aws CERTIFIED Developer Associate

aws CERTIFIED Solutions Architect Associate

aws CERTIFIED SysOps Administrator Associate

lab

lab title

Programming Amazon SQS and SNS using the AWS NodeJS SDK

V1.04

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:

- 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.

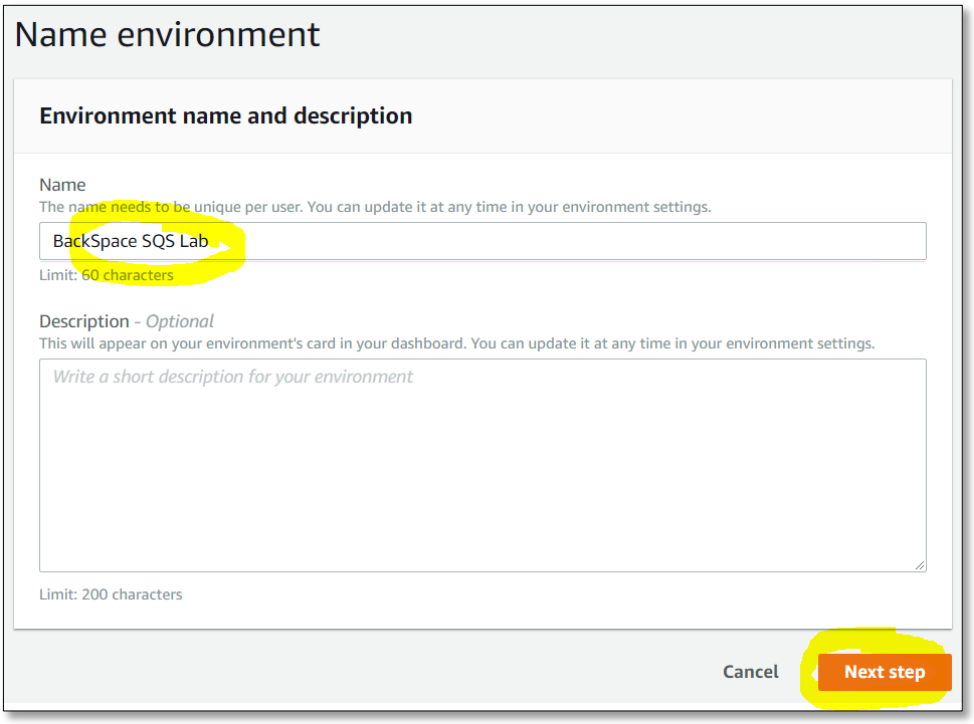
Please note these lab notes have been updated for the Cloud9 IDE due to problems with the Atom Remote-Edit package. You can still use an IDE such as Atom if you like but, you will need to upload files to your EC2 instance using FileZilla.

