**What Is Amazon Simple Queue Service?**

Amazon Simple Queue Service (Amazon SQS) offers a secure, durable, and available hosted queue that lets you integrate and decouple distributed software systems and components.

It provides a generic web services API and it can be accessed by any programming language that the AWS SDK supports.

Amazon SQS supports both standard and FIFO queues.

# Amazon SQS Standard Queues

Amazon SQS offers standard as the default queue type. Standard queues support a nearly unlimited number of transactions per second (TPS) per action. Standard queues support at-least-once message delivery. However, occasionally (because of the highly distributed architecture that allows nearly unlimited throughput), more than one copy of a message might be delivered out of order. Standard queues provide best-effort ordering which ensures that messages are generally delivered in the same order as they're sent.

FIFO queues are available in the US East (Ohio), US East (N. Virginia), US West (N. California), US West (Oregon), Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), Canada (Central), China (Ningxia), EU (Frankfurt), EU (Ireland), EU (London), EU (Paris), EU (Stockholm), and South America (São Paulo) Regions. FIFO queues have all the capabilities of the standard queue.

## What Are the Main Benefits of Amazon SQS?

* **Security** – You control who can send messages to and receive messages from an Amazon SQS queue.

Server-side encryption (SSE) lets you transmit sensitive data by protecting the contents of messages in queues using keys managed in AWS Key Management Service (AWS KMS).

* **Durability** – To ensure the safety of your messages, Amazon SQS stores them on multiple servers. Standard queues support at-least-once message delivery, and FIFO queues support exactly-once message processing.
* **Availability** – Amazon SQS uses redundant infrastructure to provide highly-concurrent access to messages and high availability for producing and consuming messages.
* **Scalability** – Amazon SQS can process each buffered request independently, scaling transparently to handle any load increases or spikes without any provisioning instructions.
* **Reliability** – Amazon SQS locks your messages during processing, so that multiple producers can send and multiple consumers can receive messages at the same time.
* **Customization** – Your queues don't have to be exactly alike—for example, you can set a default delay on a queue. You can store the contents of messages larger than 256 KB using Amazon Simple Storage Service (Amazon S3) or Amazon DynamoDB, with Amazon SQS holding a pointer to the Amazon S3 object, or you can split a large message into smaller messages.

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

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

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

## What Type of Queue Do I Need?

|  |  |
| --- | --- |
| **Standard Queue** | **FIFO Queue** |
| Available in all regions.  **Unlimited Throughput** – Standard queues support a nearly unlimited number of transactions per second (TPS) per action.  **At-Least-Once Delivery** – A message is delivered at least once, but occasionally more than one copy of a message is delivered.  **Best-Effort Ordering** – Occasionally, messages might be delivered in an order different from which they were sent. | Available in the US East (Ohio), US East (N. Virginia), US West (N. California), US West (Oregon), Asia Pacific (Mumbai), Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), Asia Pacific (Tokyo), Canada (Central), EU (Frankfurt), China (Ningxia), EU (Ireland), EU (London), EU (Paris), EU (Stockholm), and South America (São Paulo) Regions.  **High Throughput** – By default, FIFO queues support up to 3,000 messages per second with batching. To request a limit increase, file a support request. FIFO queues support up to 300 messages per second, per action (SendMessage, ReceiveMessage, or DeleteMessage) without batching.  **Exactly-Once Processing** – A message is delivered once and remains available until a consumer processes and deletes it. Duplicates aren't introduced into the queue.  **First-In-First-Out Delivery** – The order in which messages are sent and received is strictly preserved. |
| https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/images/sqs-what-is-sqs-standard-queue-diagram.png | https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/images/sqs-what-is-sqs-fifo-queue-diagram.png |
| Send data between applications when the throughput is important, for example:   * Decouple live user requests from intensive background work: let users upload media while resizing or encoding it. * Allocate tasks to multiple worker nodes: process a high number of credit card validation requests. * Batch messages for future processing: schedule multiple entries to be added to a database. | Send data between applications when the order of events is important, for example:   * Ensure that user-entered commands are executed in the right order. * Display the correct product price by sending price modifications in the right order. * Prevent a student from enrolling in a course before registering for an account. |

