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 SDK

V1.06

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.

If you get stuck, completed code for the lab can be downloaded from:

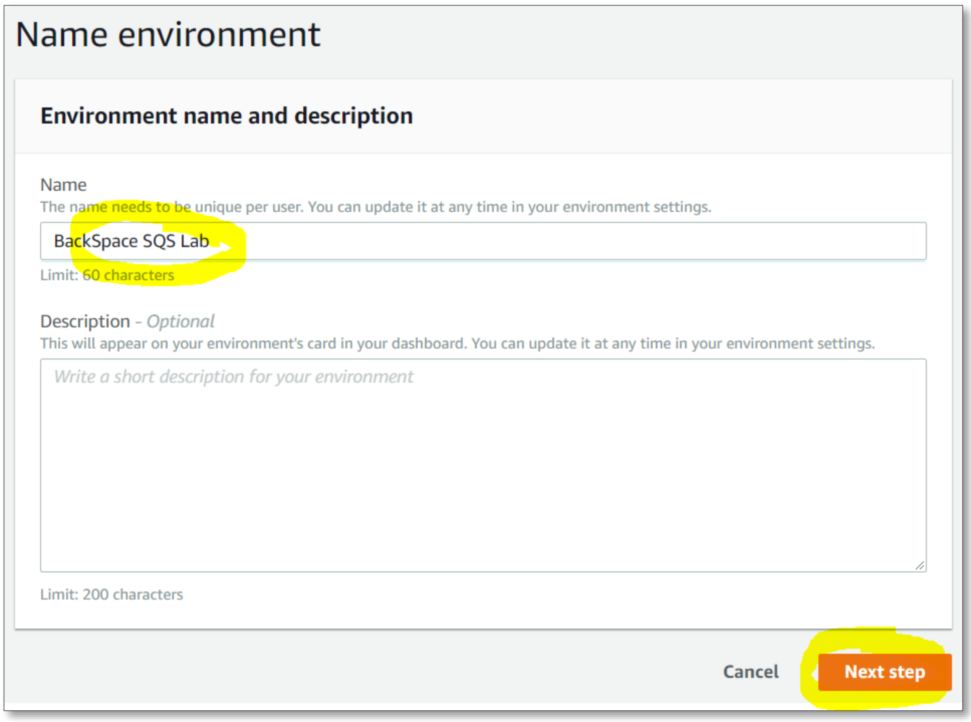
https://github.com/backspace-academy/sqs-nodejs

Go to *Services - Cloud9* from the console

Click *Create environment*

Give your environment a name

Click *Next step*



Name environment

Environment name and description

Name
The name needs to be unique per user. You can update it at any time in your environment settings.
BackSpace SQS Lab
Limit: 60 characters

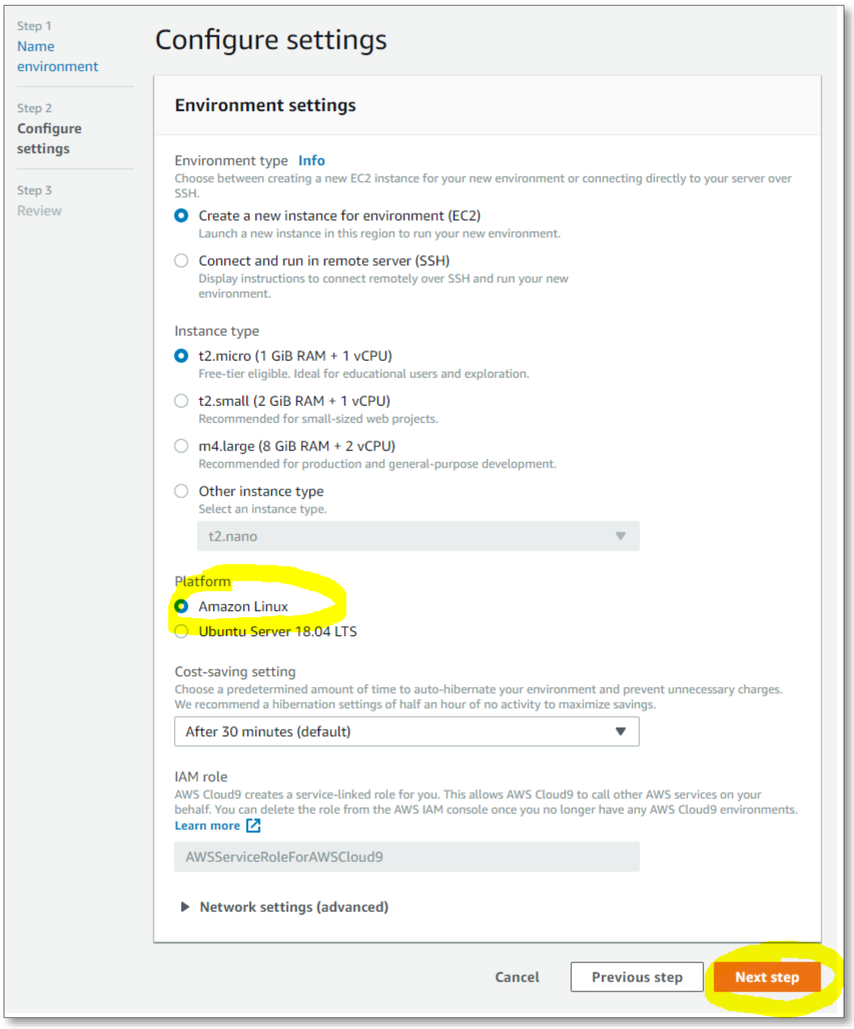
Description - *Optional*
This will appear on your environment's card in your dashboard. You can update it at any time in your environment settings.
Write a short description for your environment
Limit: 200 characters

Leave default settings

Click *Next step*

Select *Amazon Linux*

Click *Next Step*



Step 1
Name environment

Step 2
Configure settings

Step 3
Review

Configure settings

Environment settings

Environment type Info
Choose between creating a new EC2 instance for your new environment or connecting directly to your server over SSH.

- ● Create a new instance for environment (EC2)
  Launch a new instance in this region to run your new environment.
- ○ Connect and run in remote server (SSH)
  Display instructions to connect remotely over SSH and run your new environment.

Instance type

- ● t2.micro (1 GiB RAM + 1 vCPU)
  Free-tier eligible. Ideal for educational users and exploration.
- ○ t2.small (2 GiB RAM + 1 vCPU)
  Recommended for small-sized web projects.
- ○ m4.large (8 GiB RAM + 2 vCPU)
  Recommended for production and general-purpose development.
- ○ Other instance type
  Select an instance type.
  t2.nano

Platform

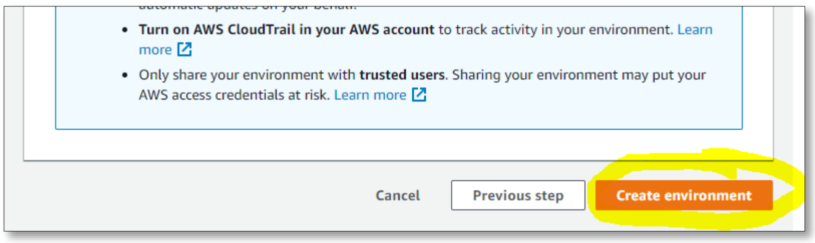
- ● Amazon Linux
- ○ Ubuntu Server 18.04 LTS

Cost-saving setting
Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a hibernation settings of half an hour of no activity to maximize savings.
After 30 minutes (default)

IAM role
AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. Learn more
AWSServiceRoleForAWSCloud9

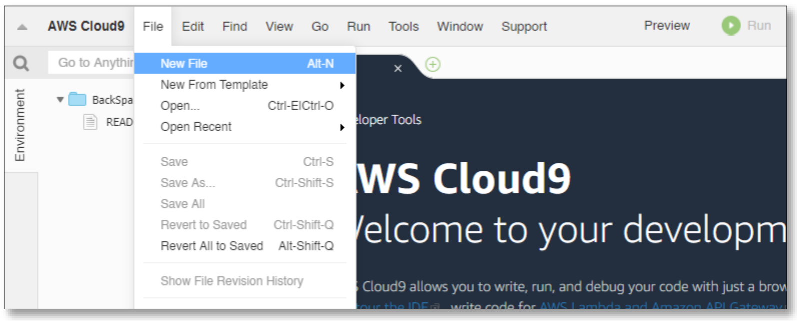
▶ Network settings (advanced)

Click *Create environment*



- **Turn on AWS CloudTrail in your AWS account** to track activity in your environment. Learn more
- Only share your environment with **trusted users**. Sharing your environment may put your AWS access credentials at risk. Learn more

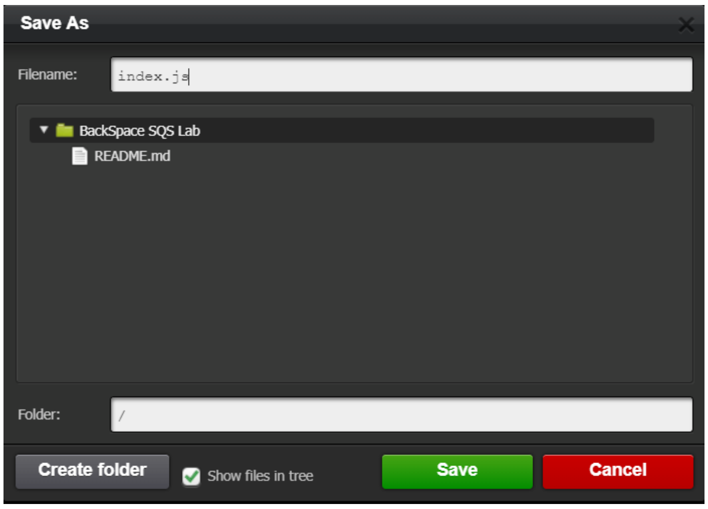
When your environment is ready select *File-New File*



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 Ctrl + S to save the file as *index.js*



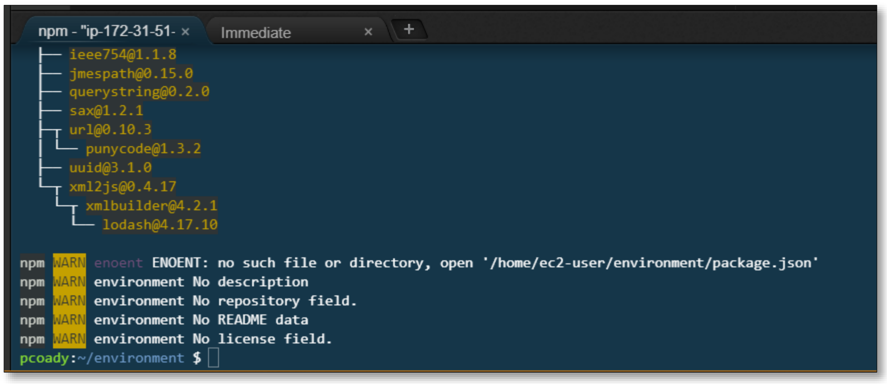
Filename: index.js

```
▾ BackSpace SQS Lab
    README.md
```

Folder: /

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

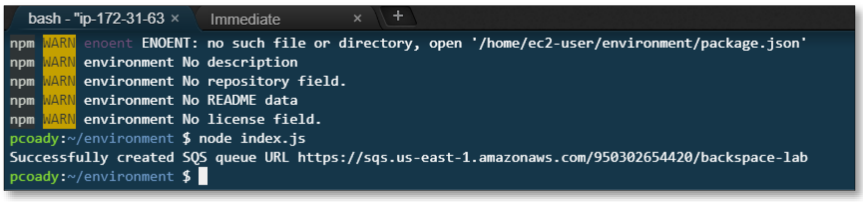
npm install aws-sdk



```
├── ieee754@1.1.8
├── jmespath@0.15.0
├── querystring@0.2.0
├── sax@1.2.1
├─┬ url@0.10.3
│ └── punycode@1.3.2
├── uuid@3.1.0
└─┬ xml2js@0.4.17
  └─┬ xmlbuilder@4.2.1
    └── lodash@4.17.10

npm WARN enoent ENOENT: no such file or directory, open '/home/ec2-user/environment/package.json'
npm WARN environment No description
npm WARN environment No repository field.
npm WARN environment No README data
npm WARN environment No license field.
pcoady:~/environment $
```

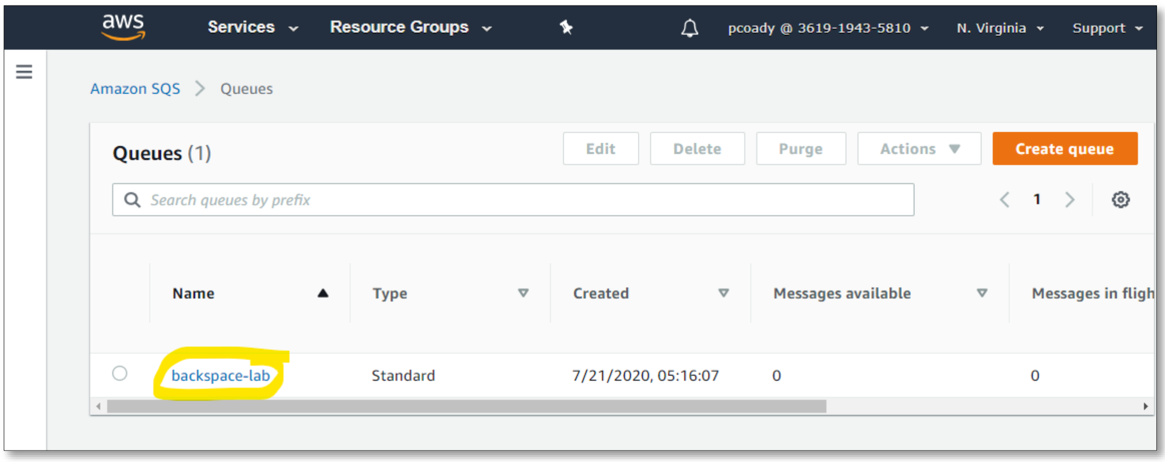
Now that you have installed the AWS SDK you can run the app

node index.js



```
npm WARN enoent ENOENT: no such file or directory, open '/home/ec2-user/environment/package.json'
npm WARN environment No description
npm WARN environment No repository field.
npm WARN environment No README data
npm WARN environment No license field.
pcoady:~/environment $ node index.js
Successfully created SQS queue URL https://sqs.us-east-1.amazonaws.com/950302654420/backspace-lab
pcoady:~/environment $
```