Go to *Services - Cloud9* from the console

Click *Create environment*

Give your environment a name

Click *Next step*

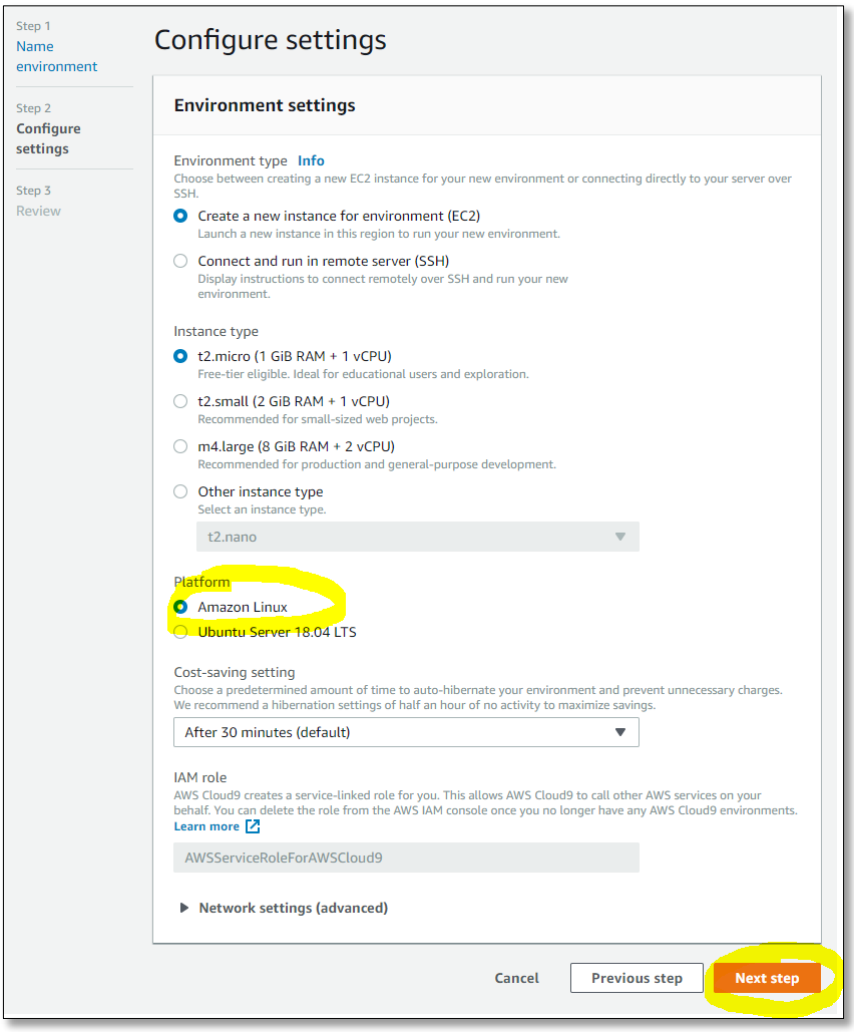


Leave default settings

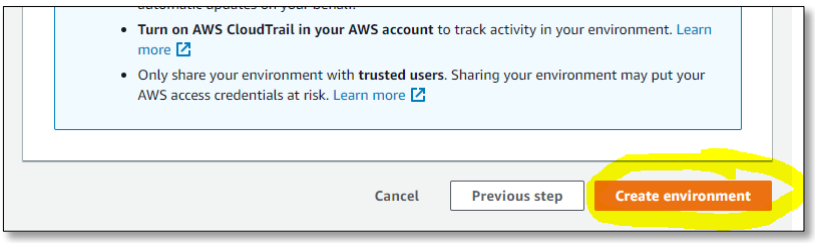
Click *Next step*

Select *Amazon Linux*

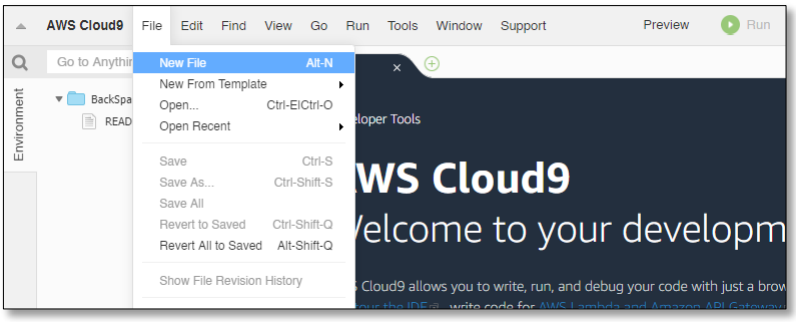
Click *Next Step*



Click *Create environment*



When your environment is ready select *File-New File*



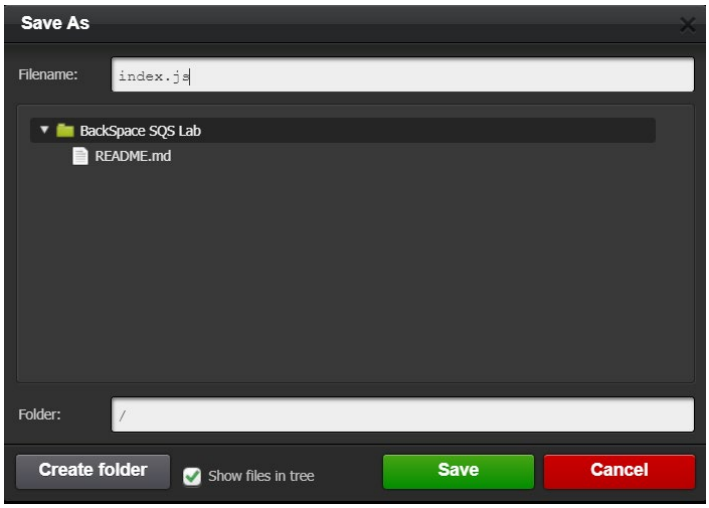
Copy the following code and paste into the new file (*ctrl-v* to paste):

```
// 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 

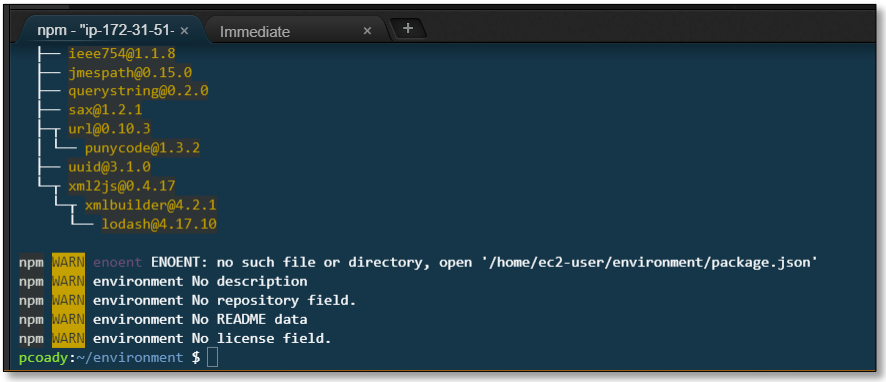
+ 

to save the file as *index.js*



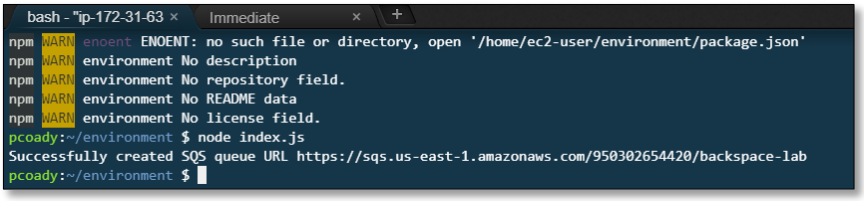
From the Bash console at the bottom of the screen enter:

npm install aws-sdk

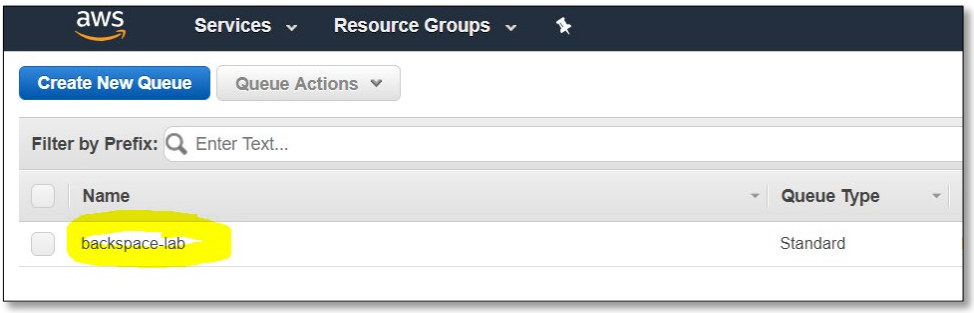


Now that you have installed the AWS SDK you can run the app

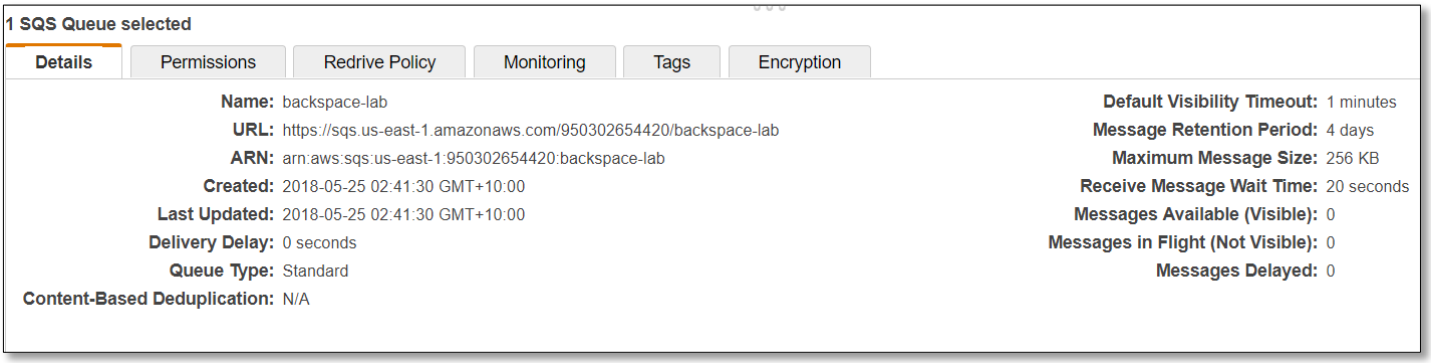
node index.js



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



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



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.

Important: These lab notes have been updated for the new built in async features of NodeJS version 8+. The code is slightly different to that on the video and does not require an npm package and will not work with the npm async package.

This part of the lab will require the new async capabilities of NodeJS version 8+

You can check the current version is 8 or higher:

```
node --version
```

```
pcoady:~/environment $ node index.js
Successfully created SQS queue URL https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab
pcoady:~/environment $ node --version
v8.16.0
pcoady:~/environment $
```

Sending Messages with *sendMessage*

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);
    }
  });
```

Now create the createMessages function:

```
// Create 50 SQS messages
async 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
  for (const message of messages){
    console.log('Sending message: '+ message)
    params = {
      MessageBody: message, /* required */
      QueueUrl: queueUrl /* required */
    };
    await sqs.sendMessage(params, function(err, data) { // Wait until callback
        if (err) console.log(err, err.stack); // an error occurred
        else     console.log(data);           // successful response
    });
  }
}
```

Click 

+ 

to save to the EC2 instance.

