

Cloud computing with Amazon Web Services, Part 4: Reliable messaging with SQS

Skill Level: Introductory

[Prabhakar Chaganti \(prabhakar@ylastic.com\)](mailto:prabhakar@ylastic.com)
CTO
Ylastic, LLC.

02 Dec 2008

In this [series](#), learn about cloud computing using Amazon Web Services. Explore how the services provide a compelling alternative for architecting and building scalable, reliable applications. In this article, learn about the reliable and scalable messaging service provided by Amazon Simple Queue Service (SQS).

Amazon SQS

Amazon Simple Queue Service (SQS) is a scalable and reliable messaging framework that makes it simple to create, store, and retrieve text messages. You can use it as a base for gluing together your Amazon Web Services-based applications. Using SQS is a great way to build Web-scale applications that are truly decoupled. You pay for the messages based entirely upon your usage. The whole queuing framework runs inside the secure environment of Amazon's own data centers.

Some of the most features provided by SQS are:

Reliability

SQS is designed to store the messages redundantly across multiple data centers and to make them available at all times.

Simplicity

The programming model for accessing and using SQS is simple and can be used from a variety of programming languages.

Security

SQS is designed to provide a high level of security. Access to messages is restricted to authorized users.

Scalability

SQS gives you the ability to create queue-based applications that can read and write unlimited messages, with no restrictions or limits.

Inexpensive

SQS rates make it a very economical and compelling alternative for your messaging needs.

The rest of this section explores the concepts that underpin the SQS framework.

Messages

Messages contain text data up to 8KB in size. Each message is stored until it is retrieved by a receiving application. A visibility timeout value, in seconds, is specified when the receiving application reads a message from a queue. This acts more like a lock and ensures that, for the specified time period:

- The retrieved message will not be available to any other consumer of the queue.
- The message will only reappear in the queue when the timeout period expires if, and only if, it has not been deleted by the reading process.

Messages are retained in a queue for four days.

Amazon CTO Werner Vogels has a discussion of the rationale for eventual consistency on his blog (see [Resources](#)).

SQS will automatically delete any messages that have been in your queues longer than four days. SQS follows the model of "eventual consistency," meaning you can send a message to the queue, but a consumer of that queue may not see the message for some significant period of time. The message will eventually be delivered, but this is an important consideration if your application cares about the order of the messages.

A message consists of the parts shown in Table 1.

Table 1. Parts of a message

Part	Description
MessageId	A unique ID that references the message.
ReceiptHandle	A unique handle that's returned when a message is received from a queue. This handle is different each time you receive a message from the queue. It is required when you delete the

	message.
MD5OfBody	The MD5 digest of the non-URL-encoded message body string.
Body	The actual message data.

Queues

Queues are containers for the messages. Each message must specify a queue that will hold it. Messages sent to a queue remain on the queue until you explicitly delete them. The ordering of the queue is FIFO (first in, first out), but the order is not guaranteed. Each queue has a default visibility timeout of 30 seconds. You can change this value for the entire queue, or it can be individually set for each message on retrieval. The maximum visibility timeout for a queue or message is two hours (7200 seconds). SQS reserves the right to automatically delete queues if there has been no activity in the queue for 30 consecutive days.

Design considerations

SQS is a little different from the common queue frameworks. There are three things you must consider before designing your SQS-based applications:

- SQS does not guarantee order of the messages in a queue. The messages are loosely ordered in the queue; they are not really stored in the order in which they are added to the queue. SQS will try to preserve the order in messages, but it is not guaranteed that you will receive messages in the exact order you sent them. If the ordering of messages is important to your application, you will need to add sequencing data to each message.
- SQS does not guarantee deletion of a message in the queue. You must design your application so that it is not affected if the same message is processed more than once. Each of your messages is stored on multiple servers by SQS to provide redundancy and high availability. If one of these multiple servers becomes unavailable while a message is being deleted, it is possible, in rare circumstances, to get a message copy again while retrieving messages.
- SQS does not guarantee that all the messages in the queue will be returned when queried. SQS uses message sampling based on weighted random distribution, and it returns messages only from the sampled subset of servers when you query for messages. Even though a particular request may not return all messages in the queue, if you keep retrieving from the queue it will

end up sampling all of the servers and you'll get all your messages.