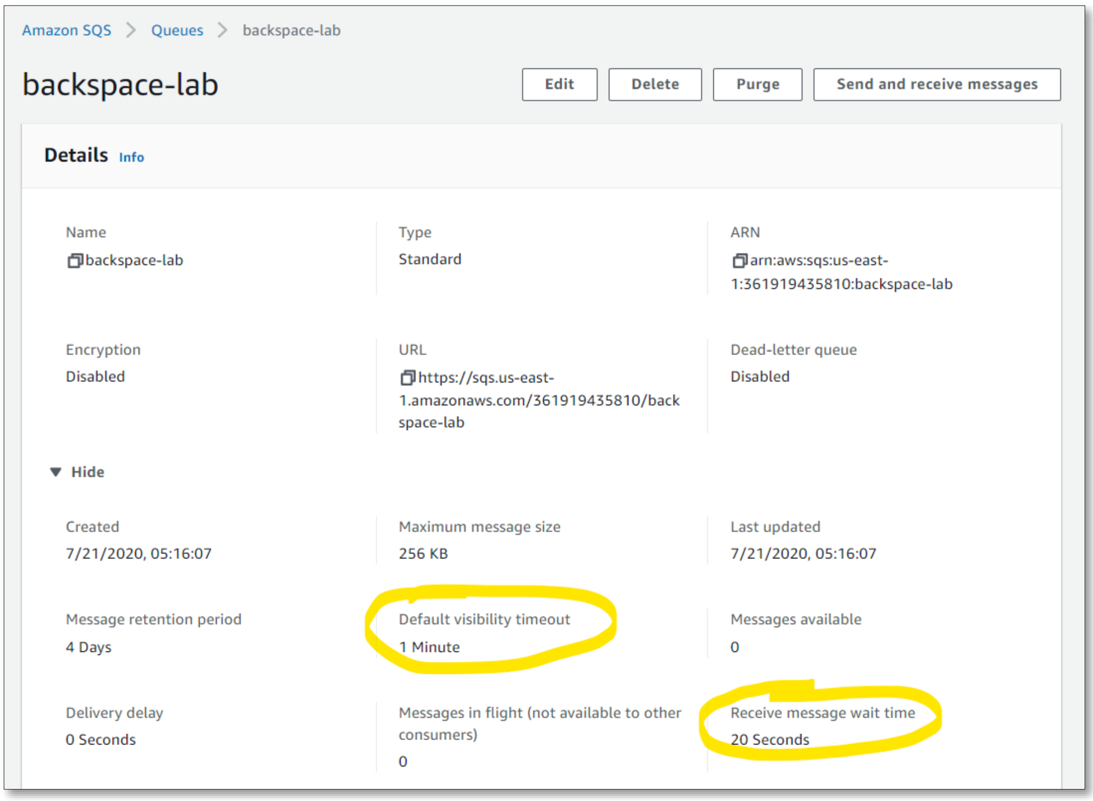
Now go to the SQS console and see your newly created SQS queue



Queues (1)

Name	Type	Created	Messages available	Messages in fligh
backspace-lab	Standard	7/21/2020, 05:16:07	0	0

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



backspace-lab

Details

Name backspace-lab	Type Standard	ARN arn:aws:sqs:us-east-1:361919435810:backspace-lab
Encryption Disabled	URL https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab	Dead-letter queue Disabled
Created 7/21/2020, 05:16:07	Maximum message size 256 KB	Last updated 7/21/2020, 05:16:07
Message retention period 4 Days	Default visibility timeout 1 Minute	Messages available 0
Delivery delay 0 Seconds	Messages in flight (not available to other consumers) 0	Receive message wait time 20 Seconds

Creating an SQS Queue using the AWS Python SDK

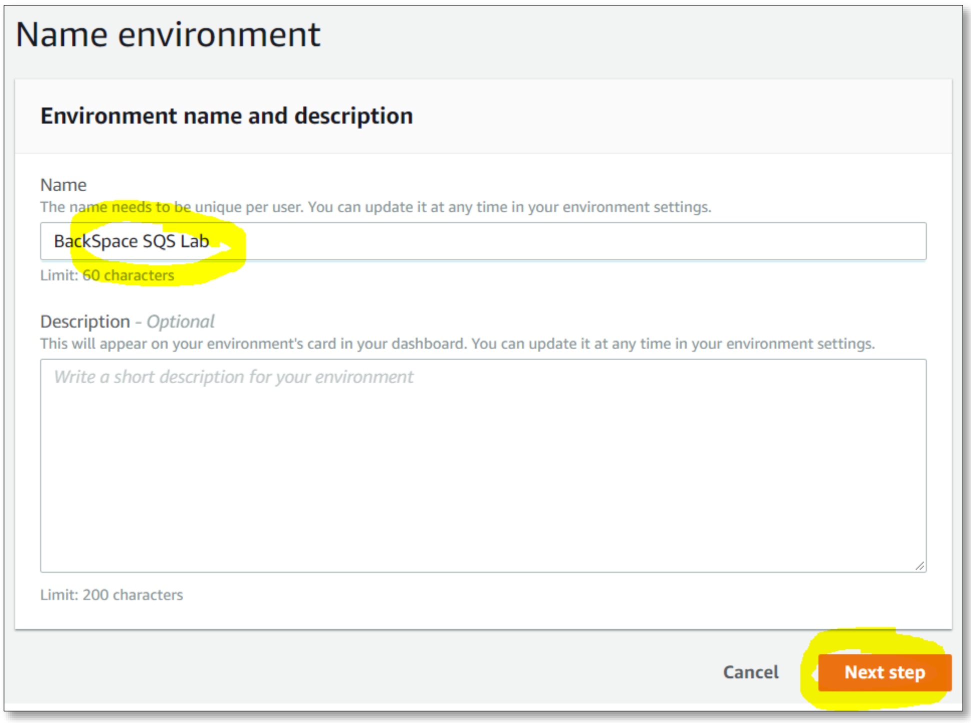
In this section we will use the AWS SDK for Python (Boto3) to create an SQS Queue.

Go to *Services - Cloud9* from the console

Click *Create environment*

Give your environment a name

Click *Next step*



Name environment

Environment name and description

Name
The name needs to be unique per user. You can update it at any time in your environment settings.
BackSpace SQS Lab
Limit: 60 characters

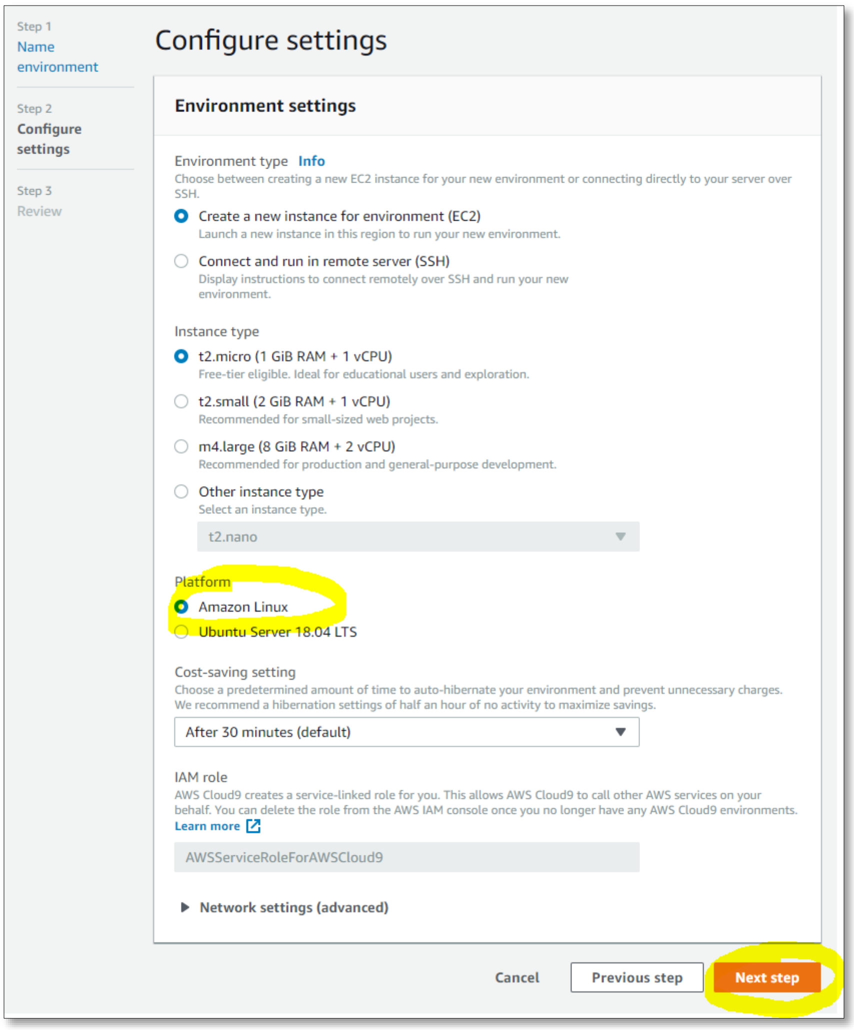
Description - *Optional*
This will appear on your environment's card in your dashboard. You can update it at any time in your environment settings.
Write a short description for your environment
Limit: 200 characters

Leave default settings

Click *Next step*

Select *Amazon Linux*

Click *Next Step*



Step 1
Name environment
Step 2
Configure settings
Step 3
Review

Configure settings

Environment settings

Environment type Info
Choose between creating a new EC2 instance for your new environment or connecting directly to your server over SSH.
- [✓ Create a new instance for environment (EC2)] Launch a new instance in this region to run your new environment.
- Connect and run in remote server (SSH) Display instructions to connect remotely over SSH and run your new environment.

Instance type
- [✓ t2.micro (1 GiB RAM + 1 vCPU)] Free-tier eligible. Ideal for educational users and exploration.
- t2.small (2 GiB RAM + 1 vCPU) Recommended for small-sized web projects.
- m4.large (8 GiB RAM + 2 vCPU) Recommended for production and general-purpose development.
- Other instance type Select an instance type. t2.nano

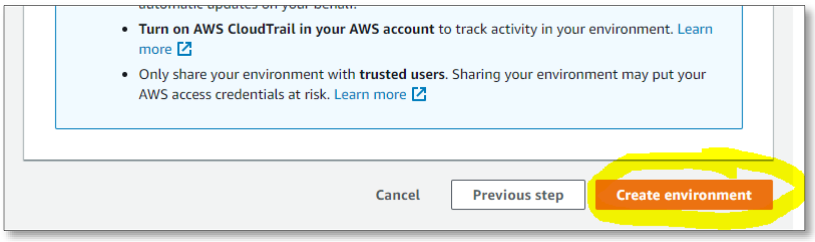
Platform
- [✓ Amazon Linux]
- Ubuntu Server 18.04 LTS

Cost-saving setting
Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a hibernation settings of half an hour of no activity to maximize savings.
After 30 minutes (default)

IAM role
AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. Learn more
AWSServiceRoleForAWSCloud9

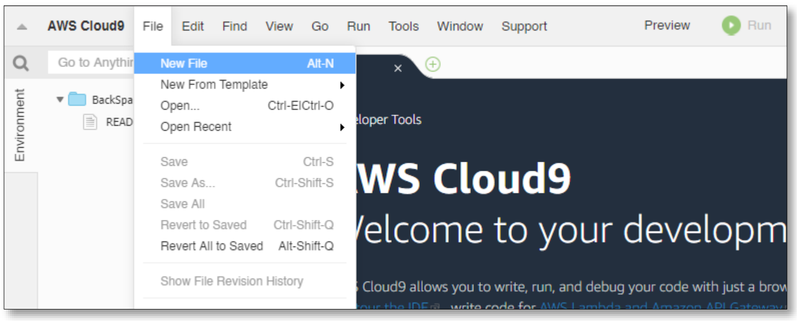
Network settings (advanced)

Click *Create environment*



- **Turn on AWS CloudTrail in your AWS account** to track activity in your environment. Learn more
- Only share your environment with **trusted users**. Sharing your environment may put your AWS access credentials at risk. Learn more

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

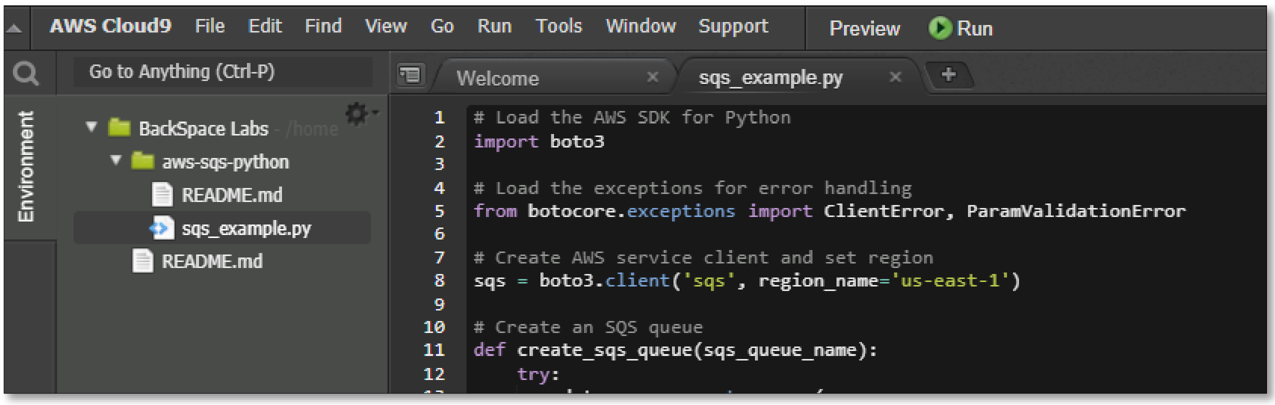


From the Bash console at the bottom of the screen clone the sample code repository for the lab:

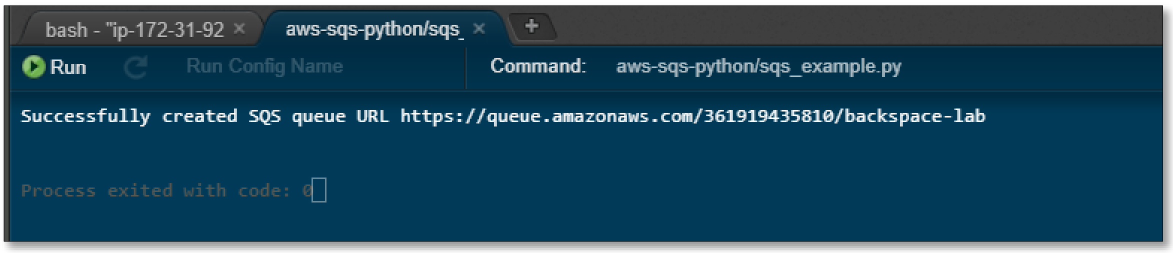
```
git clone https://github.com/backspace-academy/aws-sqs-python
```

```
# Load the AWS SDK for Python
import boto3

# Load the exceptions for error handling
from botocore.exceptions import ClientError, ParamValidationError

# Create SQS client and set region
sqs = boto3.client('sqs', region_name='us-east-1')

# Create an SQS queue
def create_sqs_queue(sqs_queue_name):
    try:
        data = sqs.create_queue(
            QueueName = sqs_queue_name,
            Attributes = {
                'ReceiveMessageWaitTimeSeconds': '20',
                'VisibilityTimeout': '60'
            }
        )
        return data['QueueUrl']
    # An error occurred
    except ParamValidationError as e:
        print("Parameter validation error: %s" % e)
    except ClientError as e:
        print("Client error: %s" % e)

# Main program
def main():
    sqs_queue_url = create_sqs_queue('backspace-lab')
    print('Successfully created SQS queue URL '+ sqs_queue_url )


if __name__ == '__main__':
    main()
```