# Creating Amazon SQS Queues

## AWS Management Console

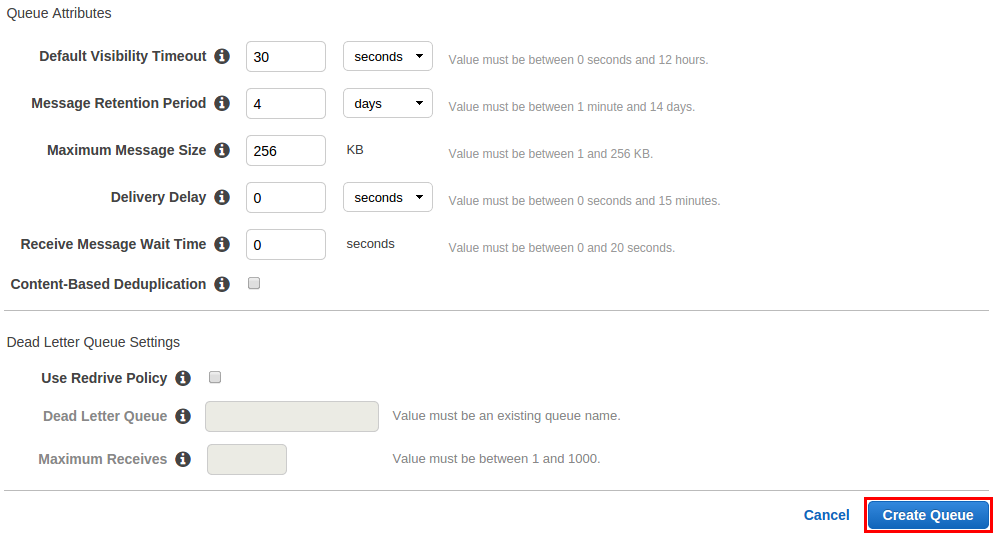
1. Sign in to the Amazon SQS console.
2. Choose **Create New Queue.**
3. On the **Create New Queue** page, ensure that you're in the correct region and then type the **Queue Name**.

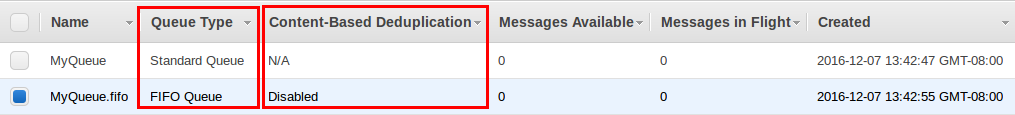
**Note**

The name of a FIFO queue must end with the .fifo suffix.

1. **Standard** is selected by default. Choose **FIFO**.
2. Create your queue.
   * To create your queue with the default parameters, choose **Quick-Create Queue**.
   * To configure your queue's parameters, choose **Configure Queue**. When you finish configuring the parameters, choose **Create Queue**. For more information about creating a queue with SSE, see Creating an Amazon SQS Queue with SSE.

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



1. Your new queue is created and selected in the queue list.
2. **Note**
3. When you create a queue, it can take a short time for the queue to propagate throughout Amazon SQS.
4. The **Queue Type** column helps you distinguish standard queues from FIFO queues at a glance. For a FIFO queue, the **Content-Based Deduplication** column displays whether you have enabled exactly-once processing.
5. 
6. Your queue's **Name**, **URL**, and **ARN** are displayed on the **Details** tab.
7. 

## AWS SDK for Java

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

### To create a standard queue

1. Copy the example program.

The following section of the code creates the MyQueue queue:

*// Create a queue*

System.out.println("Creating a new SQS queue called MyQueue.\n");

final CreateQueueRequest createQueueRequest = new CreateQueueRequest("MyQueue");

final String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();

1. Compile and run the example.

The queue is created.

### To create a FIFO queue

1. Copy the example program.

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

*// Create a FIFO queue*

System.out.println("Creating a new Amazon SQS FIFO queue called " + "MyFifoQueue.fifo.\n");

final Map<String, String> attributes = new HashMap<String, String>();

*// A FIFO queue must have the FifoQueue attribute set to True*

attributes.put("FifoQueue", "true");

*// If the user doesn't provide a MessageDeduplicationId, generate a MessageDeduplicationId based on the content.*

attributes.put("ContentBasedDeduplication", "true");

*// The FIFO queue name must end with the .fifo suffix*

final CreateQueueRequest createQueueRequest = new CreateQueueRequest("MyFifoQueue.fifo")

.withAttributes(attributes);

final String myQueueUrl = sqs.createQueue(createQueueRequest).getQueueUrl();

1. Compile and run the example.

The queue is created.

