amazon web services Certified
Developer - Associate

lab

lab title

Programming Amazon SQS and SNS using the AWS NodeJS SDK V1.00

Course title

AWS Certified Developer Associate

BackSpace

Table of Contents

Contents

About the Lab

These lab notes are to support the instructional videos on Programming Amazon SQS and SNS using the AWS NodeJS SDK in the BackSpace AWS Certified Developer course.

In this lab we will then:

- Add async package to application.
- Create an SQS queue using the AWS NodeJS SDK.
- Create SQS messages to the queue.
- Create SQS messages to the queue using the batch method.
- Process and delete SQS messages.
- Create an SNS topic.
- Create SNS messages to the SQS queue.

Please refer to the AWS JavaScript SDK documentation at:

http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/SQS.html

and

http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/SNS.html

Please note that AWS services change on a weekly basis and it is extremely important you check the version number on this document to ensure you have the lastest version with any updates or corrections.

Creating an SQS Queue using the AWS NodeJS SDK

In this section we will use the AWS NodeJS SDK to create an SQS Queue.

Make sure you have set up your NodeJS development environment as detailed in the introduction lab.

Open Atom IDE.

Go to Packages – Remote Edit – Browse Hosts

Select your EC2 instance

Select package.json

Change dependencies to include async:

```
  "dependencies": {
    "express": "^4.0.0",
    "async": "^1.4.2"
  },
```

Click Ctrl + S to save to the EC2 instance.

Now download index.js to Atom using “Remote Edit” again

Remove the code and replace with:

```
// Include the async package
// Make sure you add "async" to your package.json
var async = require('async');
// Load the AWS SDK for Node.js
```

```
var AWS = require('aws-sdk');

/**
 * Don't hard-code your credentials!
 * Create an IAM role for your EC2 instance instead.
 */

// Set your region
AWS.config.region = 'us-east-1';

var sqs = new AWS.SQS();

//Create an SQS Queue
var queueUrl;
var params = {
  QueueName: 'backspace-lab', /* required */
  Attributes: {
    ReceiveMessageWaitTimeSeconds: '20',
    VisibilityTimeout: '60'
  }
};
sqs.createQueue(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else {
    console.log('Successfully created SQS queue URL '+ data.QueueUrl);     // successful
response
  }
});
```

Click Ctrl + S to save to the EC2 instance.

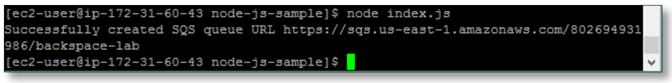
Now connect to your instance using Putty

Install dependencies using:

```
npm install
```

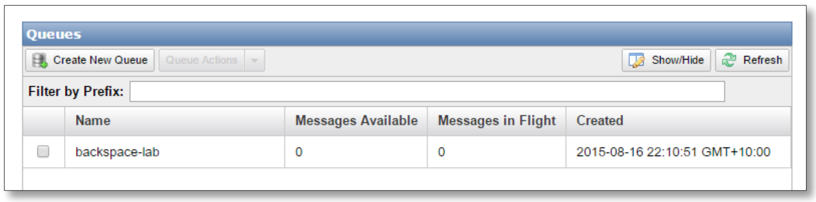
Run the application

```
node index.js
```



```
[ec2-user@ip-172-31-60-43 node-js-sample]$ node index.js
Successfully created SQS queue URL https://sqs.us-east-1.amazonaws.com/802694931
986/backspace-lab
[ec2-user@ip-172-31-60-43 node-js-sample]$
```

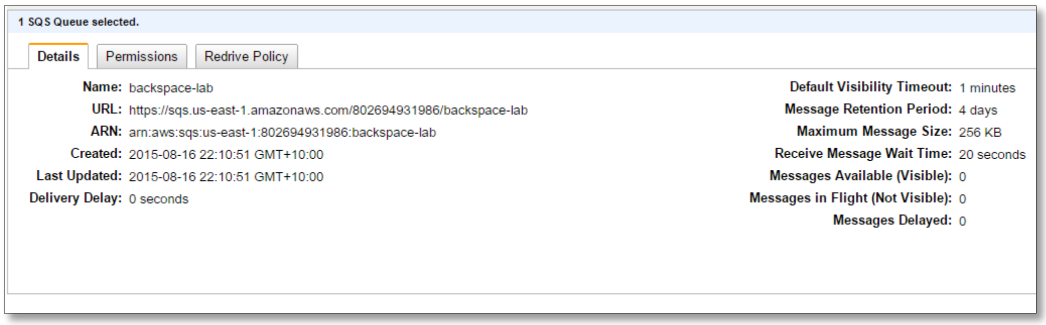
Now go to the SQS console and see your newly created SQS queue



Filter by Prefix:

	Name	Messages Available	Messages in Flight	Created
☐	backspace-lab	0	0	2015-08-16 22:10:51 GMT+10:00

Click on the queue to see its details including the visibility timeout and receive message wait time we specified in our code.



1 SQS Queue selected.

Name:	backspace-lab
URL:	https://sqs.us-east-1.amazonaws.com/802694931986/backspace-lab
ARN:	arn:aws:sqs:us-east-1:802694931986:backspace-lab
Created:	2015-08-16 22:10:51 GMT+10:00
Last Updated:	2015-08-16 22:10:51 GMT+10:00
Delivery Delay:	0 seconds
Default Visibility Timeout:	1 minutes
Message Retention Period:	4 days
Maximum Message Size:	256 KB
Receive Message Wait Time:	20 seconds
Messages Available (Visible):	0
Messages in Flight (Not Visible):	0
Messages Delayed:	0

Creating SQS Messages using the AWS NodeJS SDK

In this section we will create and add messages to our SQS queue using sendMessage asynchronously and also with sendMessageBatch.