API versions

Currently there are two different versions of SQS available: the original version (2007-05-01) and a more recent version released earlier this year (2008-01-01). Each version of the API is commonly referred to using the date of the release. The 2007-05-01 API will sunset on May 6, 2009, after which only the latest version of the API will be supported. It is strongly advised that users:

- Start migrating any applications they've built using the older API version as soon as possible.
- To minimize any disruptions, use the latest version of the API when creating new applications with SQS.

The 2008-01-01 version updated the pricing, which will bring down the cost of SQS usage for most users, and included additional features and modifications. However, it also introduced significant and incompatible changes with the older APIs. All libraries and tools that are built on the older version will need to be modified. A detailed description of the changes between the versions is available on the SQS Web site (see [Resources](#)).

Pricing

The following pricing details are only for the 2008-01-01 version. (You can get the details of the pricing structure for the older versions on the SQS site (see [Resources](#)). Pricing is based on:

- The number of requests made to SQS, which includes the following operations:
 - CreateQueue
 - ListQueues
 - DeleteQueue
 - SendMessage
 - ReceiveMessage
 - DeleteMessage
 - SetQueueAttributes
 - GetQueueAttributes

Table 2. Pricing for requests

Type	Cost
Request to SQS	\$0.000001 per request

- The amount of data transferred to and from SQS. There is no charge for data transferred between SQS and EC2 instances.

Table 3. Pricing for data transfer

Type of transfer	Cost
All data transfer	\$0.100 per GB - all data <i>transfer in</i> \$0.170 per GB - first 10TB / month data <i>transfer out</i> \$0.130 per GB - next 40TB / month data <i>transfer out</i> \$0.110 per GB - next 100TB / month data <i>transfer out</i> \$0.100 per GB - data <i>transfer out</i> / month over 150TB

Check Amazon SQS for the latest pricing information. You can also use the Amazon Web Services Simple Monthly Calculator tool for calculating your monthly usage costs for SQS and the other Amazon Web Services (see [Resources](#)).

Getting started with Amazon Web Services and SQS

To start exploring SQS, you will first need to sign up for an Amazon Web Services account (see [Resources](#)). See [Part 2](#) of this series for detailed instructions on signing up for Amazon Web Services.

Once you have an Amazon Web Services account, you must enable Amazon SQS service for your account using the following steps.

1. [Log in](#) to your Amazon Web Services account.
2. Navigate to the [SQS home page](#).
3. Click **Sign Up For Amazon SQS** on the right side.

4. Provide the requested information and complete the sign-up process.

All communication with any of the Amazon Web Services is through either the SOAP interface or the query interface. In this article, you use the query interface by way of a third-party library to communicate with SQS.

You will need to obtain your access keys, which you can access from your [Web Services Account information page](#) by selecting **View Access Key Identifiers**. You are now set up to use Amazon Web Services, and have enabled SQS service for your account.

Interacting with SQS

For this example, you use an open source third-party [python](#) library named [boto](#) to become familiar with SQS by running small snippets of code in a python shell.

Install boto and set up your environment

Download boto from the [project page](#). The latest version, as of the writing of this article, was 1.4c. Unzip the archive to the directory of your choice. Change into this directory and run `setup.py` to install boto into your local python environment, as shown in Listing 1.

Listing 1. Install boto

```
$ cd directory_where_you_unzipped_boto
$ python setup.py install
```

Set up some environment variables to point to the Amazon Web Services access keys. The access keys are available from [Web Services Account information](#).

Listing 2. Set up environment variables

```
# Export variables with your AWS access keys
$ export AWS_ACCESS_KEY_ID=Your_AWS_Access_Key_ID
$ export AWS_SECRET_ACCESS_KEY=Your_AWS_Secret_Access_Key
```

Check to make sure everything is set up correctly by starting a python shell and importing the boto library, as shown in Listing 3.

Listing 3. Check the setup

```
$ python
Python 2.4.5 (#1, Apr 12 2008, 02:18:19)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
```

```
Type "help", "copyright", "credits" or "license" for more
information.
>>> import boto
>>>
```

Explore SQS with boto

Use the `SQSConnection` class to provide the main interface for the interaction with SQS. The medium for the usage of boto is the python console. The example calls different methods on the `SQSConnection` object and examines the responses returned by SQS, which will help you get familiar with the API while you explore the concepts behind SQS.