## AWS CloudFormation

You can use the AWS CloudFormation console and a JSON (or YAML) template to create an Amazon SQS queue. For more information, see Working with AWS CloudFormation Templates and the AWS::SQS::Queue Resource in the AWS CloudFormation User Guide.

1. Copy the following JSON code to a file named MyQueue.json. To create a standard queue, omit the FifoQueue andContentBasedDeduplication properties. For more information on content-based deduplication, see Exactly-Once Processing.

**Note**

The name of a FIFO queue must end with the .fifo suffix.

{

"AWSTemplateFormatVersion": "2010-09-09",

"Resources": {

"MyQueue": {

"Properties": {

"QueueName": "MyQueue.fifo",

"FifoQueue": true,

"ContentBasedDeduplication": true

},

"Type": "AWS::SQS::Queue"

}

},

"Outputs": {

"QueueName": {

"Description": "The name of the queue",

"Value": {

"Fn::GetAtt": [

"MyQueue",

"QueueName"

]

}

},

"QueueURL": {

"Description": "The URL of the queue",

"Value": {

"Ref": "MyQueue"

}

},

"QueueARN": {

"Description": "The ARN of the queue",

"Value": {

"Fn::GetAtt": [

"MyQueue",

"Arn"

]

}

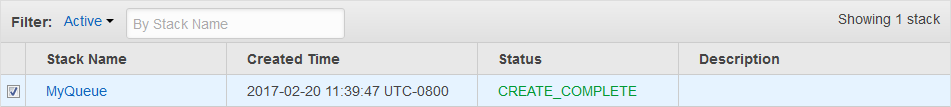
}

}

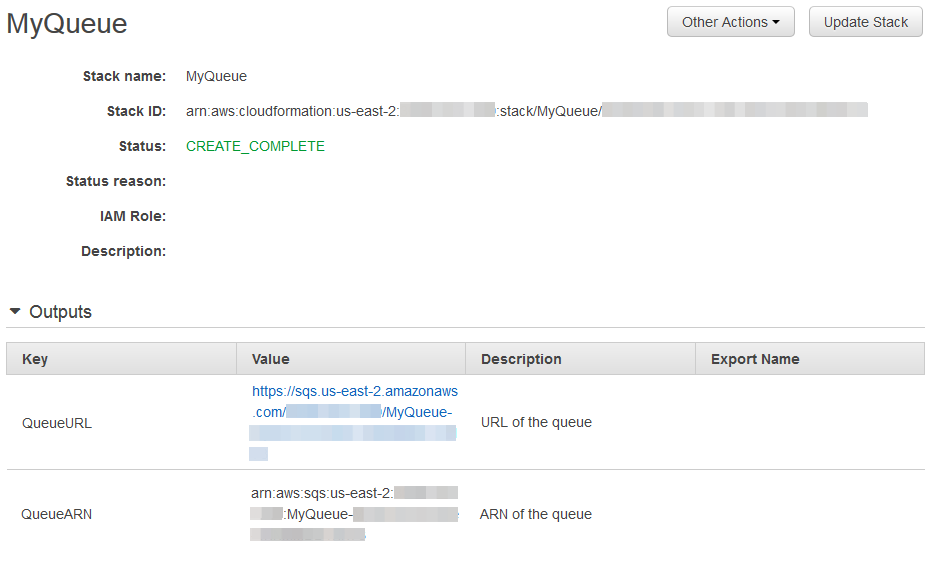
}

1. Sign in to the AWS CloudFormation console, and then choose **Create Stack**.
2. On the **Select Template** page, choose **Upload a template to Amazon S3**, choose your MyQueue.json file, and then choose **Next**.
3. On the **Specify Details** page, type MyQueue for **Stack Name**, and then choose **Next**.
4. On the **Options** page, choose **Next**.
5. On the **Review** page, choose **Create**.

AWS CloudFormation begins to create the MyQueue stack and displays the **CREATE\_IN\_PROGRESS** status. When the process is complete, AWS CloudFormation displays the **CREATE\_COMPLETE** status.



