




Practical: AWS SQS (Python)

Table of Contents

- [Prerequisites and references](#)
- [Create a queue](#)
- [Access the queue from the console](#)
- [Accessing via Python](#)
- [Look at the code](#)
- [Using visibility timeouts](#)
- [A few more notes](#)

AWS SQS (Simple Queue Service)

Prerequisites and references

- [AWS Python SDK SQS examples](#)  (https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/python/example_code/sqs)
- [AWS Python SDK SQS API reference](#)  (<https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sqs.html>)
- [AWS SQS documentation](#)  (<https://docs.aws.amazon.com/sqs/>)
- [sqs_python.zip](#) (<https://canvas.qut.edu.au/courses/20367/files/6924321/download>)
- [sqs_python_async.zip](#) (<https://canvas.qut.edu.au/courses/20367/files/6924322/download>)

Create a queue

- Find SQS in the AWS console
- *Create queue*
- Choose a name like `n12345267-test-queue`
- Add a tag with key `qut-username` and value like `n123467@qut.edu.au`
- *Create queue*

Make a note of the *URL* for the queue, which will be needed for later.

Access the queue from the console

- While viewing the queue details page, click on *Send and receive messages* in the top right.
- Enter a message (eg. `testing out SQS`) and click *Send message*
- You should see the *Messages available* count in the *Receive messages* section change to 1.
- Click *Poll for messages*. You should see a message come up in the list.
- By default polling will continue for 30 seconds, or you can stop manually.
- Click on the message ID to see the contents of the message. The message will remain on the queue until you delete it.

Accessing via Python

Download the demo script: [sqs_python.zip\]\(./files/sqs_python.zip\) or \[:filelink\\[sqs_python_async.zip \\(https://canvas.qut.edu.au/courses/20367/files/6924322/download\\)\]\(#\)](#)

. The async version uses the `aioboto3` custom library to access SQS asynchronously, while the synchronous version uses the `boto3` library. Both versions are similar in structure and functionality for demonstrating SQS operations. You can still achieve asynchronous behaviour with the synchronous version by using threads or asyncio, but the async version is more straightforward for this purpose. For those who are interested with further details, check out this blog post: <https://joelmccoy.medium.com/python-and-boto3-performance-adventures-synchronous-vs-asynchronous-aws-api-interaction-22f625ec6909>

You can do the rest on your own machine or on an EC2 instance. If you are using an EC2 instance use `scp` as in previous practicals to copy the contents of the zip file to your instance. The remaining instructions assume you have a terminal open to the directory containing the demo script.

- Install the required libraries with `pip install -r requirements.txt`
- Edit `app.py` and update the contents:
 - Change `sqsQueueUrl` to match the URL for the queue that you created. This is available on the details page for the queue.
 - If you like, change the message.
- Run `python app.py` (Note, the async version has been coded so you can just run `python app.py` as well).
- Have a look at the responses. They should all succeed and the message should be printed out.

Look at the code

SQS is quite simple as far as AWS services go. Have a look at the code, which contains three commands:

- send a message
- receive messages
- delete a message

The code shows the basic usage of these commands.

Using visibility timeouts

SQS has a requeuing mechanism which is referred to as *visibility timeout*. It works like this:

- When a message is received it is removed from the queue and stored separately. A timer is started.
 - The timer is initially set to the `VisibilityTimeout` specified when the message is sent, or the default visibility timeout for the queue.

- If a receiver deletes the message, then nothing more is done.
- If the timer expires then the message is returned to the queue.
- The receiver can also reset the timer to a new value.

A receiver should not delete the message until they have completed processing it. This way the visibility timeout mechanism will recover the message if the receiver fails.

You should set the visibility timeout (either the default, or when queueing the message) to something *longer* than the receiver will take to process it, with a healthy margin of error.

Alternatively, the receiver can periodically update the timer until it has completed processing and then delete it.

A few more notes

- You can add `MessageAttributes` to your messages. These are key-value pairs that are added to your message which can be used as metadata.
- SQS operating in standard mode does not guarantee the order of messages is preserved, and does not guarantee that a message is received only once. It is a "best effort" service and usually these properties will hold. However it is important that your application tolerate duplication of messages, for example by designing the operation required to process a message be idempotent.
- SQS has a FIFO mode which does guarantee order and non-duplication of messages, but it has lower capacity.

TEQSA PRV12079 | CRICOS 00213J | ABN 83 791 724 622