Click *Run*

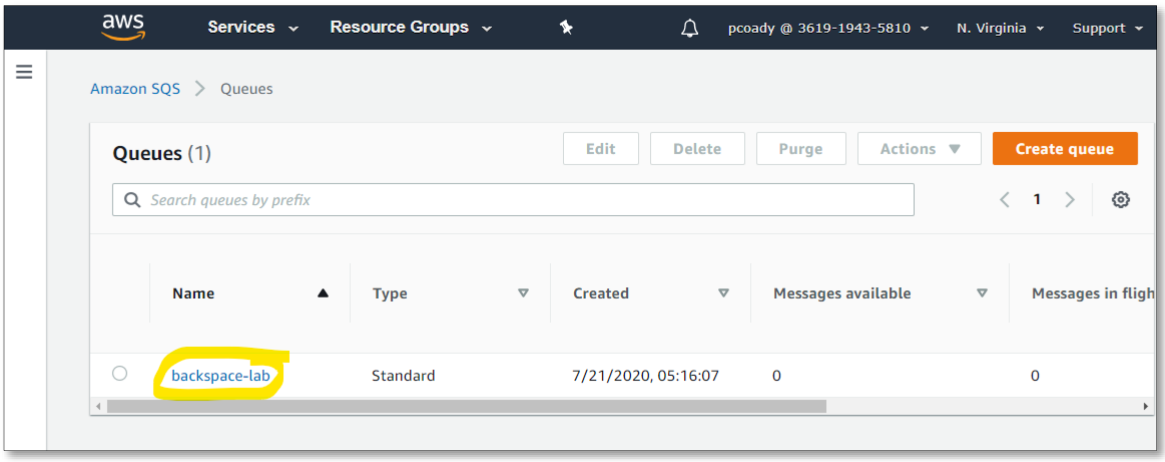


```
# Load the AWS SDK for Python
import boto3

# Load the exceptions for error handling
from botocore.exceptions import ClientError, ParamValidationError

# Create AWS service client and set region
sqs = boto3.client('sqs', region_name='us-east-1')

# Create an SQS queue
def create_sqs_queue(sqs_queue_name):
    try:
```



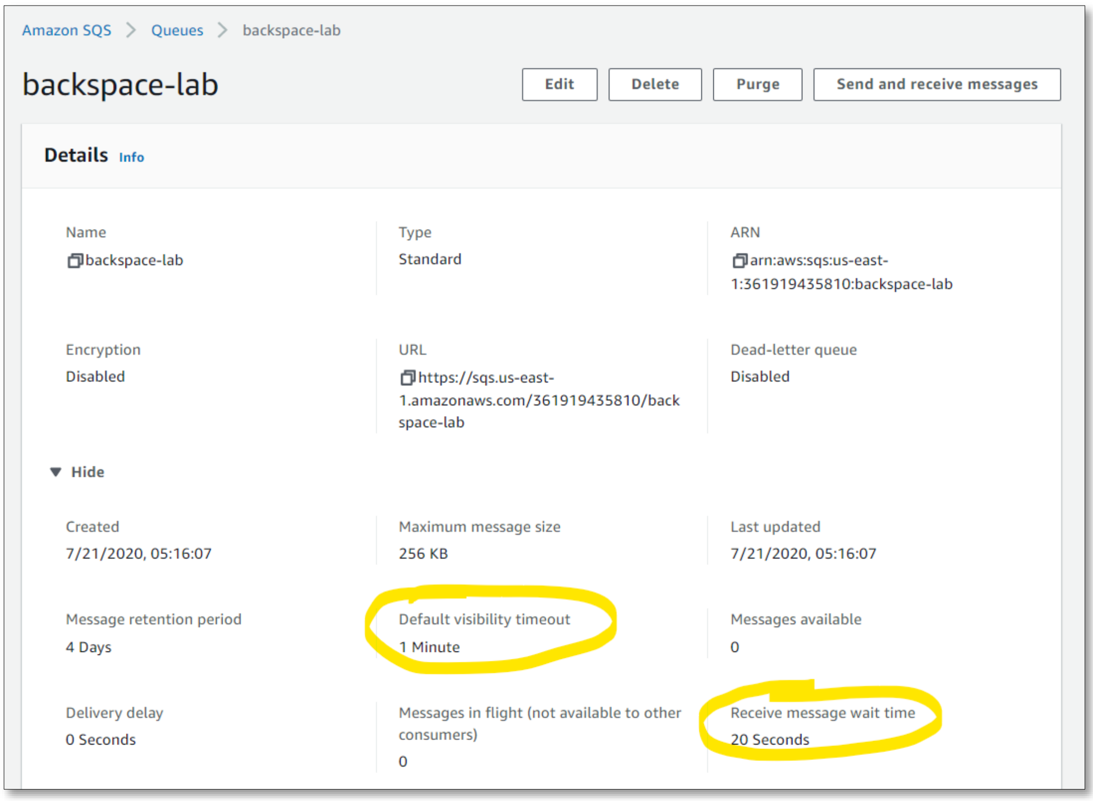
```
Successfully created SQS queue URL https://queue.amazonaws.com/361919435810/backspace-lab

Process exited with code: 0
```

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



Name	Type	Created	Messages available	Messages in fligh
backspace-lab	Standard	7/21/2020, 05:16:07	0	0

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


Amazon SQS > Queues > backspace-lab
Edit
Delete
Purge
Send and receive messages
Info
Hide

backspace-lab

Details

Name backspace-lab	Type Standard	ARN arn:aws:sqs:us-east-1:361919435810:backspace-lab
Encryption Disabled	URL https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab	Dead-letter queue Disabled
Created 7/21/2020, 05:16:07	Maximum message size 256 KB	Last updated 7/21/2020, 05:16:07
Message retention period 4 Days	Default visibility timeout 1 Minute	Messages available 0
Delivery delay 0 Seconds	Messages in flight (not available to other consumers) 0	Receive message wait time 20 Seconds

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.

Sending SQS 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:

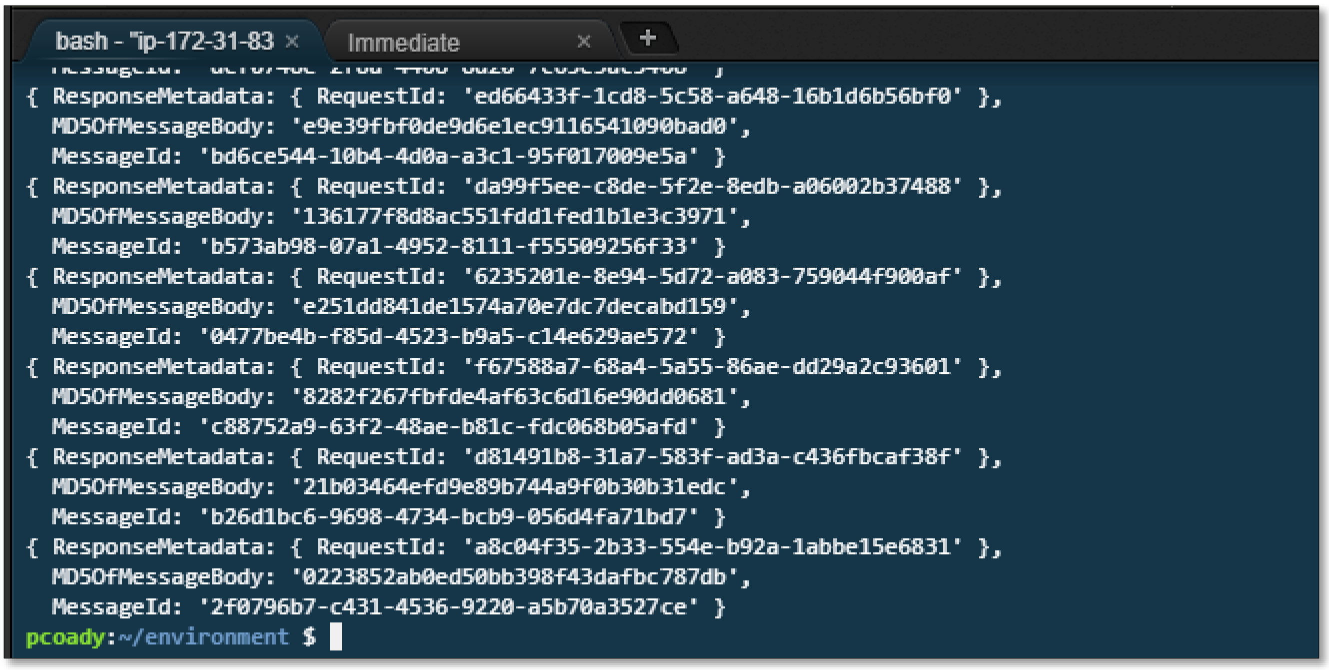
```
// Create 50 SQS messages
async function createMessages(queueUrl){
  var messages = [];
  for (var a=0; a<50; a++){
    messages[a] = 'This is the content for message '+ a + '.';
  }
  // Asynchronously deliver messages to SQS queue
  for (const message of messages){
    console.log('Sending message: '+ 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 Ctrl + S to save to the EC2 instance.

Now run index.js

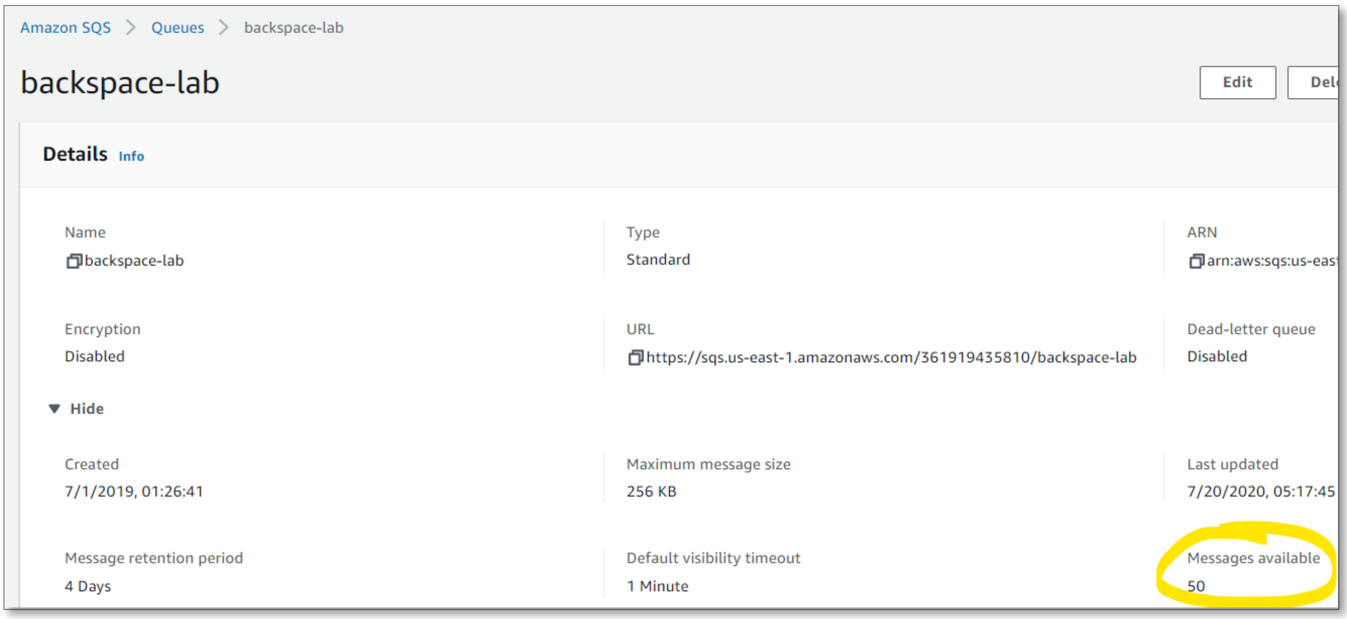
```
node index.js
```

It has now created and sent 50 messages.



```
{ 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.



backspace-lab

Details

Name: backspace-lab
Type: Standard
ARN: arn:aws:sqs:us-eas
Encryption: Disabled
URL: https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab
Dead-letter queue: Disabled
Created: 7/1/2019, 01:26:41
Maximum message size: 256 KB
Last updated: 7/20/2020, 05:17:45
Message retention period: 4 Days
Default visibility timeout: 1 Minute
Messages available: 50

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:

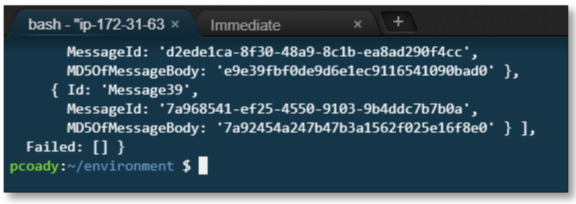
```
// Create 50 SQS messages
async function createMessages(queueUrl){
  var messages = [];
  for (var a=0; a<5; a++){
    messages[a] = [];
    for (var b=0; b<10; b++){
    messages[a][b] = 'This is the content for message '+ (a*10+b) + '.';
    }
  }

  // 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 Ctrl + S to save to the EC2 instance.

Now run index.js

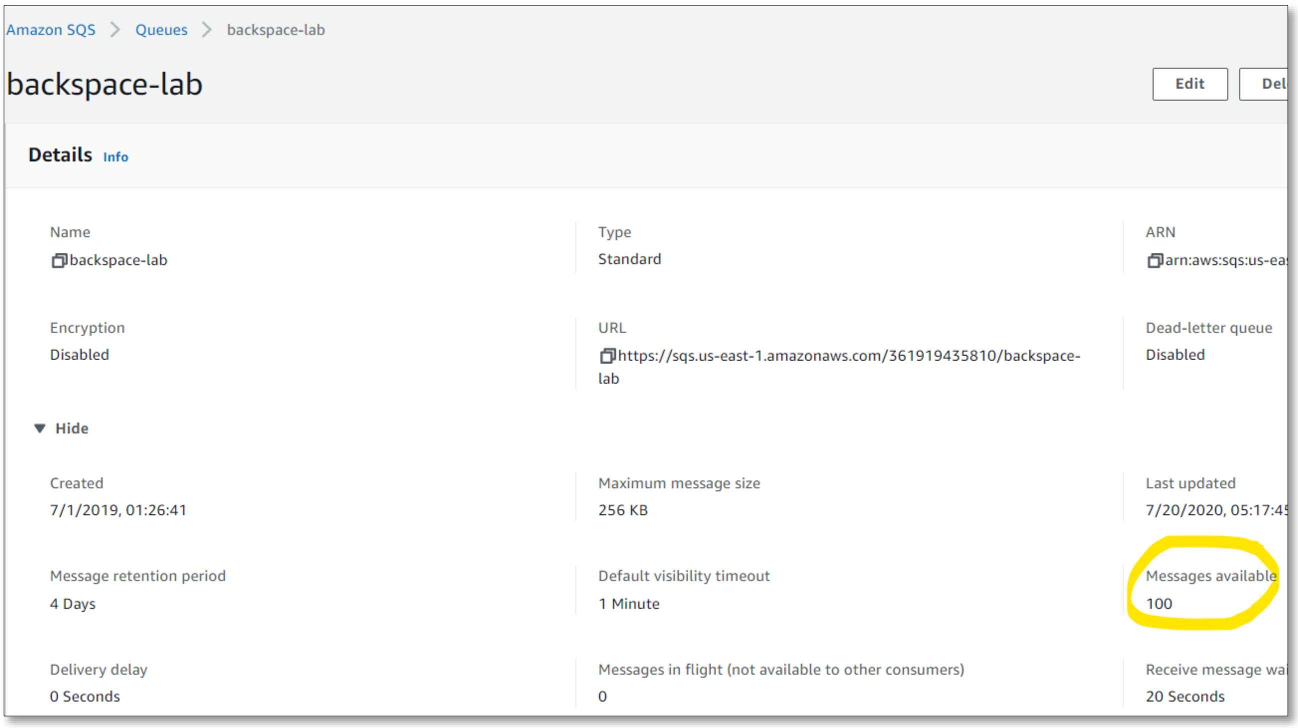
It has now created 50 messages but this time using only 5 calls to SQS instead of 50.

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