1. (Optional) To display the name, URL, and ARN of the queue, choose the name of the stack and then on the next page expand the **Outputs** section.

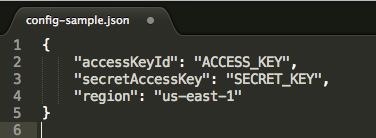


# SQS(Message Queuing Service) – Implementations

### Step:

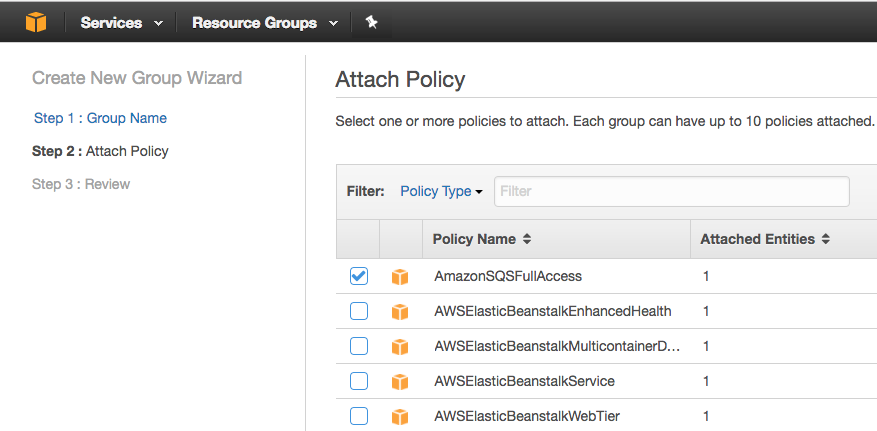
1. set up access-key and secret-key
2. $ node sqs\_createqueue.js : node (is command which start or execute the given .js file)
3. $ node sqs\_listqueues.js
4. $ node sqs\_getqueueurl.js (the reture url is used in step 5 and 6)
5. $ node sendMessage.js
6. $ node receiveMessage.js

### 1- set up access-key and secret-key

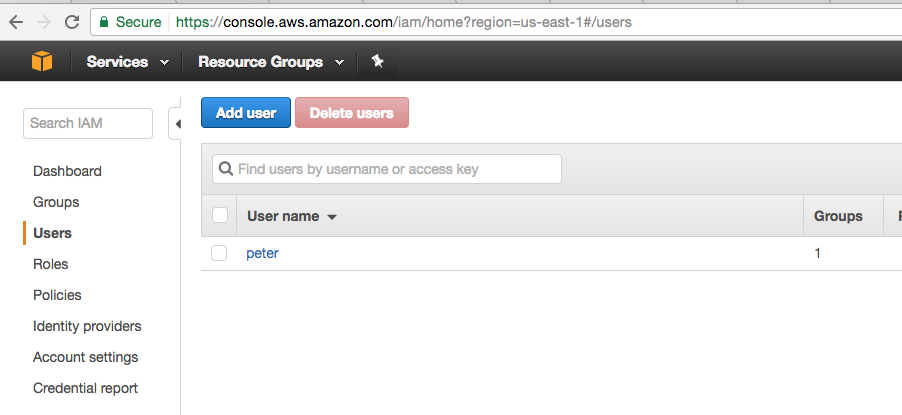


Go to **IAM management console** to create an user whom the system is going to generate access-key and secret-key