Add a createMessages call in the sqs.createQueue method callback:

```
  sqs.createQueue(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else {
    console.log('Successfully created SQS queue URL '+ data.QueueUrl);  // successful response
    createMessages(data.QueueUrl);
  }
});
```

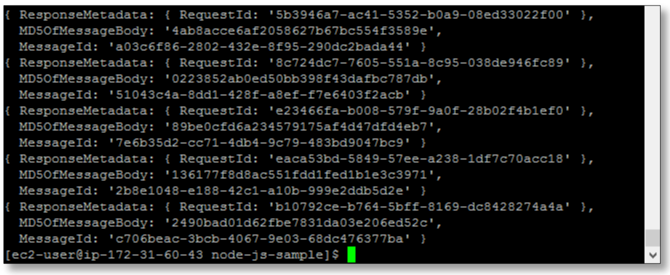
Create the createMessages function:

```
  // Create 50 SQS messages
function createMessages(queueUrl){
  var messages = [];
  for (var a=0; a<50; a++){
    messages[a] = 'This is the content for message '+ a + '.';
  }
  // Asynchronously deliver messages to SQS queue
  async.each(messages, function (content) {
    console.log('Sending message: '+content)
    tempKey = content;
    params = {
      MessageBody: content, /* required */
      QueueUrl: queueUrl /* required */
    };
    sqs.sendMessage(params, function(err, data) {
      if (err) console.log(err, err.stack); // an error occurred
      else     console.log(data);           // successful response
    });
  });
}
```

Click Ctrl + S to save to the EC2 instance.

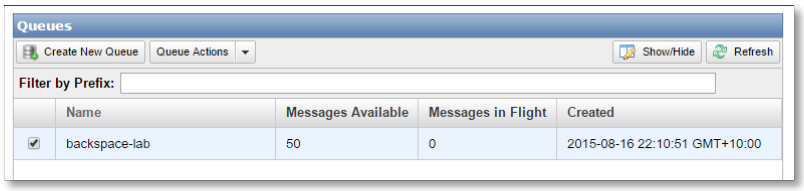
Now run index.js

It has now created 50 messages.



```
{ ResponseMetadata: { RequestId: '5b3946a7-ac41-5352-b0a9-08ed33022f00' },
  MD5OfMessageBody: '4ab8acce6af2058627b67bc554f3589e',
  MessageId: 'a03c6f86-2802-432e-8f95-290dc2bada44' }
{ ResponseMetadata: { RequestId: '8c724dc7-7605-551a-8c95-038de946fc89' },
  MD5OfMessageBody: '0223852ab0ed50bb398f43dafbc787db',
  MessageId: '51043c4a-8dd1-428f-a8ef-f7e6403f2acb' }
{ ResponseMetadata: { RequestId: 'e23466fa-b008-579f-9a0f-28b02f4b1ef0' },
  MD5OfMessageBody: '89be0cfd6a234579175af4d47dfd4eb7',
  MessageId: '7e6b35d2-cc71-4db4-9c79-483bd9047bc9' }
{ ResponseMetadata: { RequestId: 'eaca53bd-5849-57ee-a238-1df7c70acc18' },
  MD5OfMessageBody: '136177f8d8ac551fdd1fed1b1e3c3971',
  MessageId: '2b8e1048-e188-42c1-a10b-999e2ddb5d2e' }
{ ResponseMetadata: { RequestId: 'b10792ce-b764-5bff-8169-dc8428274a4a' },
  MD5OfMessageBody: '2490bad01d62fbe7831da03e206ed52c',
  MessageId: 'c706beac-3bcb-4067-9e03-68dc476377ba' }
[ec2-user@ip-172-31-60-43 node-js-sample]$
```

Now go to the SQS console and you will see the messages have been added to the queue.



	Name	Messages Available	Messages in Flight	Created
✓	backspace-lab	50	0	2015-08-16 22:10:51 GMT+10:00

If the maximum total payload size (i.e., the sum of all a batch's individual message lengths) is 256 KB (262,144 bytes) or less, we can use a single sendMessageBatch call. This reduces our number of calls and resource costs.

Now let's use sendMessageBatch to do send up to 10 messages at a time.