```
bash - "ip-172-31-63 ×    Immediate    ×    +
      MessageId: 'd2ede1ca-8f30-48a9-8c1b-ea8ad290f4cc',
      MD5OfMessageBody: 'e9e39fbf0de9d6e1ec9116541090bad0' },
    { Id: 'Message39',
      MessageId: '7a968541-ef25-4550-9103-9b4ddc7b7b0a',
      MD5OfMessageBody: '7a92454a247b47b3a1562f025e16f8e0' } ],
  Failed: [] }
pcoady:~/environment $ 
```

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



Amazon SQS > Queues > backspace-lab

backspace-lab

Edit Del

Details Info

Name backspace-lab	Type Standard	ARN arn:aws:sqs:us-ea
Encryption Disabled	URL https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab	Dead-letter queue Disabled

▼ Hide

Created 7/1/2019, 01:26:41	Maximum message size 256 KB	Last updated 7/20/2020, 05:17:4
Message retention period 4 Days	Default visibility timeout 1 Minute	Messages available 100
Delivery delay 0 Seconds	Messages in flight (not available to other consumers) 0	Receive message wa 20 Seconds

Creating SQS Messages using the AWS Python SDK

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

Sending SQS Messages with *send_message*

Now add a function before the main program to create a list of messages and send them using *send_message*

```
# Send 50 SQS messages
def create_messages(queue_url):
    # Create 50 messages
    TempMessages = []
    for a in range(50):
        tempStr = 'This is the content for message ' + str(a)
        TempMessages.append(tempStr)
    # Deliver messages to SQS queue_url
    for message in TempMessages:
        try:
            data = sqs.send_message(
                QueueUrl = queue_url,
                MessageBody = message
                )
            print(data['MessageId'])
        # An error occurred
        except ParamValidationError as e:
            print("Parameter validation error: %s" % e)
        except ClientError as e:
            print("Client error: %s" % e)
```

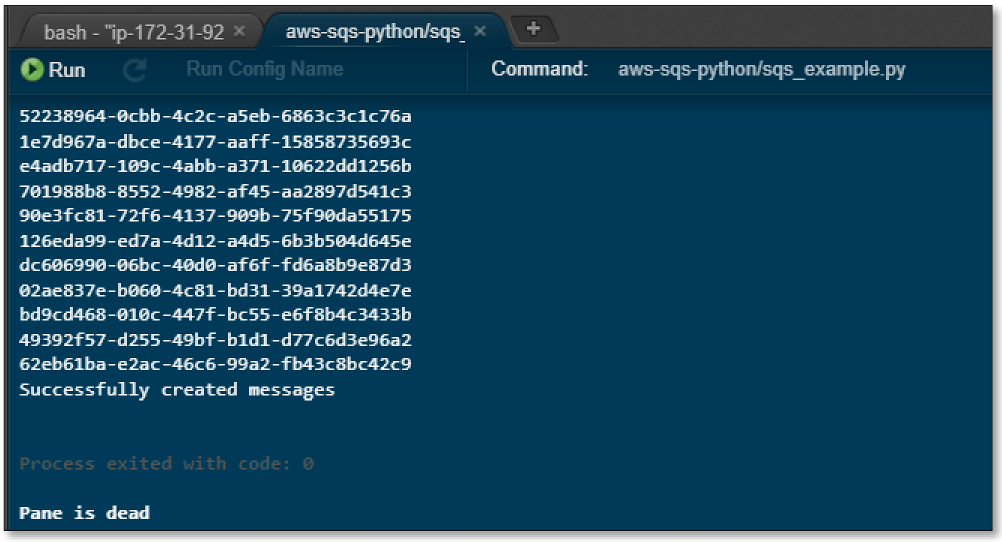
Now add a call to the function in the main program

```
# Main program
def main():
    sqs_queue_url = create_sqs_queue('backspace-lab')
    print('Successfully created SQS queue URL '+ sqs_queue_url )
    create_messages(sqs_queue_url)
    print('Successfully created messages')
```

Click Ctrl + S to save to the EC2 instance.

Click *Run*

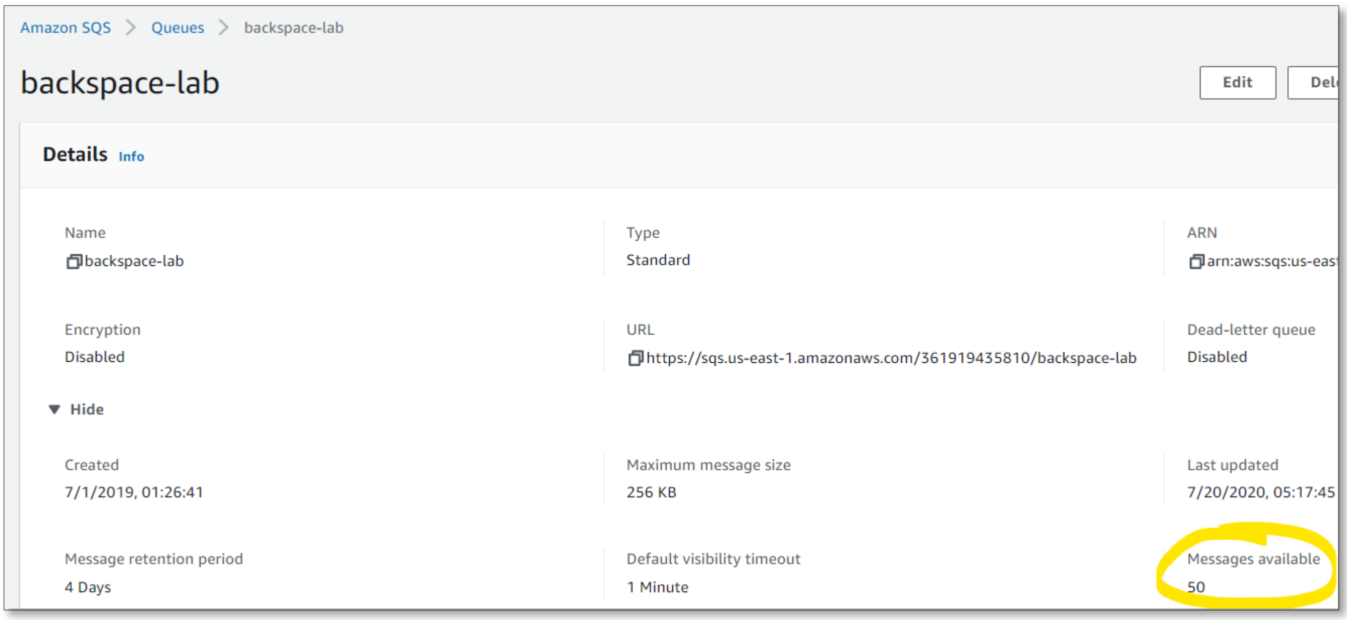
It has now sent 50 messages to the queue.



Command: aws-sqs-python/sqs_example.py

```
52238964-0cbb-4c2c-a5eb-6863c3c1c76a
1e7d967a-dbce-4177-aaff-15858735693c
e4adb717-109c-4abb-a371-10622dd1256b
701988b8-8552-4982-af45-aa2897d541c3
90e3fc81-72f6-4137-909b-75f90da55175
126eda99-ed7a-4d12-a4d5-6b3b504d645e
dc606990-06bc-40d0-af6f-fd6a8b9e87d3
02ae837e-b060-4c81-bd31-39a1742d4e7e
bd9cd468-010c-447f-bc55-e6f8b4c3433b
49392f57-d255-49bf-b1d1-d77c6d3e96a2
62eb61ba-e2ac-46c6-99a2-fb43c8bc42c9
Successfully created messages


Process exited with code: 0

Pane is dead
```

Go to the SQS console to see the 50 messages have been added.


Amazon SQS > Queues > backspace-lab

backspace-lab

Details Info

Name backspace-lab	Type Standard	ARN arn:aws:sqs:us-eas
Encryption Disabled	URL https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab	Dead-letter queue Disabled
Created 7/1/2019, 01:26:41	Maximum message size 256 KB	Last updated 7/20/2020, 05:17:45
Message retention period 4 Days	Default visibility timeout 1 Minute	Messages available 50

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.

First we'll change our array to 2 dimensional to accommodate 5 batches of ten messages.

Then we'll deliver the messages in batches of 10 using *sendMessageBatch*

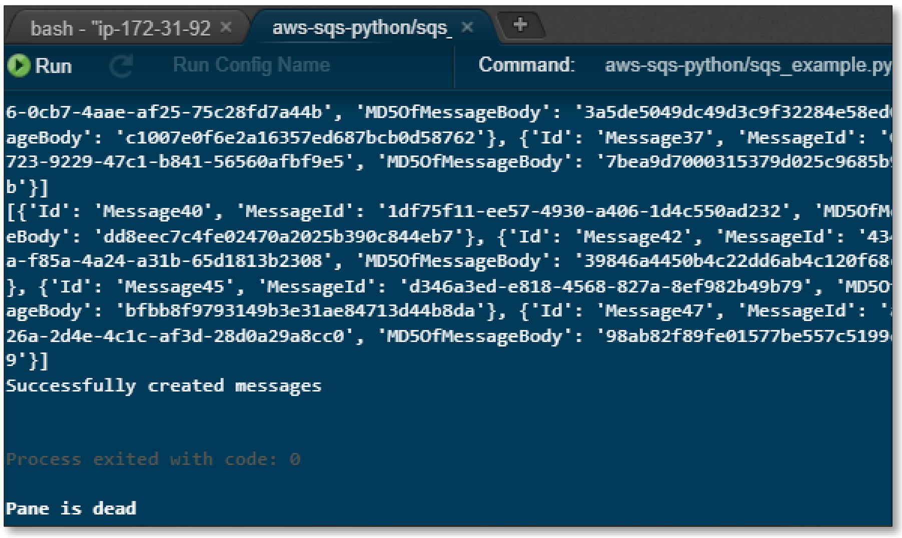
Change *create_Messages* to:

```
# Send 50 SQS messages
def create_messages(queue_url):
    # Create 50 messages in batches of 10
    TempMessages = []
    for a in range(5):
        TempEntries = []
        for b in range(10):
            tempStr1 = 'This is the content for message ' + str((a*10+b))
            tempStr2 = 'Message' +  str((a*10+b))
            tempEntry = {
                'MessageBody': tempStr1,
                'Id': tempStr2
            }
            TempEntries.append(tempEntry)
        TempMessages.append(TempEntries)
    # Deliver messages to SQS queue_url
    for batch in TempMessages:
        try:
            data = sqs.send_message_batch(
                QueueUrl = queue_url,
                Entries = batch
                )
            print(data['Successful'])
        # An error occurred
        except ParamValidationError as e:
            print("Parameter validation error: %s" % e)
        except ClientError as e:
            print("Client error: %s" % e)
```

Click Ctrl + S to save to the EC2 instance.

Click *Run*

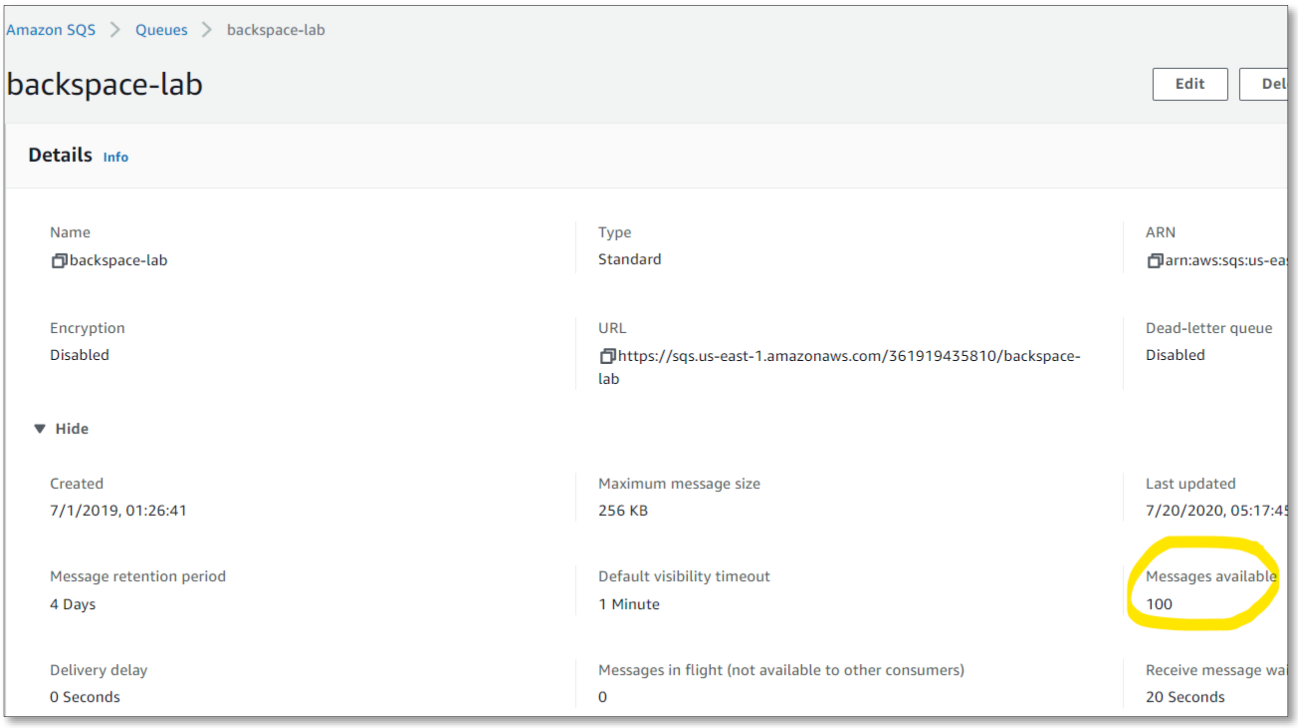
It has now sent 50 messages but this time using only 5 calls to SQS instead of 50.



Command: aws-sqs-python/sqs_example.py