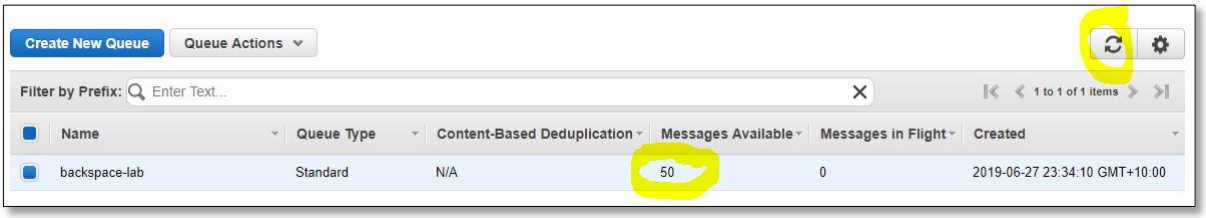
Now run index.js

```
node index.js
```

It has now created and sent 50 messages.

```
bash - "ip-172-31-83 ×   Immediate ×   +
{ ResponseMetadata: { RequestId: 'ed66433f-1cd8-5c58-a648-16b1d6b56bf0' },
  MD5OfMessageBody: 'e9e39fbf0de9d6e1ec9116541090bad0',
  MessageId: 'bd6ce544-10b4-4d0a-a3c1-95f017009e5a' }
{ ResponseMetadata: { RequestId: 'da99f5ee-c8de-5f2e-8edb-a06002b37488' },
  MD5OfMessageBody: '136177f8d8ac551fdd1fed1b1e3c3971',
  MessageId: 'b573ab98-07a1-4952-8111-f55509256f33' }
{ ResponseMetadata: { RequestId: '6235201e-8e94-5d72-a083-759044f900af' },
  MD5OfMessageBody: 'e251dd841de1574a70e7dc7decabd159',
  MessageId: '0477be4b-f85d-4523-b9a5-c14e629ae572' }
{ ResponseMetadata: { RequestId: 'f67588a7-68a4-5a55-86ae-dd29a2c93601' },
  MD5OfMessageBody: '8282f267fbfde4af63c6d16e90dd0681',
  MessageId: 'c88752a9-63f2-48ae-b81c-fdc068b05afd' }
{ ResponseMetadata: { RequestId: 'd81491b8-31a7-583f-ad3a-c436fbcaf38f' },
  MD5OfMessageBody: '21b03464efd9e89b744a9f0b30b31edc',
  MessageId: 'b26d1bc6-9698-4734-bcb9-056d4fa71bd7' }
{ ResponseMetadata: { RequestId: 'a8c04f35-2b33-554e-b92a-1abbe15e6831' },
  MD5OfMessageBody: '0223852ab0ed50bb398f43dafbc787db',
  MessageId: '2f0796b7-c431-4536-9220-a5b70a3527ce' }
pcoady:~/environment $
```

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



Increasing Throughput with *sendMessageBatch*

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
async 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) + '.';
      }
    }

    // Asynchronously deliver messages to SQS queue
    for (const message of messages){
      console.log('Sending message: '+ message)
      params = {
        Entries: [],
        QueueUrl: queueUrl /* required */
      };
      for (var b=0; b<10; b++){
        params.Entries.push({
          MessageBody: message [b],
          Id: 'Message'+ (messages.indexOf(message)*10+b)
        });
      }
      await sqs.sendMessageBatch(params, function(err, data) { // Wait until callback
          if (err) console.log(err, err.stack); // an error occurred
          else     console.log(data);           // successful response
      });
    }
  }
```

Click 

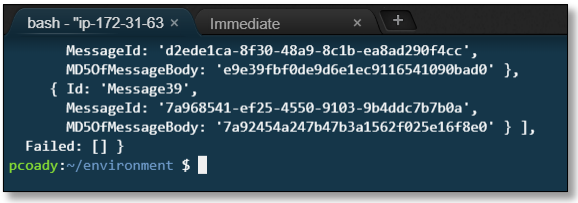
+ 

to save to the EC2 instance.

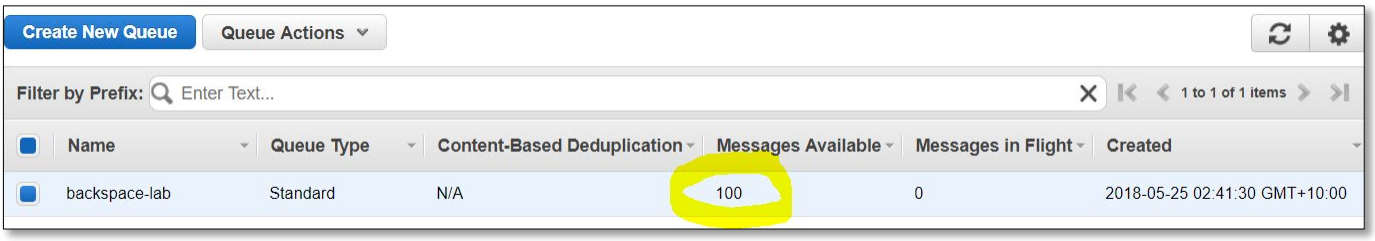
Now run index.js

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

You will also see an empty array returned for failed messages.



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



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 let's 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.');
          }
        }
      });
    }
```