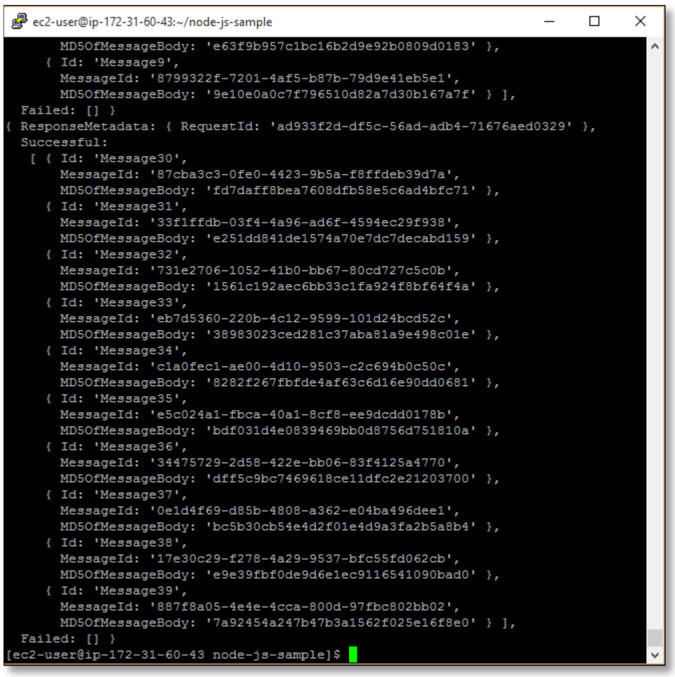
Change createMessages to:

```
  // Create 50 SQS messages
function createMessages(queueUrl){
  var messages = [];
  for (var a=0; a<5; a++){
    messages[a] = [];
    for (var b=0; b<10; b++){
    messages[a][b] = 'This is the content for message '+ (a*10+b) + '.';
    }
  }
  var a = 0;
  // Asynchronously deliver messages to SQS queue
  async.each(messages, function (content) {
        console.log('Sending messages: '+ JSON.stringify(content))
    params = {
      Entries: [],
      QueueUrl: queueUrl /* required */
    };
    for (var b=0; b<10; b++){
      params.Entries.push({
        MessageBody: content[b],
        Id: 'Message'+ (a*10+b)
      });
    }
    a++;
    // Batch deliver messages to SQS queue
    sqs.sendMessageBatch(params, function(err, data) {
      if (err) console.log(err, err.stack); // an error occurred
      else     console.log(data);           // successful response
    });
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run index.js

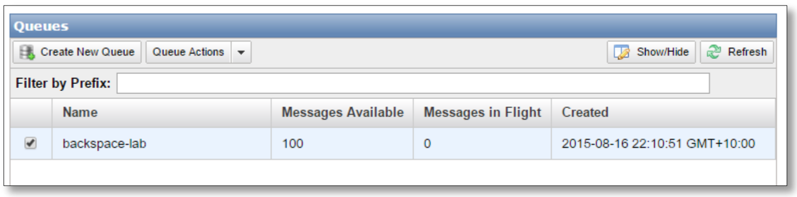
It has now created 50 messages but this time using only 5 calls to SQS instead of 50.



ec2-user@ip-172-31-60-43:~/node-js-sample

```
       MD5OfMessageBody: 'e63f9b957c1bc16b2d9e92b0809d0183' },
     { Id: 'Message9',
       MessageId: '8799322f-7201-4af5-b87b-79d9e41eb5e1',
       MD5OfMessageBody: '9e10e0a0c7f796510d82a7d30b167a7f' } ],
  Failed: [] }
{ ResponseMetadata: { RequestId: 'ad933f2d-df5c-56ad-adb4-71676aed0329' },
  Successful:
   [ { Id: 'Message30',
       MessageId: '87cba3c3-0fe0-4423-9b5a-f8ffdeb39d7a',
       MD5OfMessageBody: 'fd7daff8bea7608dfb58e5c6ad4bfc71' },
     { Id: 'Message31',
       MessageId: '33f1ffdb-03f4-4a96-ad6f-4594ec29f938',
       MD5OfMessageBody: 'e251dd841de1574a70e7dc7decabd159' },
     { Id: 'Message32',
       MessageId: '731e2706-1052-41b0-bb67-80cd727c5c0b',
       MD5OfMessageBody: '1561c192aec6bb33c1fa924f8bf64f4a' },
     { Id: 'Message33',
       MessageId: 'eb7d5360-220b-4c12-9599-101d24bcd52c',
       MD5OfMessageBody: '38983023ced281c37aba81a9e498c01e' },
     { Id: 'Message34',
       MessageId: 'c1a0fec1-ae00-4d10-9503-c2c694b0c50c',
       MD5OfMessageBody: '8282f267fbfde4af63c6d16e90dd0681' },
     { Id: 'Message35',
       MessageId: 'e5c024a1-fbca-40a1-8cf8-ee9dcdd0178b',
       MD5OfMessageBody: 'bdf031d4e0839469bb0d8756d751810a' },
     { Id: 'Message36',
       MessageId: '34475729-2d58-422e-bb06-83f4125a4770',
       MD5OfMessageBody: 'dff5c9bc7469618ce11dfc2e21203700' },
     { Id: 'Message37',
       MessageId: '0e1d4f69-d85b-4808-a362-e04ba496dee1',
       MD5OfMessageBody: 'bc5b30cb54e4d2f01e4d9a3fa2b5a8b4' },
     { Id: 'Message38',
       MessageId: '17e30c29-f278-4a29-9537-bfc55fd062cb',
       MD5OfMessageBody: 'e9e39fbf0de9d6e1ec9116541090bad0' },
     { Id: 'Message39',
       MessageId: '887f8a05-4e4e-4cca-800d-97fbc802bb02',
       MD5OfMessageBody: '7a92454a247b47b3a1562f025e16f8e0' } ],
  Failed: [] }
[ec2-user@ip-172-31-60-43 node-js-sample]$
```

Now go to the SQS console and you will see the messages have been added to the queue.



Filter by Prefix:

	Name	Messages Available	Messages in Flight	Created
✓	backspace-lab	100	0	2015-08-16 22:10:51 GMT+10:00

Processing SQS Messages using the NodeJS SDK

In this section we will use the NodeJS SDK to read, process then delete messages from an SQS queue.

First lets create a polling function with 1 second interval.

In the sqs.createQueue method success callback save the queue URL and change waitingSQS to false.