```
6-0cb7-4aae-af25-75c28fd7a44b', 'MD5OfMessageBody': '3a5de5049dc49d3c9f32284e58ed
ageBody': 'c1007e0f6e2a16357ed687bcb0d58762'}, {'Id': 'Message37', 'MessageId': '
723-9229-47c1-b841-56560afbf9e5', 'MD5OfMessageBody': '7bea9d7000315379d025c9685b
b'}]
[{'Id': 'Message40', 'MessageId': '1df75f11-ee57-4930-a406-1d4c550ad232', 'MD5OfM
eBody': 'dd8eec7c4fe02470a2025b390c844eb7'}, {'Id': 'Message42', 'MessageId': '43
a-f85a-4a24-a31b-65d1813b2308', 'MD5OfMessageBody': '39846a4450b4c22dd6ab4c120f68
}, {'Id': 'Message45', 'MessageId': 'd346a3ed-e818-4568-827a-8ef982b49b79', 'MD5O
ageBody': 'bfbb8f9793149b3e31ae84713d44b8da'}, {'Id': 'Message47', 'MessageId': '
26a-2d4e-4c1c-af3d-28d0a29a8cc0', 'MD5OfMessageBody': '98ab82f89fe01577be557c5199
9'}]
Successfully created messages


Process exited with code: 0

Pane is dead
```

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



backspace-lab

Details

Name backspace-lab	Type Standard	ARN arn:aws:sqs:us-ea
Encryption Disabled	URL https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab	Dead-letter queue Disabled
Created 7/1/2019, 01:26:41	Maximum message size 256 KB	Last updated 7/20/2020, 05:17:4
Message retention period 4 Days	Default visibility timeout 1 Minute	Messages available 100
Delivery delay 0 Seconds	Messages in flight (not available to other consumers) 0	Receive message wai 20 Seconds

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.

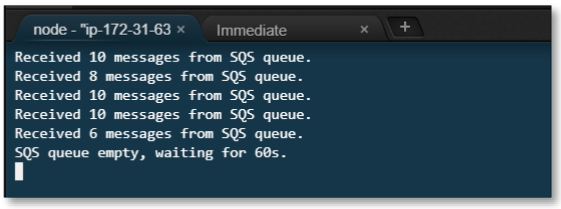
You can define *WaitTimeSeconds* and *VisibilityTimeout* in the call as shown here or the SQS service will use the defaults you set when creating the queue.

```
// Receive messages from queue
function receiveMessages(){
  var params = {
    QueueUrl: queueUrl, /* required */
    MaxNumberOfMessages: 10,
    VisibilityTimeout: 60, // Make sure the message is not visible in the queue
    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.');
      }
    }
  });
}
```

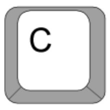
Click Ctrl + S to save to the EC2 instance.

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 8 messages from SQS queue.
Received 10 messages from SQS queue.
Received 10 messages from SQS queue.
Received 6 messages from SQS queue.
SQS queue empty, waiting for 60s.
```

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.

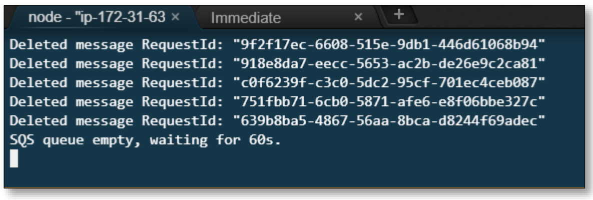
```
// Process and delete messages from queue
async function processMessages(messagesSQS){
    for (const item of messagesSQS){
        await console.log('Processing message: '+ item.Body); // Do something with the
message
        var params = {
            QueueUrl: queueUrl, /* required */
            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 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.



```
Deleted message RequestId: "9f2f17ec-6608-515e-9db1-446d61068b94"
Deleted message RequestId: "918e8da7-eecc-5653-ac2b-de26e9c2ca81"
Deleted message RequestId: "c0f6239f-c3c0-5dc2-95cf-701ec4ceb087"
Deleted message RequestId: "751fbb71-6cb0-5871-afe6-e8f06bbe327c"
Deleted message RequestId: "639b8ba5-4867-56aa-8bca-d8244f69adec"
SQS queue empty, waiting for 60s.
```

Press Ctrl + C to stop the application.

Processing SQS Messages using the Python SDK

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

First let's create a polling function with 1 second interval between polls.

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.

First we need to import the time module

```
# Load the AWS SDK for Python
import boto3
import time
```

Now create a function to continuously do the following:

- Send a *receive_message* call to the SQS queue URL
- Check if the response includes messages
- Print messages or wait 60s before making another call if empty response

```
# Receive SQS messages
def receive_messages(queue_url):
    print('Reading messages')
    while True:
        try:
            data = sqs.receive_message(
                QueueUrl = queue_url,
                MaxNumberOfMessages = 10,
                VisibilityTimeout = 60,
                WaitTimeSeconds = 20
                )
        # An error occurred
        except ParamValidationError as e:
            print("Parameter validation error: %s" % e)
        except ClientError as e:
            print("Client error: %s" % e)
        # Check if empty receive
        try:
            data['Messages']
        except KeyError:
            data = None
        if data is None:
            print('Queue empty waiting 60s')
            # Wait for 60 seconds
            time.sleep(60)
        else:
            print(data['Messages'])
            # Wait for 1 second
            time.sleep(1)
```

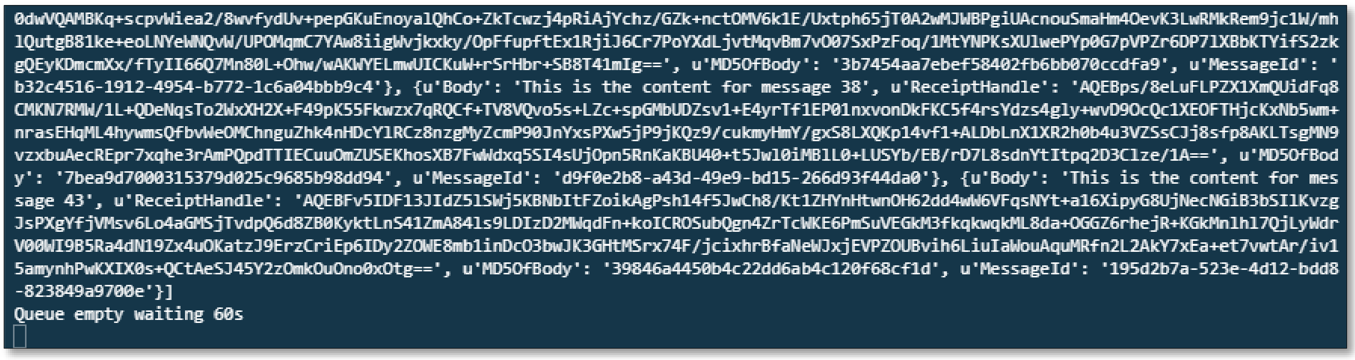
Finally add a call to the function in the main program function

```python
# Main program
def main():
    sqs_queue_url = create_sqs_queue('backspace-lab')
    print('Successfully created SQS queue URL '+ sqs_queue_url )
    create_messages(sqs_queue_url)
    print('Successfully created messages')
    receive_messages(sqs_queue_url)
```

Click Ctrl + S to save to the EC2 instance.

Click *Run*

You can see it is receiving messages then waiting 60s when an empty response is received. When the 60s has expired, the visibility timeout period will make the messages visible in the queue again.



```
0dwVQAMBKq+scpvWiea2/8wvfydUv+pepGKuEnoyalQhCo+ZkTcwzj4pRiAjYchz/GZk+nctOMV6k1E/Uxtph65jT0A2wMJWBPgiUAcnouSmaHm4OevK3LwRMkRem9jc1W/mh
lQutgB81ke+eoLNYeWNQvW/UPOMqmC7YAw8iigWvjkxky/OpFfupftEx1RjiJ6Cr7PoYXdLjvtMqvBm7vO07SxPzFoq/1MtYNPKsXUlwePYp0G7pVPZr6DP71XBbKTYifS2zk
gQEyKDmcmXx/fTyII66Q7Mn80L+Ohw/wAKWYELmwUICKuW+rSrHbr+SB8T41mIg==', u'MD5OfBody': '3b7454aa7ebef58402fb6bb070ccdfa9', u'MessageId': '
b32c4516-1912-4954-b772-1c6a04bbb9c4'}, {u'Body': 'This is the content for message 38', u'ReceiptHandle': 'AQEBps/8eLuFLPZX1XmQUidFq8
CMKN7RMW/1L+QDeNqsTo2WxXH2X+F49pK55Fkwzx7qRQCf+TV8VQvo5s+LZc+spGMbUDZsv1+E4yrTf1EP01nxvonDkFKC5f4rsYdzs4gly+wvD9OcQc1XEOFTHjcKxNb5wm+
nrasEHqML4hywmsQfbvWeOMChnguZhk4nHDcY1RCz8nzgMyZcmP90JnYxsPXw5jP9jKQz9/cukmyHmY/gxS8LXQKp14vf1+ALDbLnX1XR2h0b4u3VZSsCJj8sfp8AKLTsgMN9
vzxbuAecREpr7xqhe3rAmPQpdTTIECuuOmZUSEKhosXB7FwWdxq5SI4sUjOpn5RnKaKBU40+t5Jwl0iMBlL0+LUSYb/EB/rD7L8sdnYtItpq2D3Clze/1A==', u'MD5OfBod
y': '7bea9d7000315379d025c9685b98dd94', u'MessageId': 'd9f0e2b8-a43d-49e9-bd15-266d93f44da0'}, {u'Body': 'This is the content for mes
sage 43', u'ReceiptHandle': 'AQEBFv5IDF13JIdZ51SWj5KBNbItFZoikAgPsh14f5JwCh8/Kt1ZHYnHtwnOH62dd4wW6VFqsNYt+a16XipyG8UjNecNGiB3bSIlKvzg
JsPXgYfjVMsv6Lo4aGMSjTvdpQ6d8ZB0KyktLnS41ZmA841s9LDIzD2MWqdFn+koICROSubQgn4ZrTcWKE6PmSuVEGkM3fkqkwqkML8da+OGGZ6rhejR+KGkMnlhl7QjLyWdr
V00WI9B5Ra4dN19Zx4uOKatzJ9ErzCriEp6IDy2ZOWE8mb1inDcO3bwJK3GHtMSrx74F/jcixhrBfaNeWJxjEVPZOUBvih6LiuIaWouAquMRfn2L2AkY7xEa+et7vwtAr/iv1
5amynhPwKXIX0s+QCtAeSJ45Y2zOmkOuOno0xOtg==', u'MD5OfBody': '39846a4450b4c22dd6ab4c120f68cf1d', u'MessageId': '195d2b7a-523e-4d12-bdd8
-823849a9700e'}]
Queue empty waiting 60s
```

Press Ctrl + C to stop the application.

Lets introduce code to delete the message after we have received it.

Add a call to *delete_message* and pass the queue URL and Message Receipt Handle after the message has been printed to the screen.

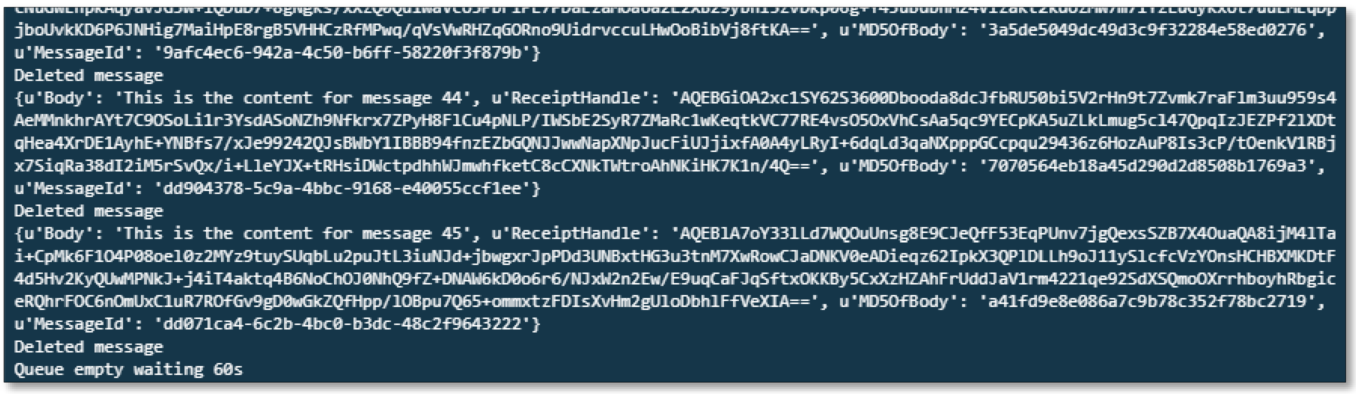
```
# Receive SQS messages
def receive_messages(queue_url):
    print('Reading messages')
    while True:
        try:
            data = sqs.receive_message(
                QueueUrl = queue_url,
                MaxNumberOfMessages = 10,
                VisibilityTimeout = 60,
                WaitTimeSeconds = 20
                )
        # An error occurred
        except ParamValidationError as e:
            print("Parameter validation error: %s" % e)
        except ClientError as e:
            print("Client error: %s" % e)
        # Check if empty receive
        try:
            data['Messages']
        except KeyError:
            data = None
        if data is None:
            print('Queue empty waiting 60s')
            # Wait for 60 seconds
            time.sleep(60)
        else:
            for message in data['Messages']:
                print(message)
                sqs.delete_message(
                    QueueUrl = queue_url,
                    ReceiptHandle = message['ReceiptHandle']
                )
                print('Deleted message')
            # Wait for 1 second
            time.sleep(1)
```

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.



