The resources that are deployed by each template are described later in this section.

The code for the samples is available on the provisioned instance(s) in /tmp/sqs-producer-consumer-sample/src. The command line for the configured run is in /tmp/sqs-producer-consumer-sample/command.log.

The default duration (20min) is set to provide three or four 5-minute Amazon CloudWatch data points of volume metrics. The Amazon EC2 cost for each run will be the m1.large instance cost. The Amazon SQS cost varies based on the API call rate for each sample, and that should range between approximately 38,000 API calls / min for the batching sample and 380,000 API calls /min for the 2 host single API sample. For example, a run of the single API sample on a single host should cost approximately 1 instance hour of an m1.large (large standard on demand instance, $0.32 as of July 2012) and 20min x 190,000 API calls / min x $1 / 1,000,000 API calls = $3.80 for SQS operations with the default 20min duration (as of July 2012, check current pricing).

If you want to deploy the AWS CloudFormation stack in a region other than US East (Virginia), in the Region box of the AWS CloudFormation console, click the region that you want.

To run the example

1. Click the link below that corresponds to the stack that you want to launch:

   - Single Operation API, One Host: The SQS_Sample_Base_Producer_Consumer.template sample template uses the single operation form of Amazon SQS API requests: SendMessage, ReceiveMessage, and DeleteMessage. A single m1.large EC2 instance animates 16 producer threads and 32 consumer threads.

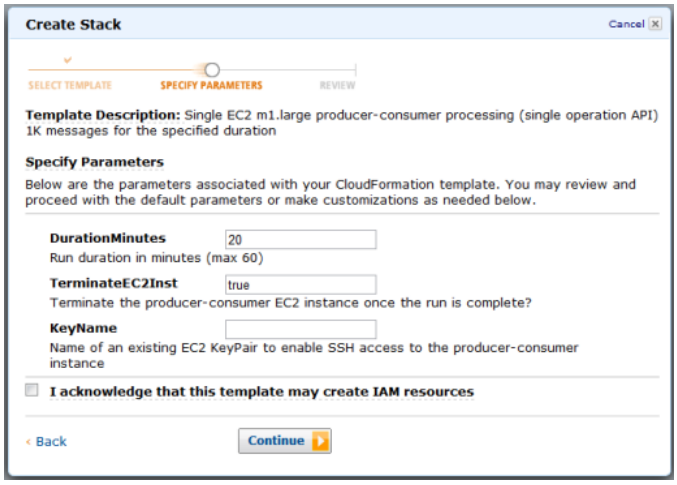
     To view the template, go to https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Base_Producer_Consumer.template

- Single Operation API, Two Hosts: SQS_Sample_Base_Producer_Consumer_x2.template sample template uses the single operation form of SQS API requests, but instead of a single m1.large EC2 instance, it uses two, each with 16 producer threads and 32 consumer threads for a total of 32 producers and 64 consumers. It illustrates Amazon SQS' elasticity with throughput increasing proportionally to the greater number of producers and consumers.

  To view the template, go to https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Base_Producer_Consumer_x2.template
- Batch API, One Host: The SQS_Sample_Batch_Producer_Consumer.template sample template uses the batch form of Amazon SQS API requests on a single m1.large EC2 instance with 12 producer threads and 20 consumer threads.

  To view the template, go to https://s3.amazonaws.com/cloudformation-templates-us-east-1/SQS_Sample_Batch_Producer_Consumer.template

2. If you are prompted, sign in to the AWS Management Console.
3. In the **Create Stack** wizard, on the **Select Template** page, click **Continue**.
4. On the **Specify Parameters** page, specify how long the program should run, whether or not you want to automatically terminate the Amazon EC2 instances when the run is complete, and provide an Amazon EC2 key pair so that you can access the instances that are running the sample. Here is an example:



5. Select the **I acknowledge that this template may create IAM resources** check box. All templates create an AWS Identity and Access Management (IAM) account so that the producer-consumer program can access the queue without requiring access to your account root credentials.
6. When all the settings are as you want them, click **Continue**.
7. On the **Review** page, review the settings. If they are as you want them, click **Continue**. If not, click **Back** and make the necessary changes.
8. On the final page of the wizard, click **Close**. Stack deployment may take several minutes.