The first step is to create a connection object to SQS using the Amazon Web Services access keys that you exported earlier to your environment. The boto library always checks the environment first to see if these variables are set. If they are set, boto automatically uses them when it creates the connection.

Listing 4. Create a connection to SQS

```
>>> import boto
>>> sqs_conn = boto.connect_sqs()
>>>
```

For the rest of this article you can use the `sqs_conn` object, created above, to interact with SQS. You can create a queue by specifying a name for the queue, along with an optional visibility timeout value. If you omit a value for the timeout, boto will create the queue with the default value of 30 seconds provided by SQS.

Listing 5. Create a queue

```
>>> q1 = sqs_conn.create_queue('devworks-sqs-1')
>>>
>>> q1.get_timeout()
30
>>> q2 = sqs_conn.create_queue('devworks-sqs-2', 60)
>>>
>>> q2.get_timeout()
60
>>>
```

Retrieve a list of all your queues, which returns a resultset object that is essentially a python list, as shown in Listing 6. You can iterate over this list and access all pertinent information for each queue.

Listing 6. List all the queues

```
>>> all_queues = sqs_conn.get_all_queues()
>>>
>>> len(all_queues)
2
>>>
```

```
>>> for q in all_queues:
...     print q.id
...     print q.count()
...     print q.get_timeout()
...     print q.url
...
/devworks-sqs-1
0
30
http://queue.amazonaws.com/devworks-sqs-1

/devworks-sqs-2
0
60
http://queue.amazonaws.com/devworks-sqs-2
```

You must delete all the messages in a queue before deleting the queue. There is a `clear()` method in boto that you can use to delete all the messages in a queue.

Listing 7. Clear and delete queues

```
>>> q2.clear()
0
>>> sqs_conn.delete_queue(q2)
True
>>>
```

You can send text messages with a maximum size of 8KB to a queue. Create a new message by using the boto `Message` class, as shown in Listing 8.

Listing 8. Send a message

```
>>> from boto.sqs.message import Message
>>>
>>> m1 = Message()
>>>
>>> m1.set_body('Hi there devworks!')
>>>
>>> status = q1.write(m1)
>>>
>>> print status
True
>>>
```

Retrieval of messages for a queue returns a resultset object that is a python list containing message objects. Each message object has a unique ID and a receipt handle associated with it. When you read a message from a queue, that message automatically becomes invisible to all other consumers of the queue until the visibility timeout period set for the queue expires. After the expiration, the message once appears in the queue, giving another consumer the chance to retrieve the message and process it. However, if the message is deleted from the queue before the expiration of the visibility timeout, it is gone forever and will not appear in the queue again.

Listing 9. Retrieve a message

```
>>> msgs = q1.get_messages()
>>>
>>> len(msgs)
1
>>>
>>> for msg in msgs:
...     print "Message ID: ",msg.id
...     print "Message Handle: ",msg.receipt_handle
...     print "Queue ID: ", msg.queue.id
...     print "Message Body: ", msg.get_body()
...
Message ID: 9a930aaf-87de-48ad-894d-b22dd0b1cd1b

Message Handle:
Pr10vft3nRjgDDT33svtLnzyPQGWFpRusXdn2v3Lwq+TDtD3hk3aBKbSH1mGc4hzO/VZO
IC0RFyAd7MhbJKPGHn3x35CTz9dAQeNoKYAHiwERXc/xrYXBLGngyuJI+kGmbjvIKqA/wpfQpqzPk2bVA==

Queue ID: /devworks-sqs-1

Message Body: Hi there devworks!
>>>
```

You can retrieve more than one message by specifying the number of messages. The default option in boto is to return one message. Let's add another message to the queue and then retrieve all the messages, as shown in Listing 10. Keep in mind that it might take a minute or so for a newly added message to show up in the queue.