```
sqs.createQueue(params, function(err, data) {
  if (err) console.log(err, err.stack); // an error occurred
  else {
    console.log('Successfully created SQS queue URL '+ data.QueueUrl);     // successful
response
    queueUrl = data.QueueUrl;
    waitingSQS = false;
    createMessages(queueUrl);
  }
});
```

After the sqs.createQueue method call place the following code for polling SQS

```
  // Poll queue for messages then process and delete
var waitingSQS = false;
var queueCounter = 0;

setInterval(function(){
  if (!waitingSQS){ // Still busy with previous request
    if (queueCounter <= 0){
      receiveMessages();
    }
    else --queueCounter; // Reduce queue counter
  }
}, 1000);
```

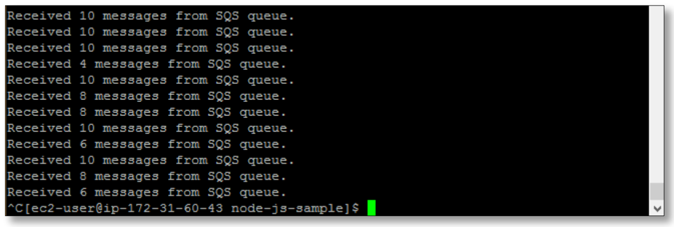
Now create a function to read up to 10 messages (the max allowed) from the SQS queue. The function halts further calls to it while it is waiting for SQS to respond. It will also halt polling for 60 seconds when the queue is empty.

```
  // Receive messages from queue
function receiveMessages(){
  var params = {
    QueueUrl: queueUrl, /* required */
    MaxNumberOfMessages: 10,
    VisibilityTimeout: 60,
    WaitTimeSeconds: 20 // Wait for messages to arrive
  };
  waitingSQS = true;
  sqs.receiveMessage(params, function(err, data) {
    if (err) {
      waitingSQS = false;
      console.log(err, err.stack); // an error occurred
    }
    else{
      waitingSQS = false;
      if ((typeof data.Messages !== 'undefined')&&(data.Messages.length !== 0)) {
        console.log('Received '+ data.Messages.length
          + ' messages from SQS queue.');           // successful response
      }
      else {
        queueCounter = 60; // Queue empty back of for 60s
        console.log('SQS queue empty, waiting for '+ queueCounter + 's.');
      }
    }
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run index.js

You can see it is receiving messages but not always 10 messages. This is normal.



```
Received 10 messages from SQS queue.
Received 10 messages from SQS queue.
Received 10 messages from SQS queue.
Received 4 messages from SQS queue.
Received 10 messages from SQS queue.
Received 8 messages from SQS queue.
Received 8 messages from SQS queue.
Received 10 messages from SQS queue.
Received 6 messages from SQS queue.
Received 10 messages from SQS queue.
Received 8 messages from SQS queue.
Received 6 messages from SQS queue.
^C[ec2-user@ip-172-31-60-43 node-js-sample]$
```

Press Ctrl + C to stop the application.

Now update receiveMessages with a call to processMessages in the callback

```
  // Receive messages from queue
function receiveMessages(){
  var params = {
    QueueUrl: queueUrl, /* required */
    MaxNumberOfMessages: 10,
    VisibilityTimeout: 60,
    WaitTimeSeconds: 20 // Wait for messages to arrive
  };
  waitingSQS = true;
  sqs.receiveMessage(params, function(err, data) {
    if (err) {
      waitingSQS = false;
      console.log(err, err.stack); // an error occurred
    }
    else{
      waitingSQS = false;
      if ((typeof data.Messages !== 'undefined')&&(data.Messages.length !== 0)) {
        console.log('Received '+ data.Messages.length
          + ' messages from SQS queue.');           // successful response
        processMessages(data.Messages);
      }
      else {
        queueCounter = 60; // Queue empty back of for 60s
        console.log('SQS queue empty, waiting for '+ queueCounter + 's.');
      }
    }
  });
}
```

Now add the function to asynchronously process and delete messages from the queue.

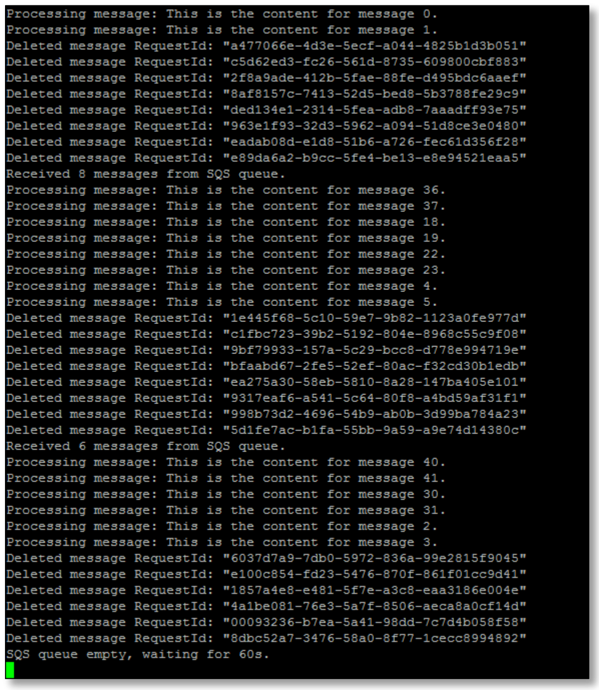
```
// Process and delete messages from queue
function processMessages(messagesSQS){
  async.each(messagesSQS, function (content) {
    console.log('Processing message: '+ content.Body); // Do something with the message
    var params = {
      QueueUrl: queueUrl, /* required */
      ReceiptHandle: content.ReceiptHandle /* required */
    };
    sqs.deleteMessage(params, function(err, data) {
      if (err) console.log(err, err.stack); // an error occurred
      else {
        console.log('Deleted message RequestId: '
          + JSON.stringify(data.ResponseMetadata.RequestId));   // successful response
      }
    });
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run the application.

You will see the messages being processed and deleted from the queue after processing.

After the SQS WaitTimeSeconds of 20 seconds has expired the SQS queue empty message will appear.