To follow the progress of stack deployment, in the AWS CloudFormation console, click the sample stack. In the lower pane, click the **Events** tab. After the stack is created, it should take less than 5 minutes for the sample to start running. When it does, you can see the queue in the Amazon SQS console.

To monitor queue activity, you can do the following:

- Access the client instance, and open its output log file (/tmp/sqs-producer-consumer-sample/output.log) for a tally of messages produced and consumed so far. This tally is updated once per second.

- In the Amazon SQS console, observe changes in the **Message Available** and **Messages in Flight** numbers.

In addition, after a delay of up to 15 minutes after the queue is started, you can monitor the queue in Amazon CloudWatch as described later in this topic.

Although the templates and samples have safeguards to prevent excessive use of resources, it is best to delete your AWS CloudFormation stacks when you are done running the samples. To do so, in the Amazon SQS console, click the stack that you want to delete, and then click **Delete Stack**. When the resources are all deleted, Amazon CloudWatch metrics will all drop to zero.

Monitoring Volume Metrics from Example Run

Amazon SQS automatically generates volume metrics for messages sent, received, and deleted. You can access those metrics and others through the Amazon CloudWatch console. The metrics can take up to 15 minutes after the queue starts to become available. To manage the search result set, click **Search**, and then select the check boxes that correspond to the queues and metrics that you want to monitor.

Here is the NumberOfMessageSent metric for consecutive runs of the three samples. Your results may vary somewhat, but the results should be qualitatively similar:

- The NumberOfMessagesReceived and NumberOfMessagesDeleted metrics show the same pattern, but we have omitted them from this graph to reduce clutter.
- The first sample (single operation API on a single m1.large) delivers approximately 210,000 messages over 5 minutes, or about 700 messages per second, with the same throughput for receive and delete operations.
- The second sample (single operation API on two m1.large instances) delivers roughly double that throughput: approximately 440,000 messages in 5 minutes, or about 1,450 messages per second, with the same throughput for receive and delete operations.
- The last sample (batch API on a single m1.large) delivers over 800,000 messages in 5 minutes, or about 2,500 messages per second, with the same throughput for received and deleted messages. With a batch size of 10, these messages are processed with far fewer requests and therefore at lower cost.

Amazon SQS Resources

The following table lists related resources that you'll find useful as you work with this service.

Resource	Description
Amazon Simple Queue Service Getting Started Guide	The getting started guide provides a quick tutorial of the service based on a simple use case. Examples and instructions in multiple programming languages are included.
Amazon Simple Queue Service API Reference	The API reference gives the WSDL location; complete descriptions of the API actions, parameters, and data types; and a list of errors that the service returns.
Amazon SQS Release Notes	The release notes give a high-level overview of the current release. They specifically note any new features, corrections, and known issues.
AWS Developer Resource Center	A central starting point to find documentation, code samples, release notes, and other information to help you build innovative applications with AWS.
Discussion Forums	A community-based forum for developers to discuss technical questions related to Amazon SQS.
AWS Support Center	The home page for AWS Technical Support, including access to our Developer Forums, Technical FAQs, Service Status page, and AWS Premium Support (if you are subscribed to this program).
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.
Product information for Amazon SQS	The primary web page for information about Amazon SQS.
Contact Us	A central contact point for inquiries concerning AWS billing, account, events, abuse etc.
Conditions of Use	Detailed information about the copyright and trademark usage at Amazon.com and other topics.