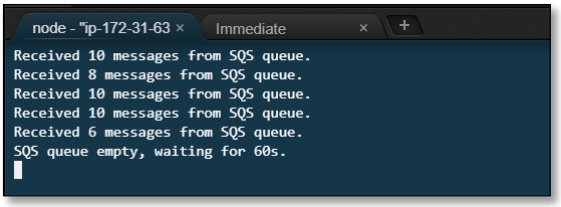
Click 

+ 

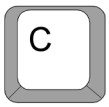
to save to the EC2 instance.

Now run index.js

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



Press 

+ 

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
async function processMessages(messagesSQS){
    for (const item of messagesSQS){
        await console.log('Processing message: '+ item.Body); // Do something with the
message
        var params = {
            QueueUrl: queueUrl, /* required */
            ReceiptHandle: item.ReceiptHandle /* required */
        }
        await sqs.deleteMessage(params, function(err, data) { // Wait until callback
            if (err) console.log(err, err.stack); // an error occurred
            else {
                console.log('Deleted message RequestId: '
                  + JSON.stringify(data.ResponseMetadata.RequestId));   // successful
response
            }
        })
    }
}
```

Click 

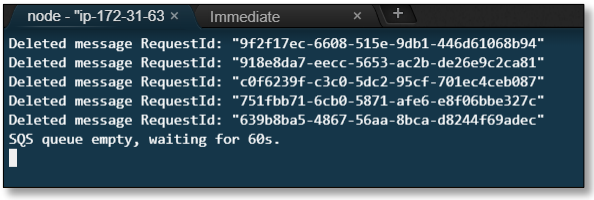
+ 

to save to the EC2 instance.

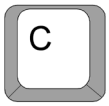
Now run the application.

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.



Press 

+ 

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.

We will sending messages to the queue from the SNS service. We won't need to call createMessages.

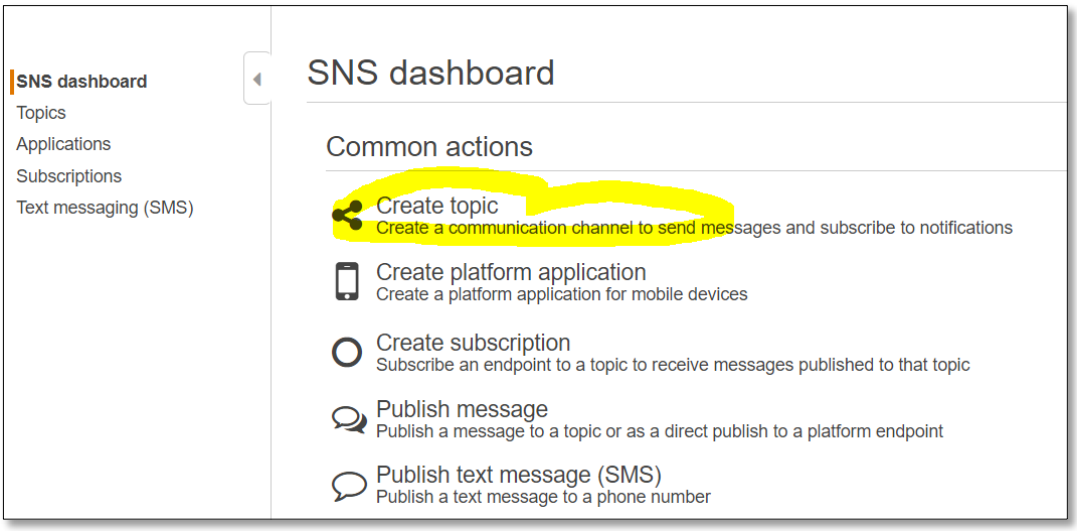
Comment out the call to createMessages