```
Processing message: This is the content for message 0.
Processing message: This is the content for message 1.
Deleted message RequestId: "a477066e-4d3e-5ecf-a044-4825b1d3b051"
Deleted message RequestId: "c5d62ed3-fc26-561d-8735-609800cbf883"
Deleted message RequestId: "2f8a9ade-412b-5fae-88fe-d495bdc6aaef"
Deleted message RequestId: "8af8157c-7413-52d5-bed8-5b3788fe29c9"
Deleted message RequestId: "ded134e1-2314-5fea-adb8-7aaadff93e75"
Deleted message RequestId: "963e1f93-32d3-5962-a094-51d8ce3e0480"
Deleted message RequestId: "eadab08d-e1d8-51b6-a726-fec61d356f28"
Deleted message RequestId: "e89da6a2-b9cc-5fe4-be13-e8e94521eaa5"
Received 8 messages from SQS queue.
Processing message: This is the content for message 36.
Processing message: This is the content for message 37.
Processing message: This is the content for message 18.
Processing message: This is the content for message 19.
Processing message: This is the content for message 22.
Processing message: This is the content for message 23.
Processing message: This is the content for message 4.
Processing message: This is the content for message 5.
Deleted message RequestId: "1e445f68-5c10-59e7-9b82-1123a0fe977d"
Deleted message RequestId: "c1fbc723-39b2-5192-804e-8968c55c9f08"
Deleted message RequestId: "9bf79933-157a-5c29-bcc8-d778e994719e"
Deleted message RequestId: "bfaabd67-2fe5-52ef-80ac-f32cd30b1edb"
Deleted message RequestId: "ea275a30-58eb-5810-8a28-147ba405e101"
Deleted message RequestId: "9317eaf6-a541-5c64-80f8-a4bd59af31f1"
Deleted message RequestId: "998b73d2-4696-54b9-ab0b-3d99ba784a23"
Deleted message RequestId: "5d1fe7ac-b1fa-55bb-9a59-a9e74d14380c"
Received 6 messages from SQS queue.
Processing message: This is the content for message 40.
Processing message: This is the content for message 41.
Processing message: This is the content for message 30.
Processing message: This is the content for message 31.
Processing message: This is the content for message 2.
Processing message: This is the content for message 3.
Deleted message RequestId: "6037d7a9-7db0-5972-836a-99e2815f9045"
Deleted message RequestId: "e100c854-fd23-5476-870f-861f01cc9d41"
Deleted message RequestId: "1857a4e8-e481-5f7e-a3c8-eaa3186e004e"
Deleted message RequestId: "4a1be081-76e3-5a7f-8506-aeca8a0cf14d"
Deleted message RequestId: "00093236-b7ea-5a41-98dd-7c7d4b058f58"
Deleted message RequestId: "8dbc52a7-3476-58a0-8f77-1cecc8994892"
SQS queue empty, waiting for 60s.
```

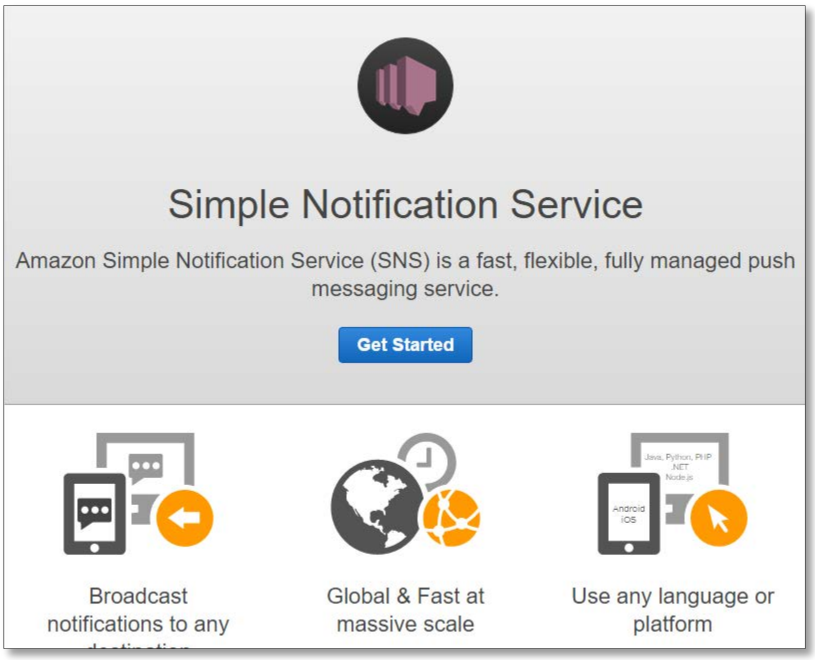
Press Ctrl + C to stop the application.

Subscribing an SQS Queue to an SNS Topic

In this section we will create and subscribe our application to an SNS topic. We will then use the NodeJS SDK to send SNS messages and then read, process and delete the messages from the SQS queue.