Glossary

Access Key ID	An identifier associated with your Secret Access Key. Used for request authentication. For more information, see Your AWS Identifiers (p. 25).
action	The action is the activity the principal has permission to perform. The action is B in the statement "A has permission to do B to C where D applies." The action is just the operation in the request to SQS. For example, Jane sends a request to Amazon SQS with `Action=ReceiveMessage`. For more information, see Shared Queues (p. 37).
conditions	The conditions are any restrictions or details about the permission. The condition is D in the statement "A has permission to do B to C where D applies." Following are some of the common types of conditions:
issuer	The issuer is the person who writes a policy to grant permissions to a resource. The issuer (by definition) is always the resource owner. AWS does not permit SQS users to create policies for resources they don't own. If John is the resource owner, AWS authenticates John's identity when he submits the policy he's written to grant permissions for that resource.
message ID	An identifier you get when you send a message to the queue.
permission	A permission allows or disallows access to a particular resource. You can state any permission like this: "A has permission to do B to C where D applies." For example, Jane (A) has permission to read messages (B) from John's Amazon SQS queue (C), as long as she asks to receive only a maximum of 10 messages from the queue at a time (D). Whenever Jane sends a request to Amazon SQS to use John's queue, the service checks to see if she has permission and if the request satisfies the conditions John set forth in the permission. For more information, see Shared Queues (p. 37).
principal	The principal is the person or persons who receive the permission in the policy. The principal is A in the statement "A has permission to do B to C where D applies." In a policy, you may set the principal to "anyone" (i.e., you can specify a wildcard to represent all people). You might do this, for example, if you don't want to restrict access based on the actual identity of the requester, but instead on some other identifying characteristic such as the requester's IP address.
queue URL	The URL uniquely identifying a queue.
policy	A policy is the formal description of the permissions for a resource. The Access Policy Language distinguishes between a policy and a statement. A policy is the complete document that can contain many different permissions for a given resource. A statement is the description of an individual permission. Therefore a

	policy can contain multiple statements. For example, a policy could specify that Jane can use John's queue (one statement), and Bob cannot use John's queue (another statement).
Query	This is a type of HTTP request that generally uses only the GET or POST HTTP method and a query string with parameters.
receipt handle	An identifier you get when you receive a message from the queue. You must provide this identifier when deleting a message from the queue or when changing a message's visibility timeout.
requester	The requester is the person who sends a request to an AWS service and asks for access to a particular resource. The requester sends a request to AWS that essentially says: "Can A do B to C where D applies?" In this question, the requester is A.
resource	The resource is the object the principal is requesting access to. The resource is C in the statement "A has permission to do B to C where D applies."
Secret Access Key	A key that Amazon Web Services (AWS) assigns to you when you sign up for an AWS account. Used for request authentication. For more information, see Your AWS Account (p. 25).
visibility timeout	The length of time (in seconds) that a message that has been received from a queue will be invisible to other receiving components when they ask to receive messages. During the visibility timeout, the component that received the message usually processes the message and then deletes it from the queue. For more information, see Visibility Timeout (p. 7).

Document History

This document is associated with the 2011-10-01 release of the Amazon Simple Queue Service. This guide was last updated on 25 July 2012.

The following table describes the important changes since the last release of the *Amazon Simple Queue Service Developer Guide*.

Change	Description	Release Date
New feature	The 2011-10-01 API version of Amazon SQS supports three new batch API actions. You can send or delete up to ten messages with a single call to either `SendMessageBatch` or `DeleteMessageBatch`. You can also change the visibility timeout of up to ten messages with one call to `ChangeMessageVisibilityBatch`. For more information about the new batch actions, see Amazon SQS Batch API actions (p. 18).	In this release
New feature	The 2011-10-01 API version of Amazon SQS introduces a new `GetQueueUrl` action. You can quickly retrieve a queue's URL with `GetQueueUrl`. For more information about `GetQueueUrl`, go to GetQueueUrl in the *Amazon Simple Queue Service API Reference*.	In this release
New feature	The 2011-10-01 API version of Amazon SQS adds delay queues and message timers. Delay queues allow you to postpone the delivery of all messages in a queue for a specific number of seconds. Message timers allow you to postpone delivery of a single message for a specific number of seconds. For both delay queues and message timers, the delay period begins when you add the message to the queue. For more information about delay queues, see Amazon SQS Delay Queues (p. 10). For more information about message timers, see Amazon SQS Message Timers (p. 14).	In this release

Index

V