```
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(data.QueueUrl);
    }
  });
```

Now let's create an SNS topic.

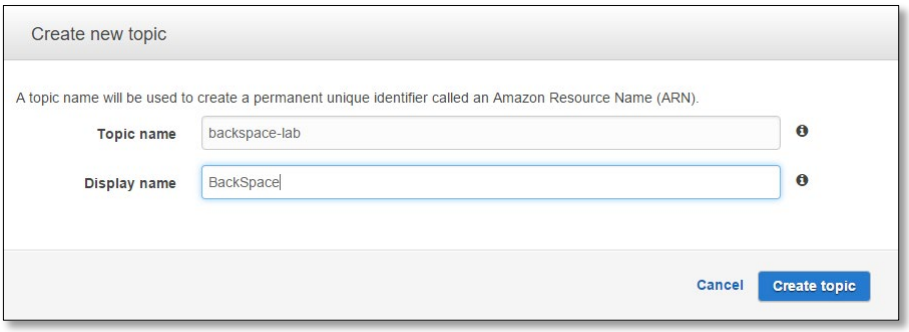
Go to the SNS console.

Click 'Create Topic"



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

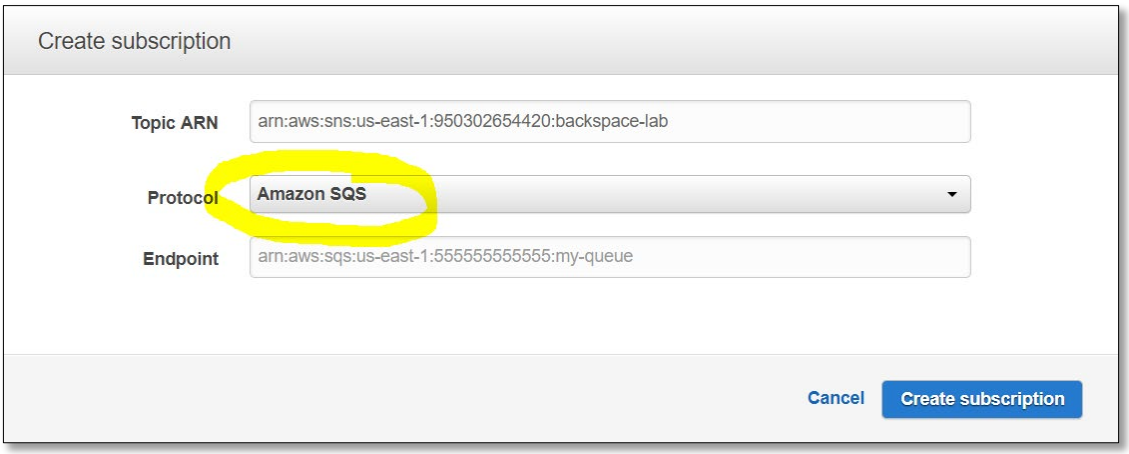
Click 'Create Topic"



Click 'Create subscription"

Select 'Amazon SQS" for protocol

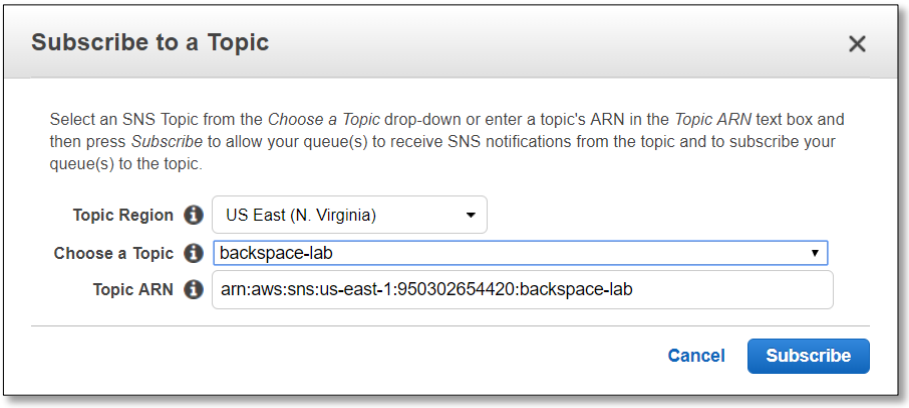
Copy and paste the SQS ARN from the SQS console for the endpoint.



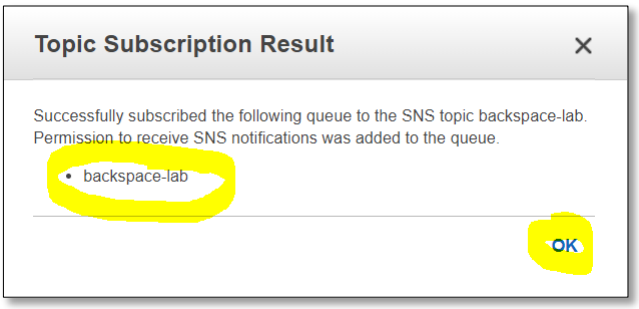
Now go back to the SQS console.

Select 'Queue Actions" 'Subscribe queue to SNS topic"

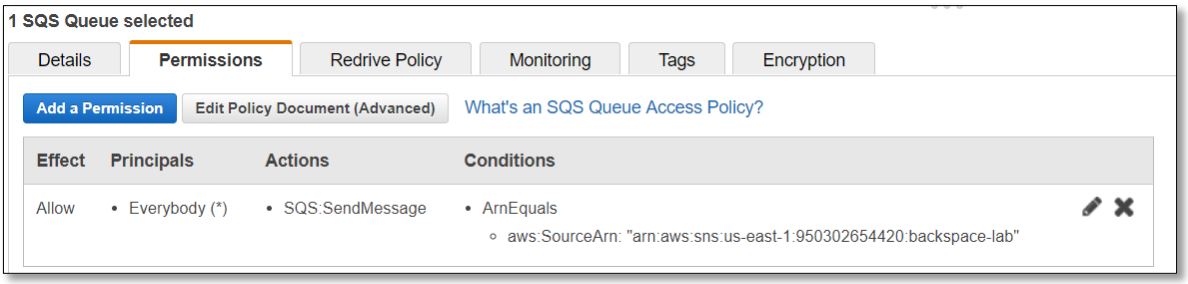
Select the topic and click 'Subscribe"



Click 'Subscribe"



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

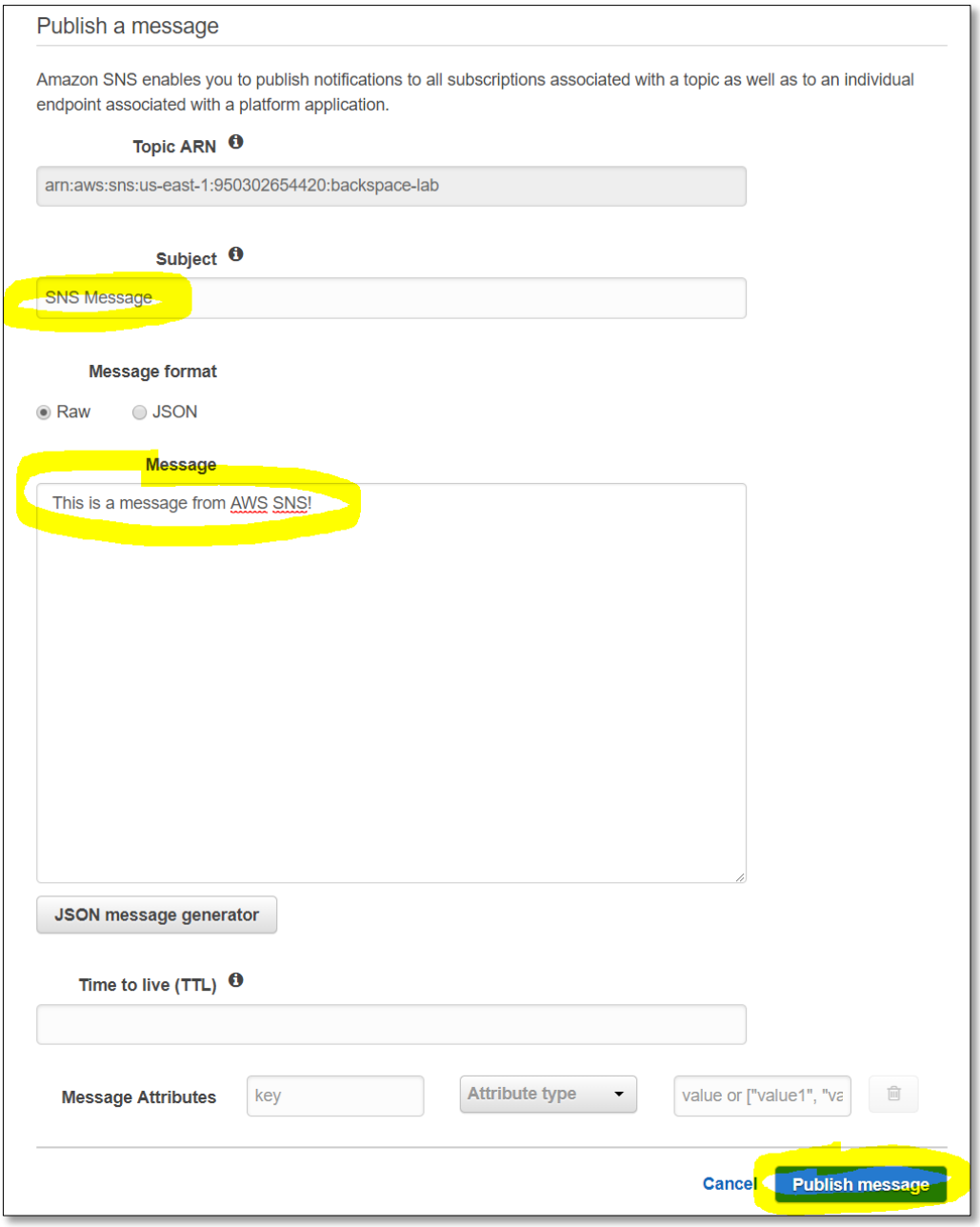


Now go back to the SNS console.

Select 'Topics"

Select the topic and click 'Publish to topic"

Create a message.



Now run your app again.

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