Now let's create an SNS topic.

Go to the SNS console.


Simple Notification Service

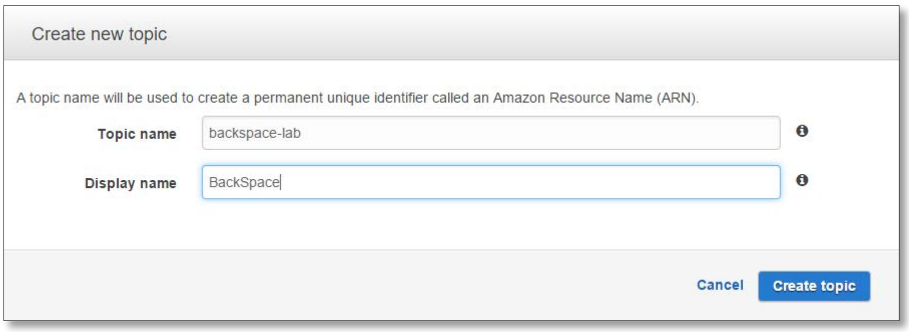
Amazon Simple Notification Service (SNS) is a fast, flexible, fully managed push messaging service.

Get Started
Broadcast notifications to any
Global & Fast at massive scale
Use any language or platform
Java, Python, PHP .NET Node.js
Android iOS

Click "Get Started"

Click "Create Topic"

Give it the topic name backspace-lab, and display name backspace

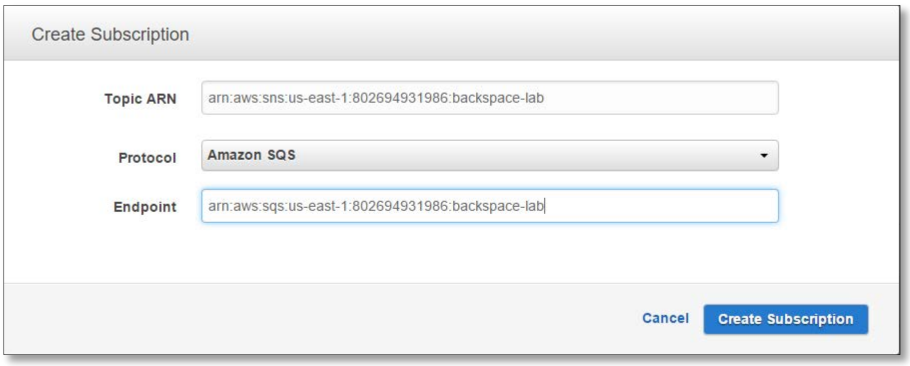


Create new topic

A topic name will be used to create a permanent unique identifier called an Amazon Resource Name (ARN).

Topic name	backspace-lab
Display name	BackSpace

Click “Create Topic”



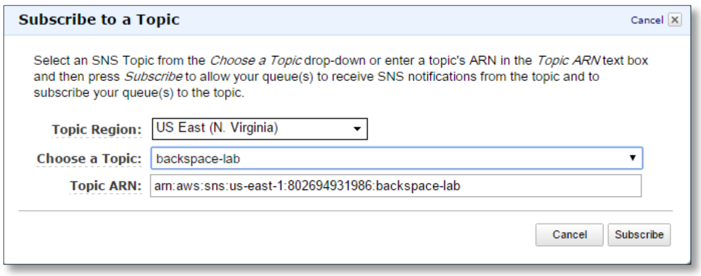
Create Subscription

Topic ARN	arn:aws:sns:us-east-1:802694931986:backspace-lab
Protocol	Amazon SQS
Endpoint	arn:aws:sqs:us-east-1:802694931986:backspace-lab

Now go back to the SQS console.

Select “Queue Actions” “Subscribe queue to SNS topic”

Select the topic and click “Subscribe”

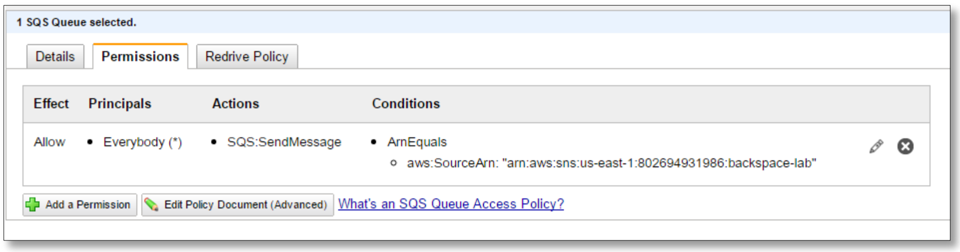


Subscribe to a Topic

Select an SNS Topic from the *Choose a Topic* drop-down or enter a topic's ARN in the *Topic ARN* text box and then press *Subscribe* to allow your queue(s) to receive SNS notifications from the topic and to subscribe your queue(s) to the topic.

Topic Region:	US East (N. Virginia)
Choose a Topic:	backspace-lab
Topic ARN:	arn:aws:sns:us-east-1:802694931986:backspace-lab

Now click on the permissions tab at the bottom of the screen to see the permission for SNS created.



1 SQS Queue selected.

Details
Permissions
Redrive Policy

Effect	Principals	Actions	Conditions
Allow	• Everybody (*)	• SQS:SendMessage	• ArnEquals ○ aws:SourceArn: "arn:aws:sns:us-east-1:802694931986:backspace-lab"

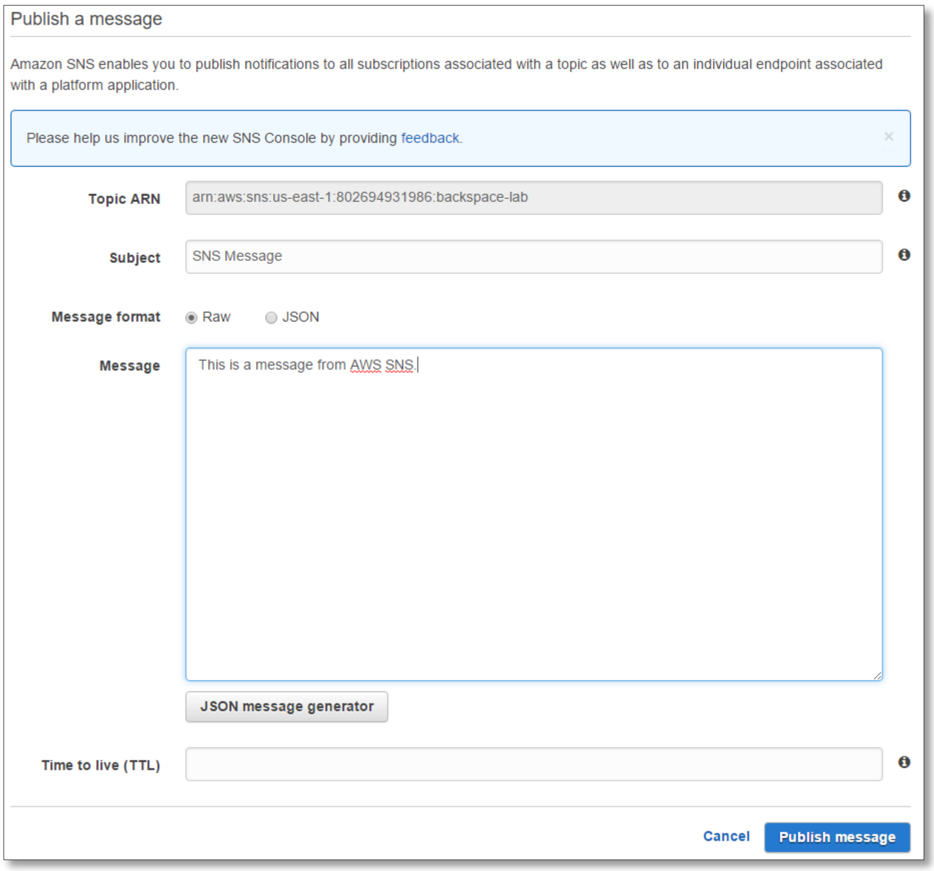
Add a Permission
Edit Policy Document (Advanced)
What's an SQS Queue Access Policy?

Now go back to the SNS console.

Select “Topics”

Select the topic and click “Publish to topic”

Create a message.



Publish a message

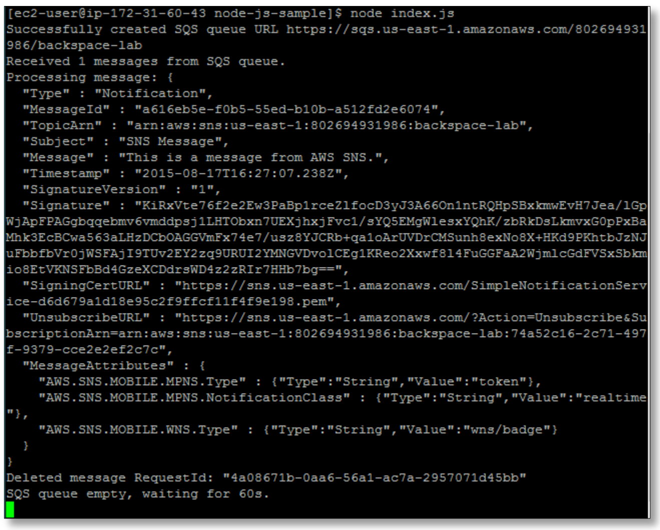
Amazon SNS enables you to publish notifications to all subscriptions associated with a topic as well as to an individual endpoint associated with a platform application.

Please help us improve the new SNS Console by providing feedback.

Topic ARN	arn:aws:sns:us-east-1:802694931986:backspace-lab
Subject	SNS Message
Message format	[✓ Raw] JSON
Message	This is a message from AWS SNS
Time to live (TTL)	

Now run your app again.

You will see the message has been delivered to SQS and processed by your app.