```
jboUvkKD6P6JNHig7MaiHpE8rgB5VHHCzRfMPwq/qVsVwRHZqGORno9UidrvccuLHwOoBibVj8ftKA==', u'MD5OfBody': '3a5de5049dc49d3c9f32284e58ed0276',
u'MessageId': '9afc4ec6-942a-4c50-b6ff-58220f3f879b'}
Deleted message
{u'Body': 'This is the content for message 44', u'ReceiptHandle': 'AQEBGiOA2xc1SY62S3600Dbooda8dcJfbRU50bi5V2rHn9t7Zvmk7raFlm3uu959s4
AeMMnkhrAYt7C9OSoLi1r3YsdASoNZh9Nfkrx7ZPyH8FlCu4pNLP/IWSbE2SyR7ZMaRc1wKeqtkVC77RE4vsO5OxVhCsAa5qc9YECpKA5uZLkLmug5cl47QpqIzJEZPf21XDt
qHea4XrDE1AyhE+YNBfs7/xJe99242QJsBWbY1IBBB94fnzEZbGQNJJwwNapXNpJucFiUJjixfA0A4yLRyI+6dqLd3qaNXpppGCcpqu29436z6HozAuP8Is3cP/tOenkV1RBj
x7SiqRa38dI2iM5rSvQx/i+LleYJX+tRHsiDWctpdhhWJmwhfketC8cCXNkTWtroAhNKiHK7K1n/4Q==', u'MD5OfBody': '7070564eb18a45d290d2d8508b1769a3',
u'MessageId': 'dd904378-5c9a-4bbc-9168-e40055ccf1ee'}
Deleted message
{u'Body': 'This is the content for message 45', u'ReceiptHandle': 'AQEB1A7oY331Ld7WQOuUnsg8E9CJeQfF53EqPUnv7jgQexsSZB7X4OuaQA8ijM41Ta
i+CpMk6F1O4P08oe10z2MYz9tuySUqbLu2puJtL3iuNJd+jbwgxrJpPDd3UNBxtHG3u3tnM7XwRowCJaDNKV0eADieqz62IpkX3QP1DLLh9oJ11ySlcfcVzYOnsHCHBXMKDtF
4d5Hv2KyQUwMPNkJ+j4iT4aktq4B6NoChOJ0NhQ9fZ+DNAW6kD0o6r6/NJxW2n2Ew/E9uqCaFJqSftxOKKBy5CxXzHZAhFrUddJaV1rm4221qe92SdXSQmoOXrrhboyhRbgic
eRQhrFOC6nOmUxC1uR7ROfGv9gD0wGkZQfHpp/1OBpu7Q65+ommxtzFDIsXvHm2gUloDbhlFfVeXIA==', u'MD5OfBody': 'a41fd9e8e086a7c9b78c352f78bc2719',
u'MessageId': 'dd071ca4-6c2b-4bc0-b3dc-48c2f9643222'}
Deleted message
Queue empty waiting 60s
```

Press Ctrl + C to stop the application.

Subscribing an SQS Queue to an SNS Topic using the NodeJS SDK

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

Comment out the call to createMessages and save the file.

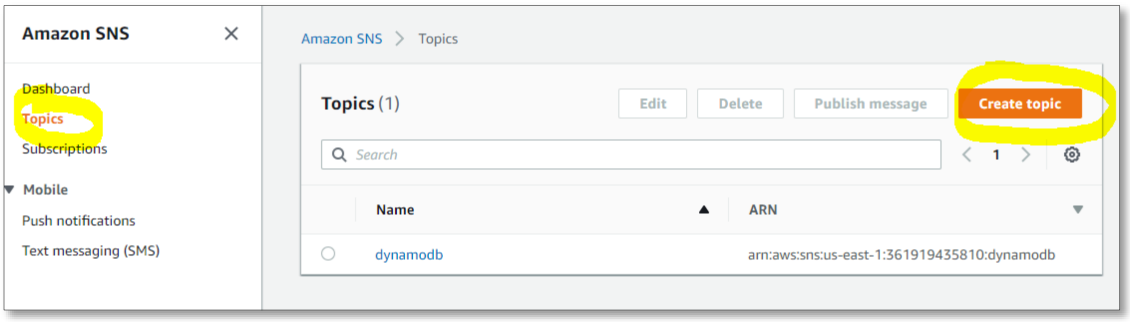
```
sqs.createQueue(params, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else {
      console.log('Successfully created SQS queue URL '+ data.QueueUrl);     //
successful response
      queueUrl = data.QueueUrl;
      waitingSQS = false;
      // createMessages(data.QueueUrl);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

Creating an SNS Topic

Go to the SNS console.

Go to *Topics*

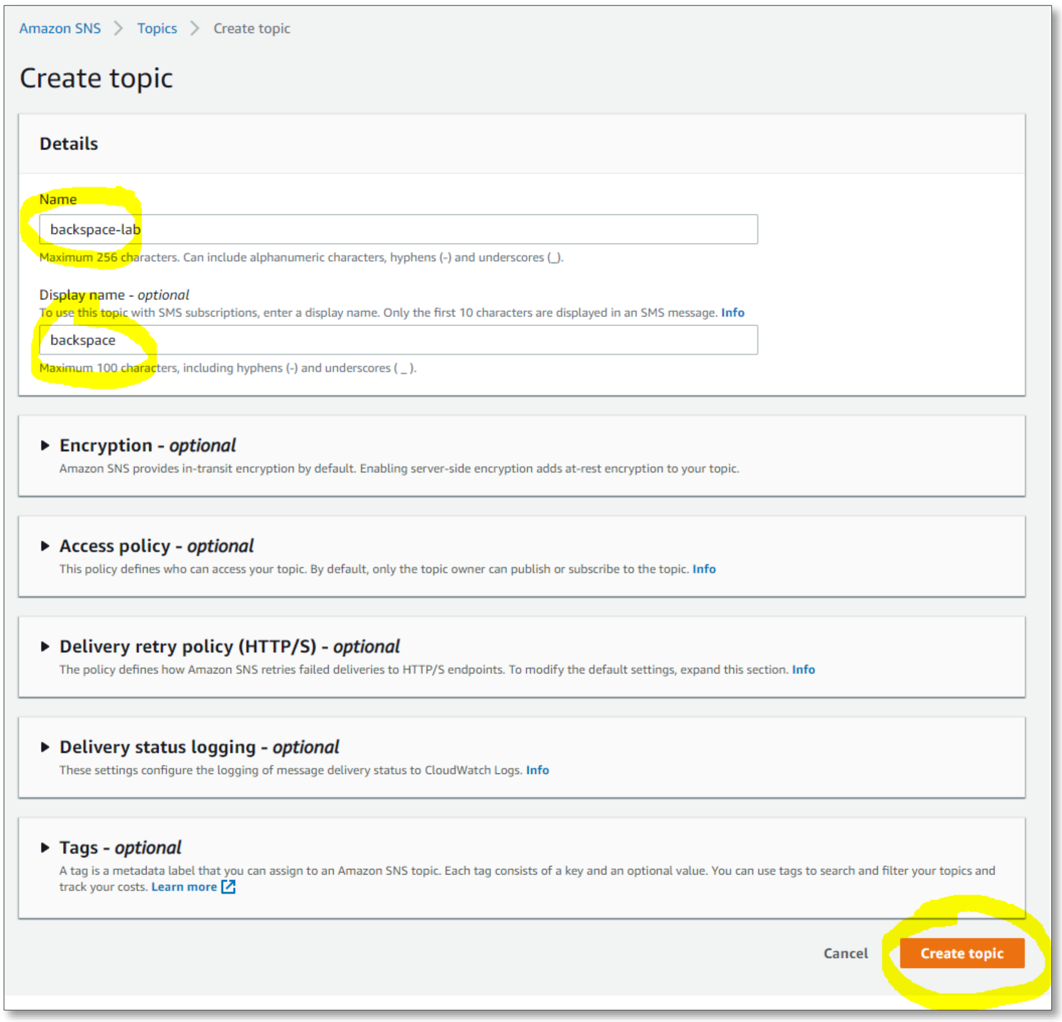
Click *Create Topic*



Name	ARN
dynamodb	arn:aws:sns:us-east-1:361919435810:dynamodb

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

Click *Create Topic*



Name
backspace-lab
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - *optional*
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message. Info
backspace
Maximum 100 characters, including hyphens (-) and underscores (_).

Encryption - *optional*
Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic.

Access policy - *optional*
This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic. Info

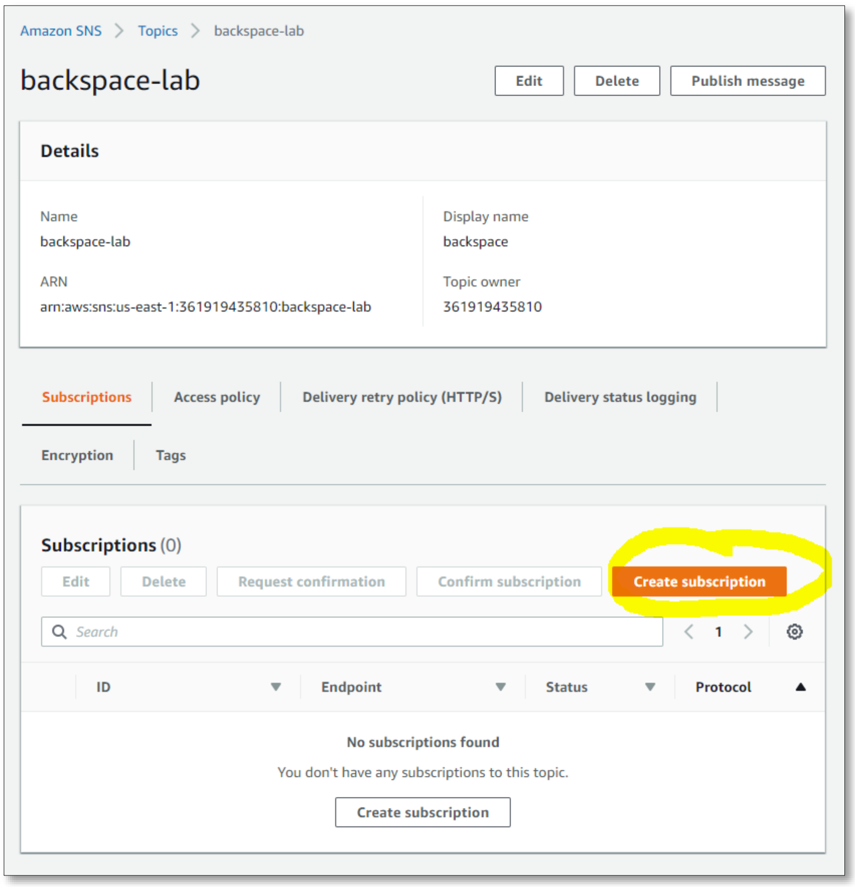
Delivery retry policy (HTTP/S) - *optional*
The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section. Info

Delivery status logging - *optional*
These settings configure the logging of message delivery status to CloudWatch Logs. Info

Tags - *optional*
A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more

Subscribing an SQS Queue to an SNS Topic

Click 'Create subscription"



Name
backspace-lab

ARN
arn:aws:sns:us-east-1:361919435810:backspace-lab

Display name
backspace

Topic owner
361919435810

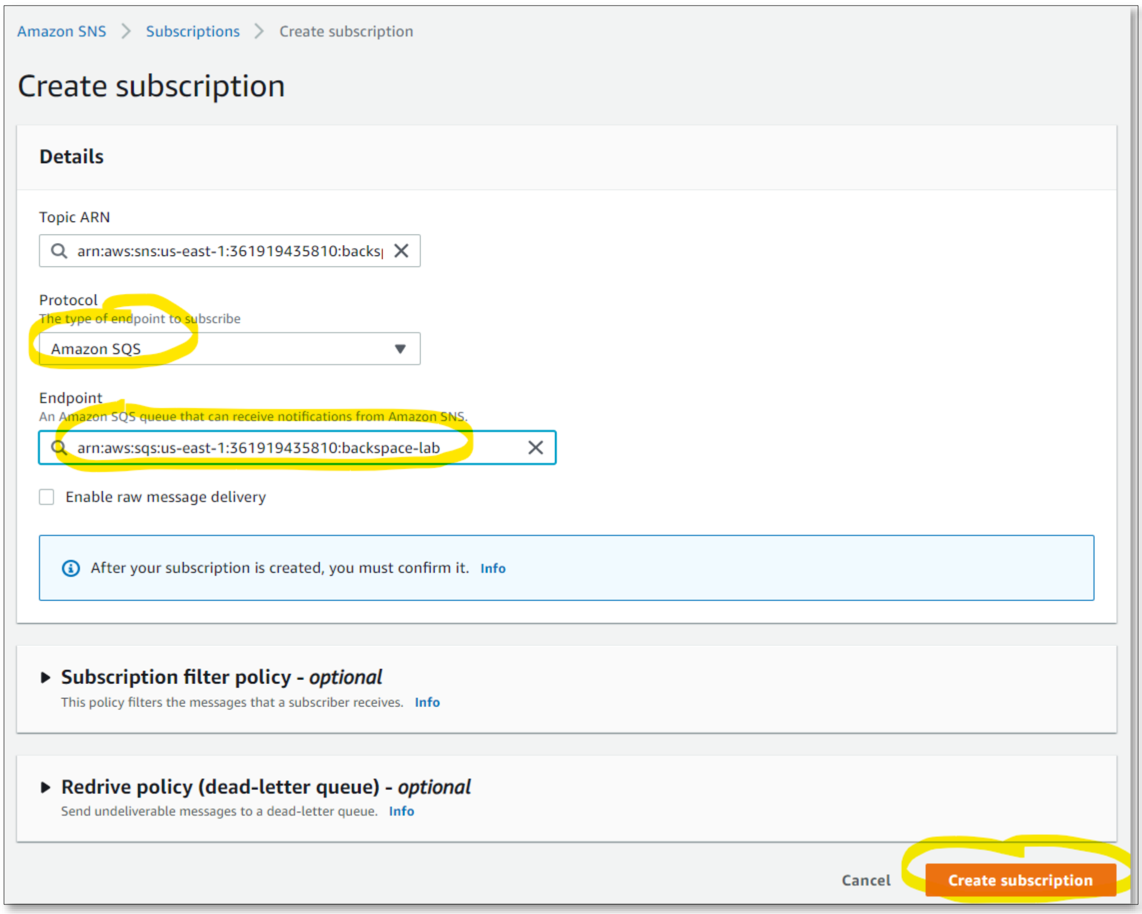
No subscriptions found

You don't have any subscriptions to this topic.

Select ‘Amazon SQS” for protocol

Select our SQS queue ARN.

Click *Create subscription*



Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN

arn:aws:sns:us-east-1:361919435810:backs|

Protocol
The type of endpoint to subscribe

Amazon SQS

Endpoint
An Amazon SQS queue that can receive notifications from Amazon SNS.

arn:aws:sqs:us-east-1:361919435810:backspace-lab

☐ Enable raw message delivery

After your subscription is created, you must confirm it. Info

Subscription filter policy - *optional*
This policy filters the messages that a subscriber receives. Info

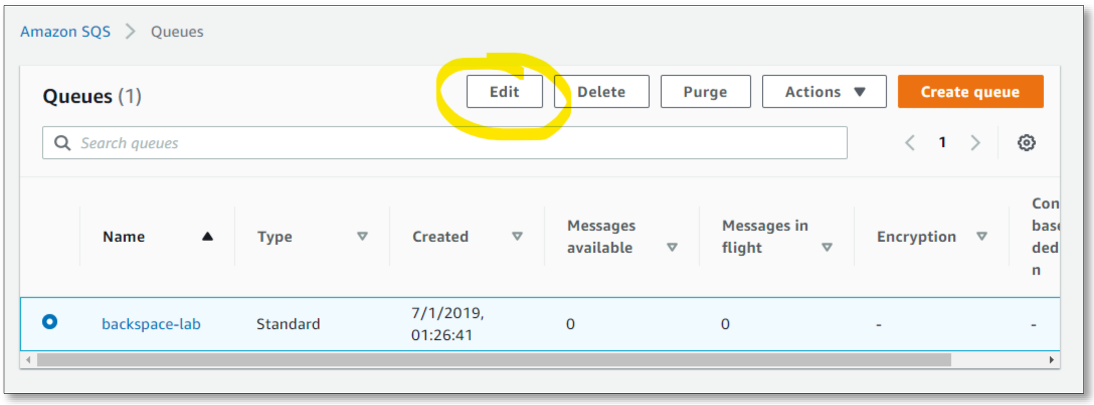
Redrive policy (dead-letter queue) - *optional*
Send undeliverable messages to a dead-letter queue. Info

Cancel | Create subscription

Granting SNS Permission to send messages to SQS

Now go back to the SQS console.