```
pcoady:~/environment $ node index.js
Successfully created SQS queue URL https://sqs.us-east-1.amazonaws.com/950302654420/backspace-lab
Received 1 messages from SQS queue.
Processing message: {
  "Type" : "Notification",
  "MessageId" : "3e0f3bc9-40e9-51f0-9018-68484e017ba2",
  "TopicArn" : "arn:aws:sns:us-east-1:950302654420:backspace-lab",
  "Subject" : "Another SNS message",
  "Message" : "This message was sent by SNS!",
  "Timestamp" : "2018-05-24T19:04:08.341Z",
  "SignatureVersion" : "1",
  "Signature" : "i2WNXVCsaCnGPfCha0YwrcSfmrsFp2KpKnGkMqu0HGeC9NgNKeDJhad+Ac8FIvSGQJTY5hNqDYLp9JL+1uZt29iVXuGI+rhFC
7fiqnghzg7Oq/clmsF6sdqySNoYwinaQ7jHWXDAr9QZ8m5xBIiCf4LPejYRbZmLDZubNBzModcYbgJMEgI/vy0rWMEEmDT1TcEIy1/81juJnyi4rMS
+QlTfhItOyWFqNIL9CGGGj84C0qRvwwCUTqRCy5hBM1qrdg7lsfj7cSusRVkuhic5HTxpT4rUTb4UEXC4UIeyRKPLXPdzi1FTcVPS6AJQJJ2VrFPbg
TwzHTqyh4GG0o0+xQ==",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationService-eaea6120e66ea12e88dcd8bcbddca7
52.pem",
  "UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-
1:950302654420:backspace-lab:1ad636bf-48a5-417c-bb7a-a9cd048b1873",
  "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: "22d6b3cf-82e7-5bde-afcd-e04e8de57480"
```

Now we will send an SNS message using the NodeJS SDK

Uncomment the createMessages call from createQueue