```
[ec2-user@ip-172-31-60-43 node-js-sample]$ node index.js
Successfully created SQS queue URL https://sqs.us-east-1.amazonaws.com/802694931
986/backspace-lab
Received 1 messages from SQS queue.
Processing message: {
  "Type" : "Notification",
  "MessageId" : "a616eb5e-f0b5-55ed-b10b-a512fd2e6074",
  "TopicArn" : "arn:aws:sns:us-east-1:802694931986:backspace-lab",
  "Subject" : "SNS Message",
  "Message" : "This is a message from AWS SNS.",
  "Timestamp" : "2015-08-17T16:27:07.238Z",
  "SignatureVersion" : "1",
  "Signature" : "KiRxVte76f2e2Ew3PaBp1rceZlfocD3yJ3A66On1ntRQHpSBxkmwEvH7Jea/1Gp
WjApFPAGgbqqebmv6vmddpsj1LHTObxn7UEXjhxjFvc1/sYQ5EMgWlesxYQhK/zbRkDsLkmvxG0pPxBa
Mhk3EcBCwa563aLHzDCbOAGGVmFx74e7/usz8YJCRb+qa1oArUVDrCMSunh8exNo8X+HKd9PKhtbJzNJ
uFbbfbVr0jWSFAjI9TUv2EY2zq9URUI2YMNGVDvolCEg1KReo2Xxwf814FuGGFaA2WjmlcGdFVSxSbkm
io8EtVKNSFbBd4GzeXCDdrsWD4z2zRIr7HHb7bg==",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationServ
ice-d6d679a1d18e95c2f9ffcf11f4f9e198.pem",
  "UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&Su
bscriptionArn=arn:aws:sns:us-east-1:802694931986:backspace-lab:74a52c16-2c71-497
f-9379-cce2e2ef2c7c",
  "MessageAttributes" : {
    "AWS.SNS.MOBILE.MPNS.Type" : {"Type":"String","Value":"token"},
    "AWS.SNS.MOBILE.MPNS.NotificationClass" : {"Type":"String","Value":"realtime
"},
    "AWS.SNS.MOBILE.WNS.Type" : {"Type":"String","Value":"wns/badge"}
  }
}
Deleted message RequestId: "4a08671b-0aa6-56a1-ac7a-2957071d45bb"
SQS queue empty, waiting for 60s.
```

Now we will send an SNS message using the NodeJS SDK

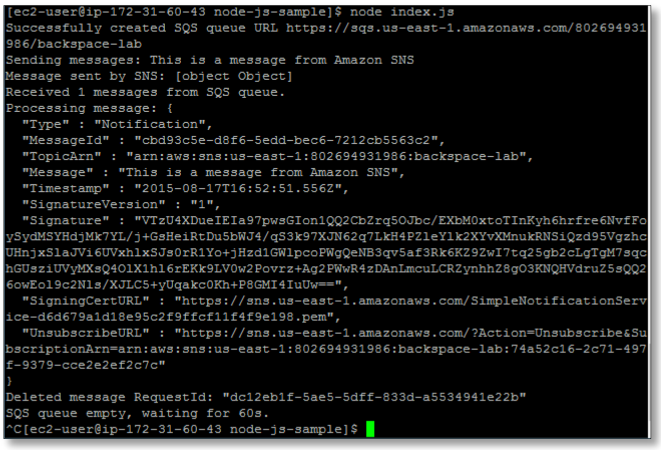
Replace the createMessages code with (make sure to replace YOUR_SNS_ARN):