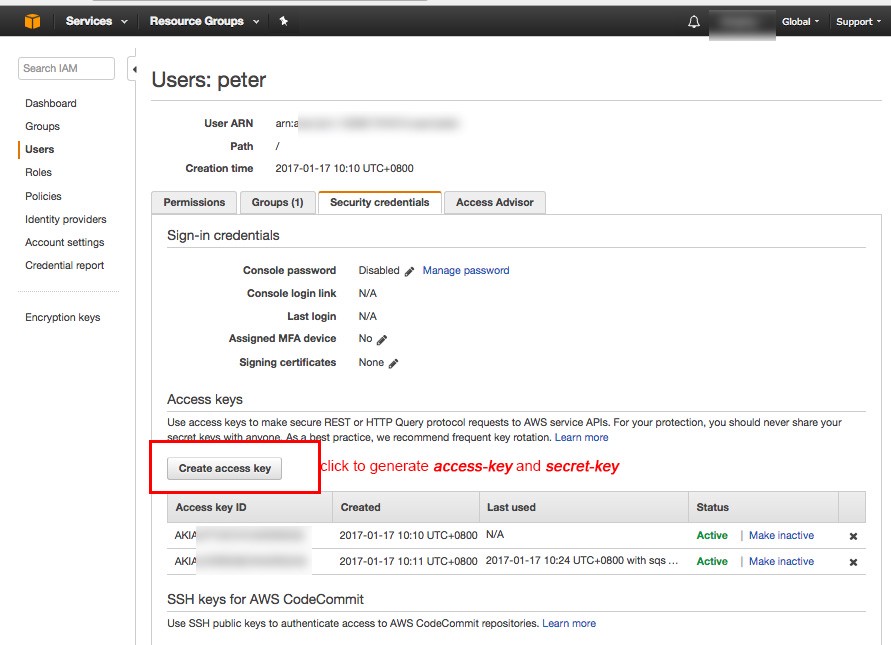
Before creating the user, a configured group with SQS permission is required. In the permission windows, select the click-box for **AmazonSQSFullAccess**, and next.



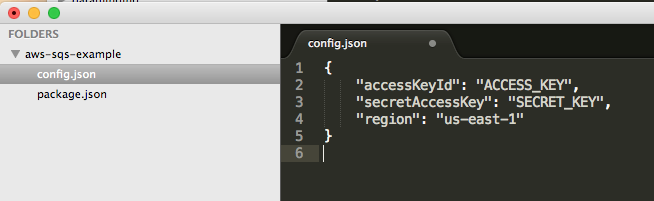
Add user to the group with access SQS permission.



Generating **access-key** and **secret-key**, which will be used in **config.json** later.



Create a **config.json** file under the Node.js directory, in which paste the **access-key** and **secret-key** which generated from the previous step to field **accessKeyId** and **secretAccessKey**.

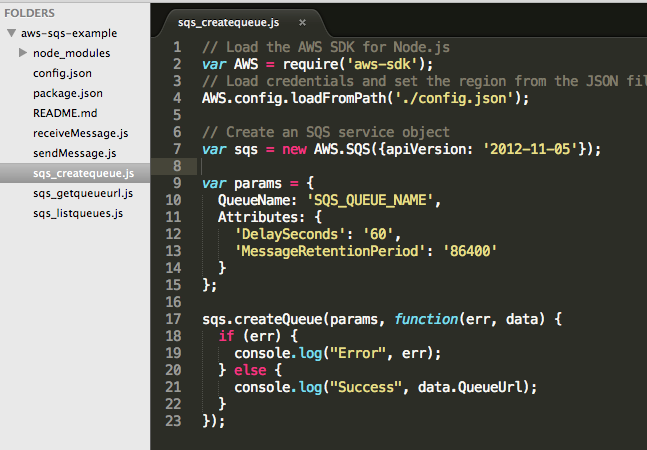


### 2- Creating Queue

Amazon not only brings us **diverse** Another great things Amazon provides us are, very easy to read documentation and super easy to use sample code. With the doc and git example, running the Helloworld example, sending/receiving message is just a piece of cake