```
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(data.QueueUrl);
    }
  });
```

Replace the createMessages code with (make sure to replace YOUR_SNS_ARN with the SNS topic 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: 'arn:aws:sns:us-east-1:950302654420:backspace-lab'  }, function(err,
data) {
    if (err) {
      console.log(err.stack);
    }
    else{
      console.log('Message sent by SNS: '+ data);
    }
  });
}
```

Click 

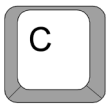
+ 

to save to the EC2 instance.

Now run index.js again

```
pcoady:~/environment $ node index.js
Successfully created SQS queue URL https://sqs.us-east-1.amazonaws.com/950302654420/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" : "0c30e948-59d4-5e97-9b08-bac9490ba13d",
  "TopicArn" : "arn:aws:sns:us-east-1:950302654420:backspace-lab",
  "Message" : "This is a message from Amazon SNS",
  "Timestamp" : "2018-05-24T19:12:31.074Z",
  "SignatureVersion" : "1",
  "Signature" : "wFbXqUvcKCbwn8sO+qpaFmKNDiQHJGMy7yKsajIKXvjUXt+ryCTuWt98r0BENdcjzyK0ruijOw/0ENz3a+X1b+E/kFqBlE40H
ui0N2MeLmCvV/FUB2VfbfzInH3gZlW0g7xPpUHxUo+sIVv6RRYQpwcFho95LVDVUlQa2L7BK161b2a0saAkCczYxcV/rG4YVuH5qv+VmEupNrJxwfG
jSEjiLQ67ow+fU8g1sZLmW6ZnIH2tJrcBv/pxk2Z2rieroXEqWpWPMxwvrfNxGoFJoJcKrBAWPJ5JaeOegowOcPYDA3vrz33hyve4J/ZTcAJW3TUNg
/AO8cTrQ8ghExAKUA==",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationService-eaea6120e66ea12e88dcd8bcbddca7
52.pem",
  "UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-
1:950302654420:backspace-lab:1ad636bf-48a5-417c-bb7a-a9cd048b1873"
}
Deleted message RequestId: "32033a3d-12da-5d1a-8f07-69bc023a196c"
SQS queue empty, waiting for 60s.
```

Press 

+ 

to stop the application.