```
// Create an SNS messages
var sns = new AWS.SNS();

function createMessages(){
  var message = 'This is a message from Amazon SNS';
  console.log('Sending messages: '+ message);
  sns.publish({
    Message: message,
    TargetArn: ‘YOUR_SNS_ARN’  }, function(err, data) {
    if (err) {
      console.log(err.stack);
    }
    else{
      console.log('Message sent by SNS: '+ data);
    }
  });
}
```

Click Ctrl + S to save to the EC2 instance.

Now run index.js again



```
[ec2-user@ip-172-31-60-43 node-js-sample]$ node index.js
Successfully created SQS queue URL https://sqs.us-east-1.amazonaws.com/802694931
986/backspace-lab
Sending messages: This is a message from Amazon SNS
Message sent by SNS: [object Object]
Received 1 messages from SQS queue.
Processing message: {
  "Type" : "Notification",
  "MessageId" : "cbd93c5e-d8f6-5edd-bec6-7212cb5563c2",
  "TopicArn" : "arn:aws:sns:us-east-1:802694931986:backspace-lab",
  "Message" : "This is a message from Amazon SNS",
  "Timestamp" : "2015-08-17T16:52:51.556Z",
  "SignatureVersion" : "1",
  "Signature" : "VTzU4XDueIEIa97pwsGIon1QQ2CbZrq5OJbc/EXbM0xtoTInKyh6hrfre6NvfFo
ySydMSYHdjMk7YL/j+GsHeiRtDu5bWJ4/qS3k97XJN62q7LkH4PZleYlk2XYvXMnukRNSiQzd95Vgzhc
UHnjxSlaJVi6UVxhlxSJs0rR1Yo+jHzd1GWlpcoPWgQeNB3qv5af3Rk6KZ9ZwI7tq25gb2cLgTgM7sqc
hGUsziUVyMXsQ4OlX1hl6rEKk9LV0w2Povrz+Ag2PWwR4zDAnLmcuLCRZynhhZ8gO3KNQHVdruZ5sQQ2
6owEol9c2Nls/XJLC5+yUqakc0Kh+P8GMI4IuUw==",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationServ
ice-d6d679a1d18e95c2f9ffcf11f4f9e198.pem",
  "UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&Su
bscriptionArn=arn:aws:sns:us-east-1:802694931986:backspace-lab:74a52c16-2c71-497
f-9379-cce2e2ef2c7c"
}
Deleted message RequestId: "dc12eb1f-5ae5-5dff-833d-a5534941e22b"
SQS queue empty, waiting for 60s.
^C[ec2-user@ip-172-31-60-43 node-js-sample]$
```

Press Ctrl + C to stop the application.