Listing 10. Retrieve multiple messages

```
>>> m2 = Message()
>>>
>>> m2.set_body('Still there?')
>>>
>>> status = q1.write(m2)
>>>
>>> print status
True
>>>
>>> msgs = q1.get_messages(10)
>>>
>>> len(msgs)
2
>>>
>>> for msg in msgs:
...     print "Message ID: ",msg.id
...     print "Message Handle: ",msg.receipt_handle
...     print "Queue ID: ", msg.queue.id
...     print "Message Body: ", msg.get_body()
...     print "*" * 80
...
Message ID: 9a930aaf-87de-48ad-894d-b22dd0b1cd1b

Message Handle:
Pr10vft3nRjgDDT33svtLnzyPQGWFpRusXdn2v3Lwq+TDtD3hk3aBKbSH1mGc4hzO/VZOIC0R
FyAd7MhbJKPGHn3x35CTz9dAQeNoKYAHiwERXc/xrYXBLGngyuJI+kGmbjvIKqA/wpfQpqzPk2bVA==

Queue ID: /devworks-sqs-1

Message Body: Hi there devworks!
```

```
Message ID: ce1632b3-0a6e-4ee2-a5b0-b2e9821d150f
Message Handle:
Pr10vft3nRiRunVNVvjOQEc7Tm+uSBQpW4bZcpFMbzWTDtD3hk3aBKbSH1mGc4hzO/VZOIC0R
FxbhtlykUxvNbRQNWJqrMXrxj5m6GwhA7iX0Nu9mqjo+9/hnda8Ou0df+LQ3dOMfXSybzbhed128w==
Queue ID: /devworks-sqs-1
Message Body: Still there?
>>>
```

Messages can be deleted from a queue by invoking the `delete_message()`. Remember, you must delete all the messages in a queue before you delete a queue.

Listing 11. Delete a message

```
>>> msgs = q1.get_messages()
>>>
>>> len(msgs)
1
>>> print msgs[0].get_body()
Hi there devworks!
>>>
>>> q1.delete_message(msgs[0])
True
>>>
```

Conclusion

This article introduced you to Amazon's SQS service. You learned some of the basic concepts and explored some of the functions provided by boto, an open source python library for interacting with SQS. It is highly recommended that you read the Amazon SQS Developer Guide for more information (see [Resources](#)).

Stay tuned for Part 5 in this series, which will examine Amazon SimpleDB for dataset processing in the cloud.

Resources

Learn

- Check out the other parts in this series:
 - [Part 1, "Introduction: When it's smarter to rent than to buy"](#)
 - [Part 2, "Storage in the cloud with Amazon Simple Storage Service \(S3\)"](#)
 - [Part 3, "Servers on demand with EC2"](#)
- Learn about specific Amazon Web Services:
 - [Amazon Simple Storage Service \(S3\)](#)
 - [Amazon Elastic Compute Cloud \(EC2\)](#)
 - [Amazon Simple Queue Service \(SQS\)](#)
 - [Amazon SimpleDB \(SDB\)](#)
 - The Service Health [Dashboard](#) is updated by the Amazon team and provides the current status of each service.
 - The latest happenings in the world of Amazon Web Services are on the [blog](#).
- [Sign up](#) for an Amazon Web Services account.
- The Amazon Web Services [Developer Connection](#) is the gateway to all the developer resources.
- The [Amazon SQS Developer Guide](#) contains information on the various components of SQS, along with advanced usage and configuration.
- [Migrating to Amazon SQS API Version 2008-01-01](#) (Amazon Articles & Tutorials) describes the changes in the 2008-01-01 version.
- Use the [Simple Monthly Calculator](#) for calculating your monthly usage costs for EC2 and the other Amazon Web Services.
- Check out Werner Vogel's [blog](#) for a discussion of the rationale for eventual consistency.
- Get the [RSS](#) feed for this series.
- In the [Architecture area on developerWorks](#), get the resources you need to advance your skills in the architecture arena.
- Browse the [technology bookstore](#) for books on these and other technical topics.

Get products and technologies

- Download [IBM product evaluation versions](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational, Tivoli® and WebSphere.

Discuss

- Check out [developerWorks blogs](#) and get involved in the [developerWorks community](#).

About the author

Prabhakar Chaganti

Prabhakar Chaganti is the CTO of [Ylastic](#), a start-up that is building a single unified interface to architect, manage, and monitor a user's entire AWS Cloud computing environment: EC2, S3, SQS and SimpleDB. He is the author of two recent books, *Xen Virtualization* and *GWT Java AJAX Programming*. He is also the winner of the community choice award for the most innovative virtual appliance in the VMware Global Virtual Appliance Challenge.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>.

Other company, product, or service names may be trademarks or service marks of others.