Select the queue and click *Edit*



Amazon SQS > Queues

Queues (1)

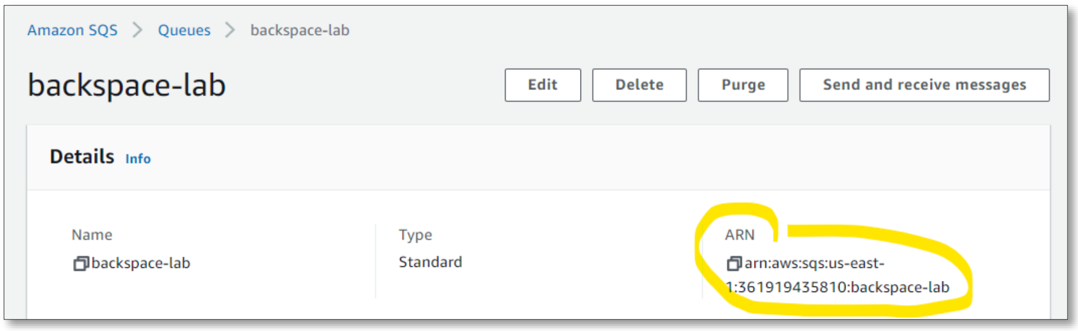
Edit | Delete | Purge | Actions | Create queue

Search queues

	Name	Type	Created	Messages available	Messages in flight	Encryption	Con base ded n
●	backspace-lab	Standard	7/1/2019, 01:26:41	0	0	-	-

Paste in the following Access policy.

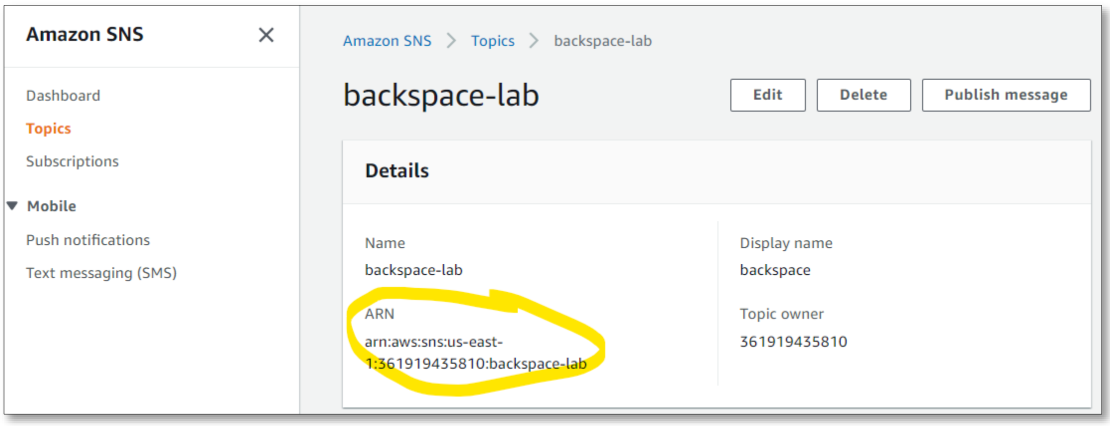
Make sure your copy and paste in your SQS queue ARN and SNS Topic (not the subscription) ARN.


Amazon SQS > Queues > backspace-lab
backspace-lab
Edit
Delete
Purge
Send and receive messages
Details Info

Name
backspace-lab

Type
Standard

ARN
arn:aws:sqs:us-east-1:361919435810:backspace-lab



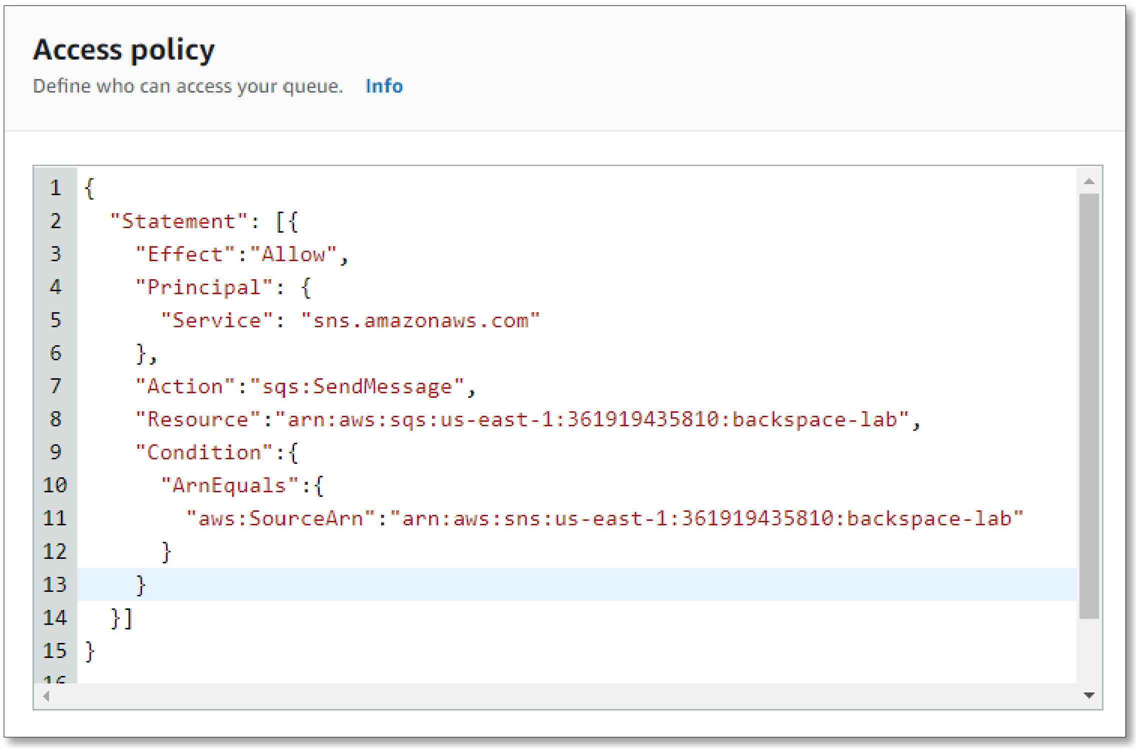
Name
backspace-lab

ARN
arn:aws:sns:us-east-1:361919435810:backspace-lab

Display name
backspace

Topic owner
361919435810

```
{
  "Statement": [{
    "Effect":"Allow",
    "Principal": {
      "Service":  "sns.amazonaws.com"
    },
    "Action":"sqs:SendMessage",
    "Resource":"YOUR_SQS_QUEUE_ARN",
    "Condition":{
      "ArnEquals":{
        "aws:SourceArn":"YOUR_SNS_TOPIC_ARN"
      }
    }
  }]
}
```

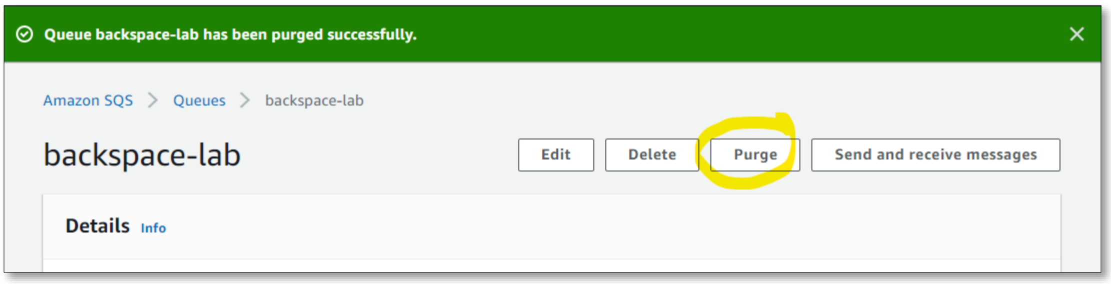

Access policy

Define who can access your queue. Info

```
{
  "Statement": [{
    "Effect":"Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action":"sqs:SendMessage",
    "Resource":"arn:aws:sqs:us-east-1:361919435810:backspace-lab",
    "Condition":{
      "ArnEquals":{
        "aws:SourceArn":"arn:aws:sns:us-east-1:361919435810:backspace-lab"
      }
    }
  }]
}
```

Click *Save*

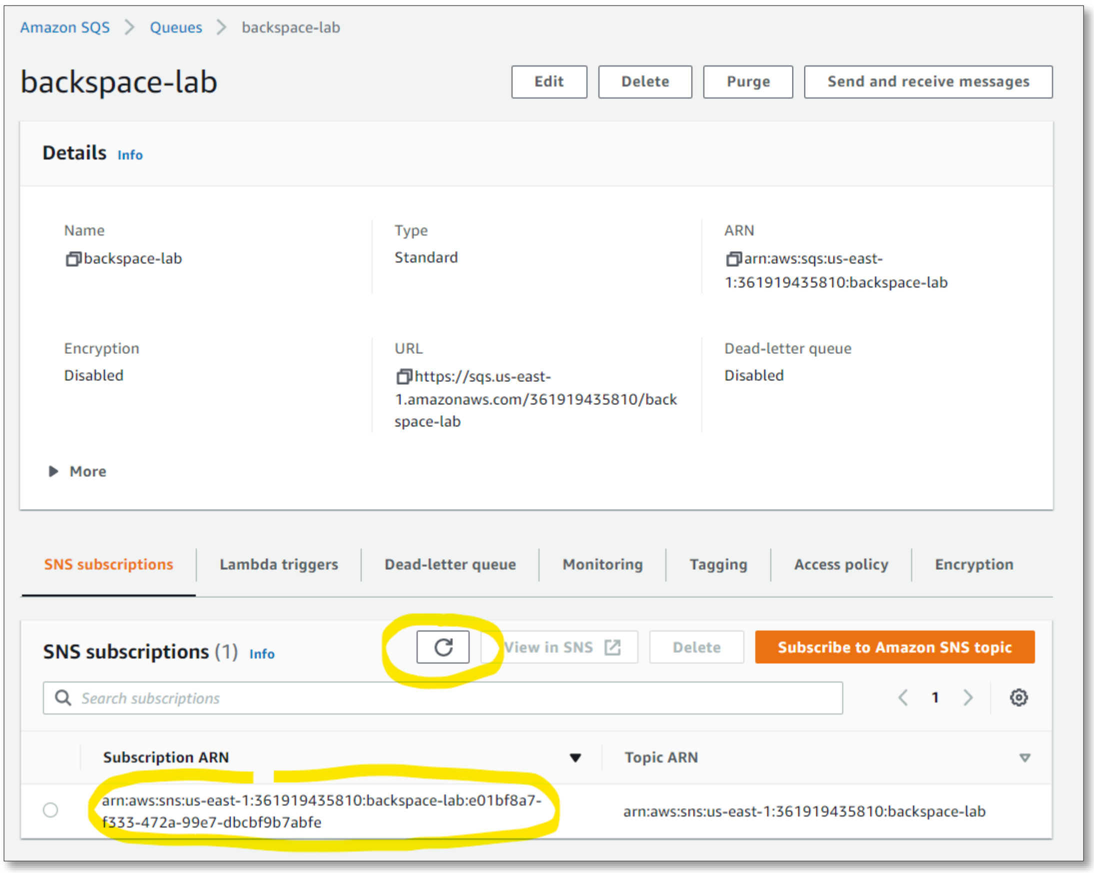
Purge any messages from the *backspace-lab* queue


Queue backspace-lab has been purged successfully.

Amazon SQS > Queues > backspace-lab

backspace-lab

Scroll down to see the SNS subscription has been added.



backspace-lab

Details Info

Name	Type	ARN
backspace-lab	Standard	arn:aws:sqs:us-east-1:361919435810:backspace-lab
Encryption	URL	Dead-letter queue
Disabled	https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab	Disabled

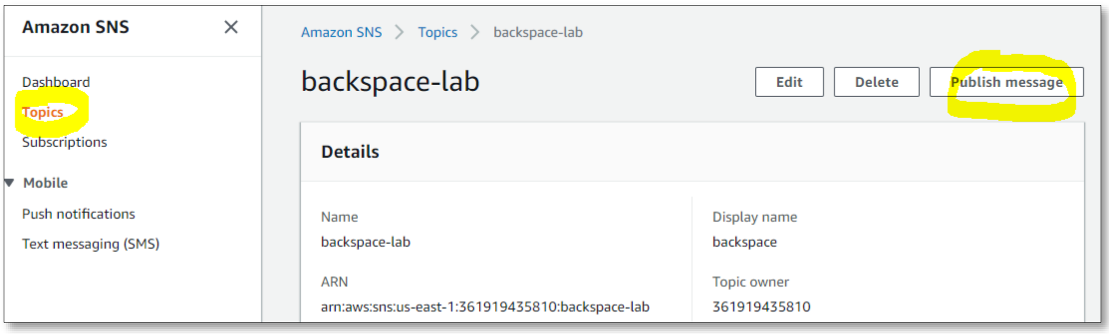
SNS subscriptions (1) Info

Subscription ARN	Topic ARN
arn:aws:sns:us-east-1:361919435810:backspace-lab:e01bf8a7-f333-472a-99e7-dbcbf9b7abfe	arn:aws:sns:us-east-1:361919435810:backspace-lab

Now go back to the SNS console.