#### Creating a Queue

run sqs\_createqueue.js file



$ node sqs\_createqueue.js

Success https://sqs.us-east-1.amazonaws.com/xxx/SQS\_QUEUE\_NAME

#### List Queue

$ node sqs\_listqueues.js

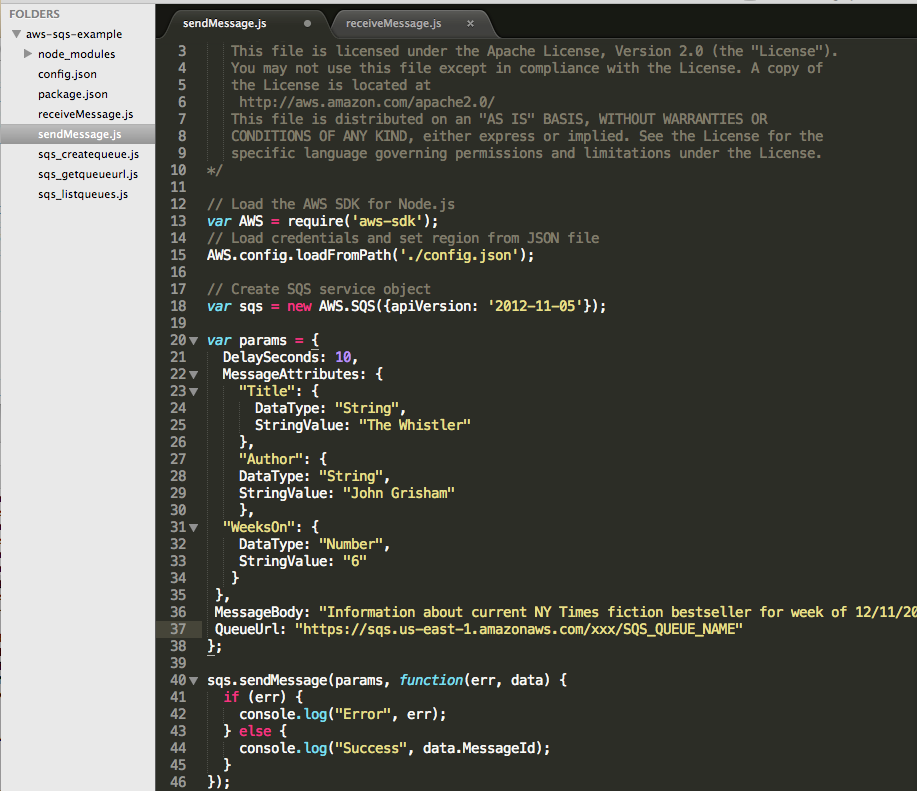
Success [ 'https://sqs.us-east-1.amazonaws.com/xxx/SQS\_QUEUE\_NAME' ]

#### Get Queue Url

$ node sqs\_getqueueurl.js

Success **https://sqs.us-east-1.amazonaws.com/xxx/SQS\_QUEUE\_NAME**

In **sendMessage.js**, set value to valuable **QueueUrl** from the return result, and the **message “Information about current NY Times…”** is going to be sent:



#### Sending Message to Queue

$ node sendMessage.js

Success 169607bb-xxx-xxx-xxx-d4d1cc270e9f

#### Receiving Message from Queue

In **receiveMessage.js**, Set value to **QueueUrl** from the **sqs\_getqueueurl.js**result:

$ node receiveMessage.js

{ ResponseMetadata: { RequestId: 'f7ae9017-0692-5eae-874e-c0e0a13a070b' },

Messages:

[ { MessageId: '169607bb-3d5a-4dab-a8bf-d4d1cc270e9f',  
 ReceiptHandle: 'xxx',  
 MD5OfBody: 'xxx',  
 **Body: 'Information about current NY Times fiction bestseller for week of 12/11/2016.',** Attributes: [Object],  
 MD5OfMessageAttributes: 'xxxx',  
 MessageAttributes: [Object] } ] }

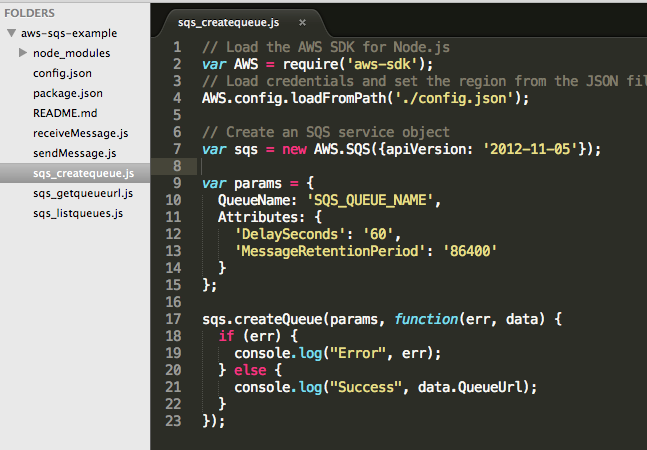
### Note

#### Message Attribute VS Message Body

— **Message attributes** are supposed to be used as message metadata (like timestamp or possibly some category) and not the message itself.

