AWS SQS + Lambda Setup Tutorial - Step by Step

Many cloud-based solutions require its tasks, like web applications or backends, to call external services, third-party or not. A service that has variable loads at variable times can present performance or reliability issues if it is under-provisioned when traffic peaks or it can be wasting resources constantly if it's over-provisioned most of the time.

One common architecture pattern used to control the load and the rate at which a service processes this load is to use message queues that act as buffers between a task and the services being called. This smooths out heavy loads that could cause the service to fail or the task to time out.

We can control the rate at which data is processed by buffering it. What if we could automatically increase and decrease our computing capacity according to the volume of data? i.e. when queues have a higher or lower number of messages? This is where event-driven serverless architecture shines. With AWS Lambda, you can run code without provisioning or managing servers. Just upload your code and Lambda scales and runs it for you.

Now, how do you use queues to invoke the right amount of Lambda functions that match your load? Lambda comes out of the box with a feature called event source mapping. You can use it to process items from a stream or queue of AWS services that don't invoke Lambda functions directly. Thus, you can use an Amazon Simple Queue Service (SQS) with Lambda by setting queue events as event sources that trigger your Lambda function.

In this article, we present a step-by-step guide on how to set up a Lambda function that responds to events of an SQS queue and how the services work under the hood.

Under The Hood

- How Lambda works

Lambda runs instances of a **function** to process **events** when **triggered**. A **function** is a resource that contains the code to process the **events** you pass into it or that other AWS services send to it. A **trigger** can be either AWS services that invoke a function directly or **event source mappings**, resources that read items from a stream or queue and invokes a function.

When Lambda invokes your function it does so in an execution environment. The lifecycle of the execution environment includes the phases depicted below.

Lambda EXECUTION ENVIRONMENT lifecycle.

In the ***Init*** phase, Lambda creates the execution environment, downloads the code for the function and all layers, initializes any extensions, initializes the runtime, and then runs the function's **initialization code** (the code outside the handler).

In the ***Invoke*** phase, Lambda invokes the function **handler**. The Lambda **runtime** passes two arguments to the function handler: *event* and *context*. When the handler exits or returns a response, it becomes available to handle another event.

The ***Shutdown*** phase is triggered if the function does not receive any invocations for a while.

- How the Simple Queue Service works

SQS is an implementation of the producer-consumer design pattern, where a producer is responsible for adding data, called messages, to a buffer (queue) that will be removed and processed by the consumer.

Queue design example.

The lifecycle of a message goes as follows:

- A producer (Task) sends a message to a queue, which is distributed across the SQS servers.
- When a consumer (Service) is ready to process messages, it consumes messages from the queue.
- While a message is being processed, it remains invisible in the queue for the duration of the **visibility timeout**.
- The consumer deletes the message from the queue to prevent the message from being received and processed again when the visibility timeout expires.
- SQS automatically deletes messages that have been in a queue for more than the maximum message configured **retention period**.

SQS supports two types of queues – standard queues and First-In-First-Out (FIFO) queues. Standard type is the default and supports at-least-once message delivery but, as it is distributed and redundant, one message can be delivered more than once if you have more than one consumer for a queue. FIFO queues give you more control at the cost of lower throughput.

- How Lambda and SQS work together

When a Lambda function subscribes to an SQS queue, meaning it uses a queue as an event source, it becomes a consumer of that queue. Lambda then polls the queue and invokes your Lambda function with an event that contains queue messages.

Lambda reads messages in batches and invokes your function **once for each batch**. Before invoking the function, Lambda continues to poll messages from the queue until the **batch window** expires or the **maximum batch size** is reached. You can configure the event source to buffer records for up to 5 minutes or 10,000 records of

batch size, whatever comes first. There is an **invocation payload size quota** set by Lambda that is also a threshold for invoking a function, It tops at 6 MB each for request and response.

A function returning THE BATCH to A queue after failing to process the third message.
Source: shorturl.at/hqCO0

Each time a batch of messages is received by a function, it processes the data until it fails or succeeds. Some behaviors upon success or failure go as follows:

- When your function successfully processes a batch, Lambda deletes its messages from the queue.
- When your function returns an error, Lambda leaves the messages in the queue.
- When some messages fail, you can configure your event source mapping to make only the failed messages visible again to avoid reprocessing all messages in a failed batch.
- If your function fails to process a message multiple times, SQS can send it to a dead-letter queue (DLQ), if you've configured one.

For standard queues, Lambda uses long polling to poll a queue until it becomes active. When messages are available, Lambda reads up to five batches and sends them to your function. If messages are still available, Lambda increases the number of processes that are reading batches by up to 60 more instances per minute. The maximum number of batches that an event source mapping can process simultaneously is 1,000.

Tutorial

In this tutorial, we will be using the AWS Management Console to create our Lambda function, and our queue and to connect and test everything. AWS provides many tools for you to manage services and resources, check this article for an overview and to learn when to use each tool.

Part 1 - Creating your lambda function.
Part 2 - Creating your queue.
Part 3 - Subscribing the function to the queue.
Part 4 - Editing your lambda function to receive messages.
Part 4 - Testing the architecture behavior.

Part 1 - Creating your lambda function

1. Open the functions page of the Lambda console and click Create function.

Lambda service console.

2. Choosing your deployment package.

You can deploy your function using a deployment package. Two types are supported: a .zip file archive that contains your code and its dependencies or, a container image to which you add your code and dependencies.

Function Deployment package options.

3. As we are not adding any dependencies we can choose 'Author from Scratch' and edit our code later in the console. Lambda will zip it and store it for us.Configuring Basic Information

After choosing a name for your function configure the following settings.

1. Runtime

Lambda provides runtimes for .NET (PowerShell, C#), Go, Java, Node.js, Python, and Ruby. Choose Python 3.9.

2. Architecture

Lambda offers two architectures x86_64 and arm64. Functions that use arm64 architecture offer a lower cost per Gb/s compared with the equivalent function running on an x86-based CPU.

3. Permissions

We need to allow our Lambda function to poll from our queue, thus select "Create a new role from AWS policy templates", give it a name of your choice and search for SQS in "Policy Templates". Then, select "Amazon SQS poller permissions". Lambda will create the IAM user with the correct permissions policies.

Selecting a policy template.

4. Advanced Settings

We don't need to set any Advanced settings for now.

4. Click on “Create Function”.

5. Find the code editor.

On your lambda function page, under the tab “Code”, in the box “Code Source” you can edit your code.

Lambda console code editor.

We will come back to edit the code in Part 4 - Editing your lambda function to receive messages, after subscribing our lambda function to our queue.Part 2 - Creating your SQS queue.

1. Open the Amazon SQS console and choose Create queue.
2. Leave the default Standard queue type and enter a name for your queue.
3. Under Configuration, you can set new values for the following parameters:

Configuration section of create queue in sqs.

    1. **Visibility timeout** is the amount of time a message being processed will stay invisible before returning to the queue. If a message is not deleted by the consumer after processing it will return after this period.
    2. **Message retention period** is the maximum time a message stays in the queue awaiting processing.
    3. **Delivery delay** is the time a message will stay invisible when it is first sent to the queue before it can be read by the consumers.
    4. **Receive message wait time** is the maximum amount of time that polling will wait for messages to become available to receive. Any non-zero value sets long polling.
2. Encryption of the messages is enabled by default.
3. Your Access Policies are the definitions of who can send and receive messages from the queue. You can set permissions to let other users and accounts to use your queue. For now, choose “Only the queue owner” for both sending and receiving.
4. The Dead Letter Queue (DLQ) configuration is out of the scope of this article.
5. Click “Create Queue”.

Part 3 - Subscribing the function to the queue.

1. Go back to your Lambda function page at the console.
2. Under Function overview, choose ‘Add trigger’.

Lambda function page - function overview.

3. Click “Select Source” and search for SQS.

Lambda Trigger configuration page.

4. Under SQS queue select the queue you’ve created.
5. Unmark the checkbox “Enable Trigger”. This will help with testing later.
6. Configure the Batch Size and Batch Window.
    1. Batch Size: the **maximum** number of messages each function receives with each invocation. If your Batch Window is 0 it will take as many as are available to the maximum of Batch Size. Leave it at 10.
    2. Batch Window: the time lambda will wait and gather messages before invoking. Leave it blank or zero for now.
7. (Optional) Configure Additional Settings
    1. Set the event source to allow ‘partial successful response of batch records. This avoids reprocessing all the messages of a batch if only some failed.
    2. Filter Criteria filters the messages to contain only the fields you require for processing.
8. Click Add.Part 4 - Editing your lambda function to receive messages.

Your code starts with a single function called *lambda_handler* that receives two arguments, *event* and *context*, as shown in the snippet below.

EXAMPLE OF handler function in Python.

The first argument is the **event object** which is a JSON document that contains data for a function to process. The second argument is the **context object**, passed to your function by Lambda at runtime. This object provides methods and properties containing information about the invocation, function, and runtime environment.

Your messages will come from the *event object*. This JSON object is translated to a dictionary in Python and it contains the key *records* whose value is an array of messages in the format:

Example of a batch OF RECORDS with 1 message.

To retrieve the contents of the messages one by one, we should iterate on the records array and retrieve the *body* field of each record. The code will become as follows:

handler function THAT PRINTS MESSAGE CONTENTS in Python.

Part 5 - Testing the architecture behavior

To test our trigger and batch processing we will start by sending some messages with the trigger still deactivated (step 5 of part 3), activate it, see it processing and send more messages to see how it behaves.

1. Go back to your Queue page at the console.
2. Click on “Send and receive messages” and send any amount of random messages.

Sending messages to an sqs queue from the console.

3. On your Lambda function page, under *Configuration,* select *Triggers*. Then, *Edit.*

Lambda Trigger configuration page.

Here you can play with your Batch Size, Batch Window, and activate or deactivate the queue event source as a trigger.

4. Check *Enable trigger* and click Save.
5. On your Lambda function page, under *Monitor,* select *Logs*.

Under recent invocations are the latest invocations of the lambda function by any trigger you’ve configured. As we only have our SQS queue messages as triggers it should look something like the image below:

Lambda function recent invocations log.

Click on one of the LogStreams to open CloudWatch and read the logs. We used the AWS CLI to compile the logs of the latest invocations in the terminal. To learn more about how to use the AWS CLI, refer to this article. If everything worked as expected your logs should look something like this:

AWS Cloudwatch logs fetched from the aws cli.

Every *Start* to *End* marker represents an invocation of the function. With 9 messages pending to be processed before the trigger was activated, Lambda created 5 invocations with batches varying from 1 to 3 messages, as expected. Check the section **under the hood** for more information.