#### Creating a Queue

run sqs\_createqueue.js file



$ node sqs\_createqueue.js

Success https://sqs.us-east-1.amazonaws.com/xxx/SQS\_QUEUE\_NAME

#### List Queue

$ node sqs\_listqueues.js

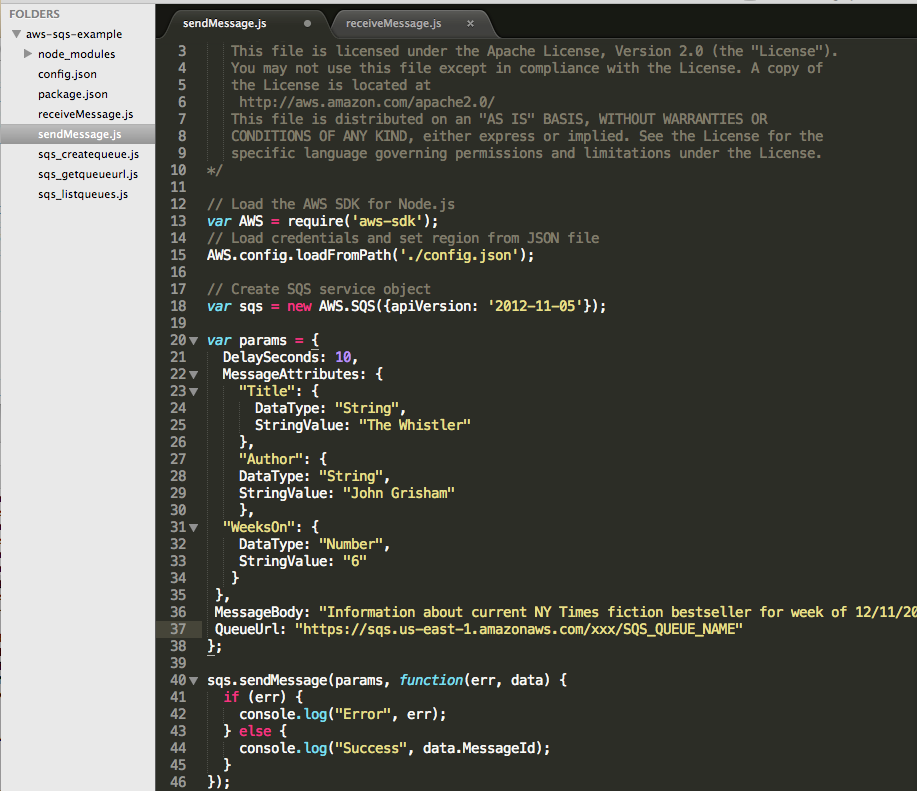
Success [ 'https://sqs.us-east-1.amazonaws.com/xxx/SQS\_QUEUE\_NAME' ]

#### Get Queue Url

$ node sqs\_getqueueurl.js

Success **https://sqs.us-east-1.amazonaws.com/xxx/SQS\_QUEUE\_NAME**

In **sendMessage.js**, set value to valuable **QueueUrl** from the return result, and the **message “Information about current NY Times…”** is going to be sent:



#### Sending Message to Queue

$ node sendMessage.js

Success 169607bb-xxx-xxx-xxx-d4d1cc270e9f

#### Receiving Message from Queue

In **receiveMessage.js**, Set value to **QueueUrl** from the **sqs\_getqueueurl.js**result:

$ node receiveMessage.js

{ ResponseMetadata: { RequestId: 'f7ae9017-0692-5eae-874e-c0e0a13a070b' },

Messages:

[ { MessageId: '169607bb-3d5a-4dab-a8bf-d4d1cc270e9f',  
 ReceiptHandle: 'xxx',  
 MD5OfBody: 'xxx',  
 **Body: 'Information about current NY Times fiction bestseller for week of 12/11/2016.',** Attributes: [Object],  
 MD5OfMessageAttributes: 'xxxx',  
 MessageAttributes: [Object] } ] }

### Note

#### Message Attribute VS Message Body

— **Message attributes** are supposed to be used as message metadata (like timestamp or possibly some category) and not the message itself.

### Reference:

https://github.com/awsdocs/aws-doc-sdk-examples/tree/master/javascript/example\_code/sqs

http://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/sqs-examples-using-queues.html

https://github.com/wahengchang/aws-sqs-example