Select *Topics*

Select the topic and click *Publish message*

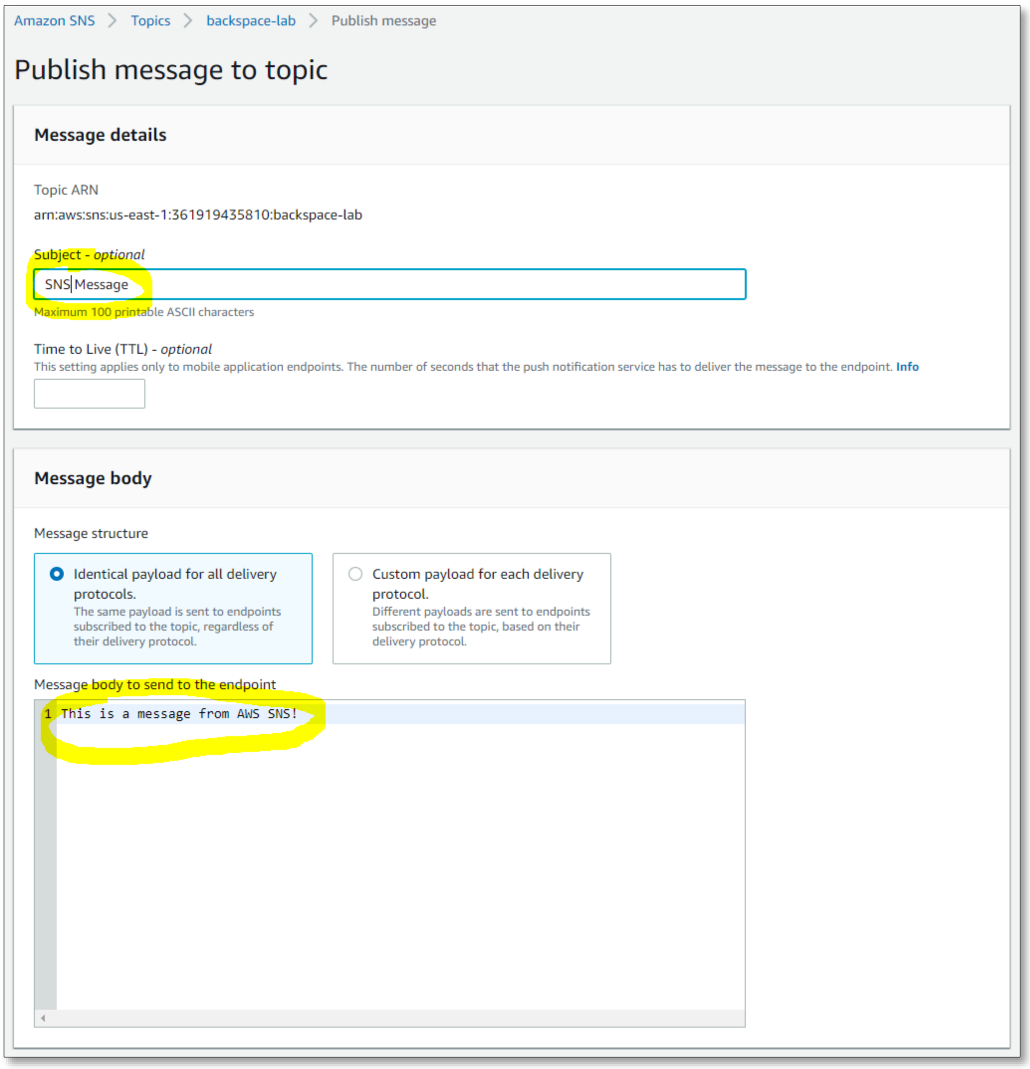


backspace-lab

Details

Name	Display name
backspace-lab	backspace
ARN	Topic owner
arn:aws:sns:us-east-1:361919435810:backspace-lab	361919435810

Create a subject and message.



Amazon SNS > Topics > backspace-lab > Publish message

Publish message to topic

Message details

Topic ARN
arn:aws:sns:us-east-1:361919435810:backspace-lab

Subject - *optional*
SNS Message
Maximum 100 printable ASCII characters

Time to Live (TTL) - *optional*
This setting applies only to mobile application endpoints. The number of seconds that the push notification service has to deliver the message to the endpoint. Info

Message body

Message structure

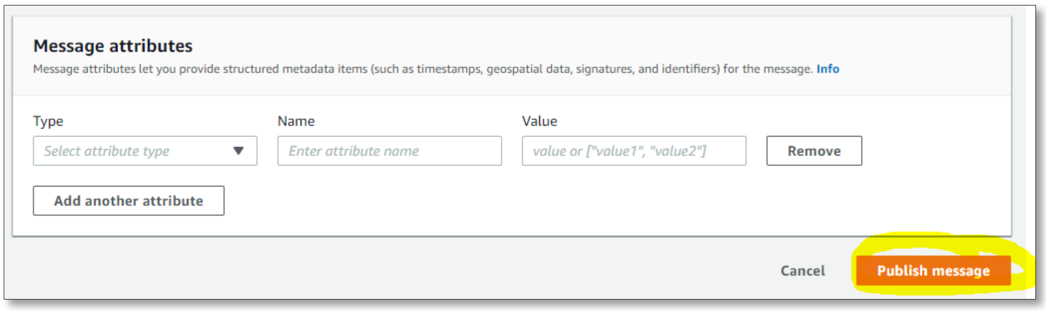
● Identical payload for all delivery protocols.
The same payload is sent to endpoints subscribed to the topic, regardless of their delivery protocol.

○ Custom payload for each delivery protocol.
Different payloads are sent to endpoints subscribed to the topic, based on their delivery protocol.

Message body to send to the endpoint

```
This is a message from AWS SNS!
```

Click *Publish message*

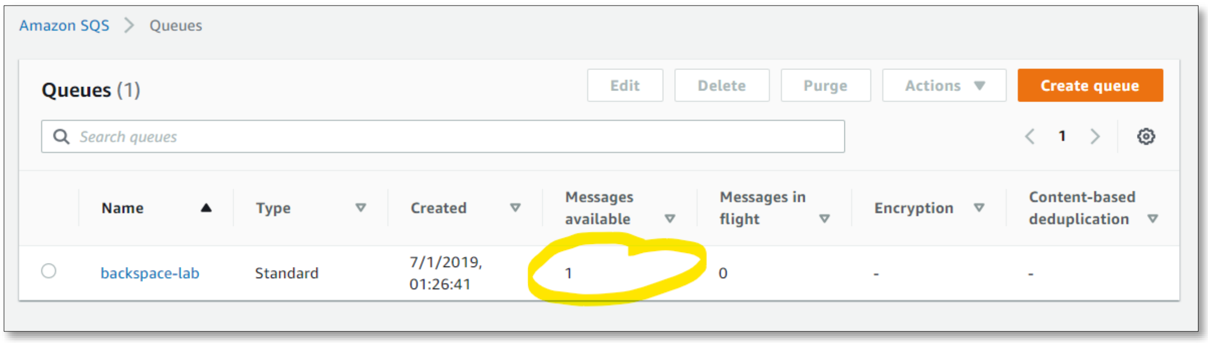


Message attributes
Message attributes let you provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) for the message. Info

Type	Name	Value	
Select attribute type	Enter attribute name	value or ["value1", "value2"]	Remove

Now go back to the SQS console

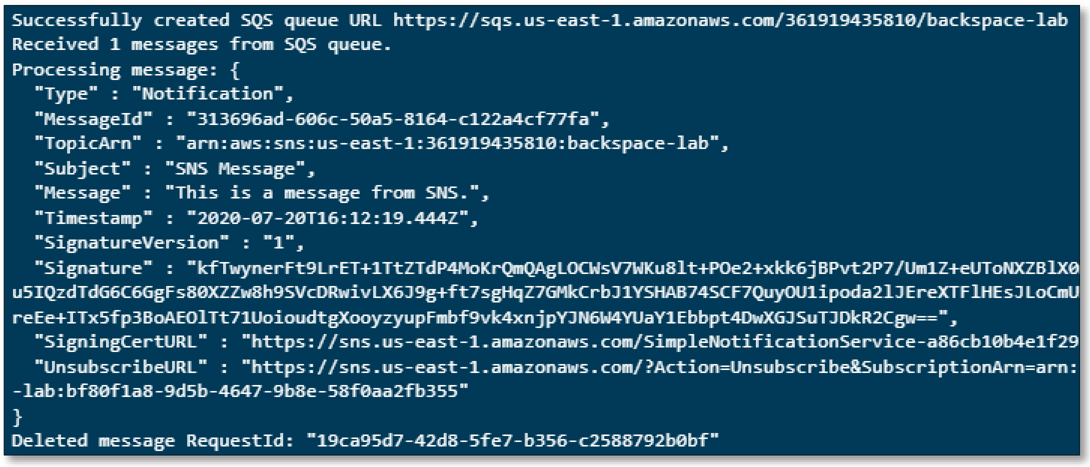
You will now see that the SNS message has been sent to the SQS queue



	Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication
○	backspace-lab	Standard	7/1/2019, 01:26:41	1	0	-	-

Now run your app again.

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



```
Successfully created SQS queue URL https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab
Received 1 messages from SQS queue.
Processing message: {
  "Type" : "Notification",
  "MessageId" : "313696ad-606c-50a5-8164-c122a4cf77fa",
  "TopicArn" : "arn:aws:sns:us-east-1:361919435810:backspace-lab",
  "Subject" : "SNS Message",
  "Message" : "This is a message from SNS.",
  "Timestamp" : "2020-07-20T16:12:19.444Z",
  "SignatureVersion" : "1",
  "Signature" : "kfTwynerFt9LrET+1TtZTdP4MoKrQmQAgLOCWsV7WKu8lt+POe2+xkk6jBPvt2P7/Um1Z+eUToNXZBlX0
u5IQzdTdG6C6GgFs80XZZw8h9SVcDRwivLX6J9g+ft7sgHqZ7GMkCrbJ1YSHAB74SCF7QuyOU1ipoda2lJEreXTFlHEsJLoCmU
reEe+ITx5fp3BoAEOlTt71UoioudtgXooyzyupFmbf9vk4xnjpYJN6W4YUaY1Ebbpt4DwXGJSuTJDkR2Cgw==",
  "SigningCertURL" : "https://sns.us-east-1.amazonaws.com/SimpleNotificationService-a86cb10b4e1f29
  "UnsubscribeURL" : "https://sns.us-east-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:
-lab:bf80f1a8-9d5b-4647-9b8e-58f0aa2fb355"
}
Deleted message RequestId: "19ca95d7-42d8-5fe7-b356-c2588792b0bf"
```

Now we will send an SNS message using the NodeJS SDK

Uncomment the createMessages call from createQueue

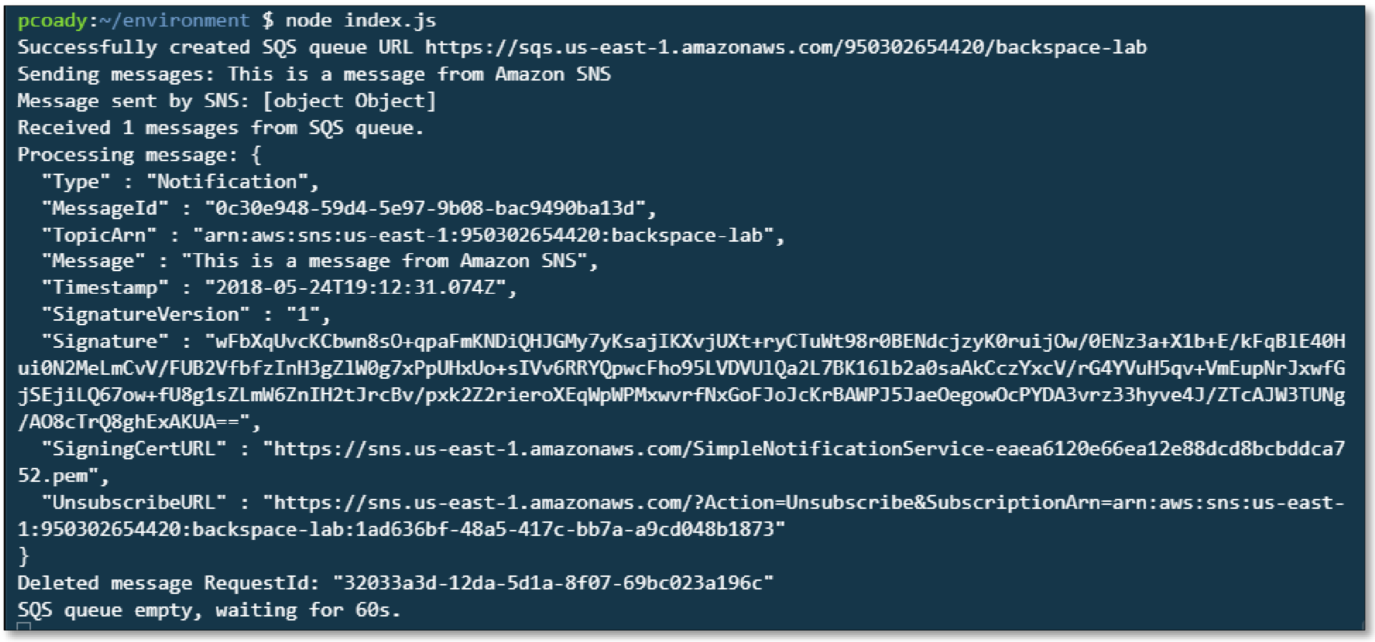
```
sqs.createQueue(params, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else {
      console.log('Successfully created SQS queue URL '+ data.QueueUrl);     //
successful response
      queueUrl = data.QueueUrl;
      waitingSQS = false;
      createMessages(data.QueueUrl);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

Replace the createMessages code with (make sure to replace YOUR_TOPIC_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: 'YOUR_TOPIC_ARN'  }, function(err, data) {
    if (err) {
      console.log(err.stack);
    }
    else{
      console.log('Message sent by SNS: '+ data. MessageId);
    }
  });
}
```

Click Ctrl + S 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/FUB2VfbfzInH3gZ1W0g7xPpUHxUo+sIVv6RRYQpwcFho95LVDVUlQa2L7BK16lb2a0saAkCczYxcV/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 Ctrl + C to stop the application.

Subscribing an SQS Queue to an SNS Topic Using the Python SDK

In this section we will create and subscribe our application to an SNS topic. We will then use the Python 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

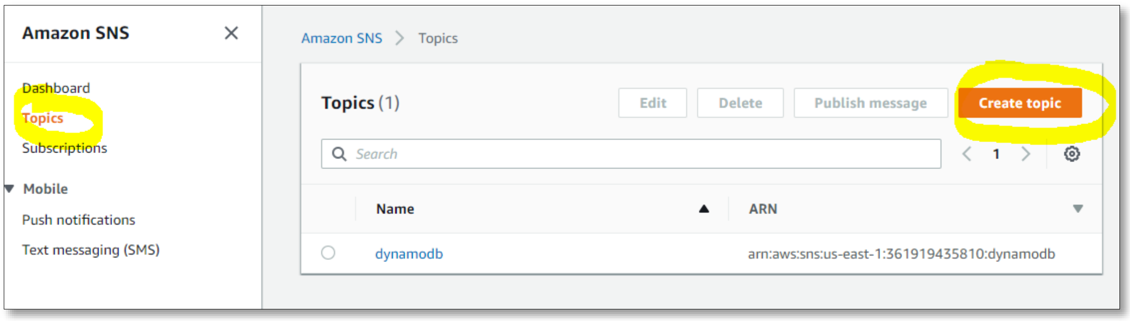
```
# Main program
def main():
    sqs_queue_url = create_sqs_queue('backspace-lab')
    print('Successfully created SQS queue URL '+ sqs_queue_url )
    #create_messages(sqs_queue_url)
    #print('Successfully created messages')
    receive_messages(sqs_queue_url)
```

Creating an SNS Topic

Go to the SNS console.

Go to *Topics*

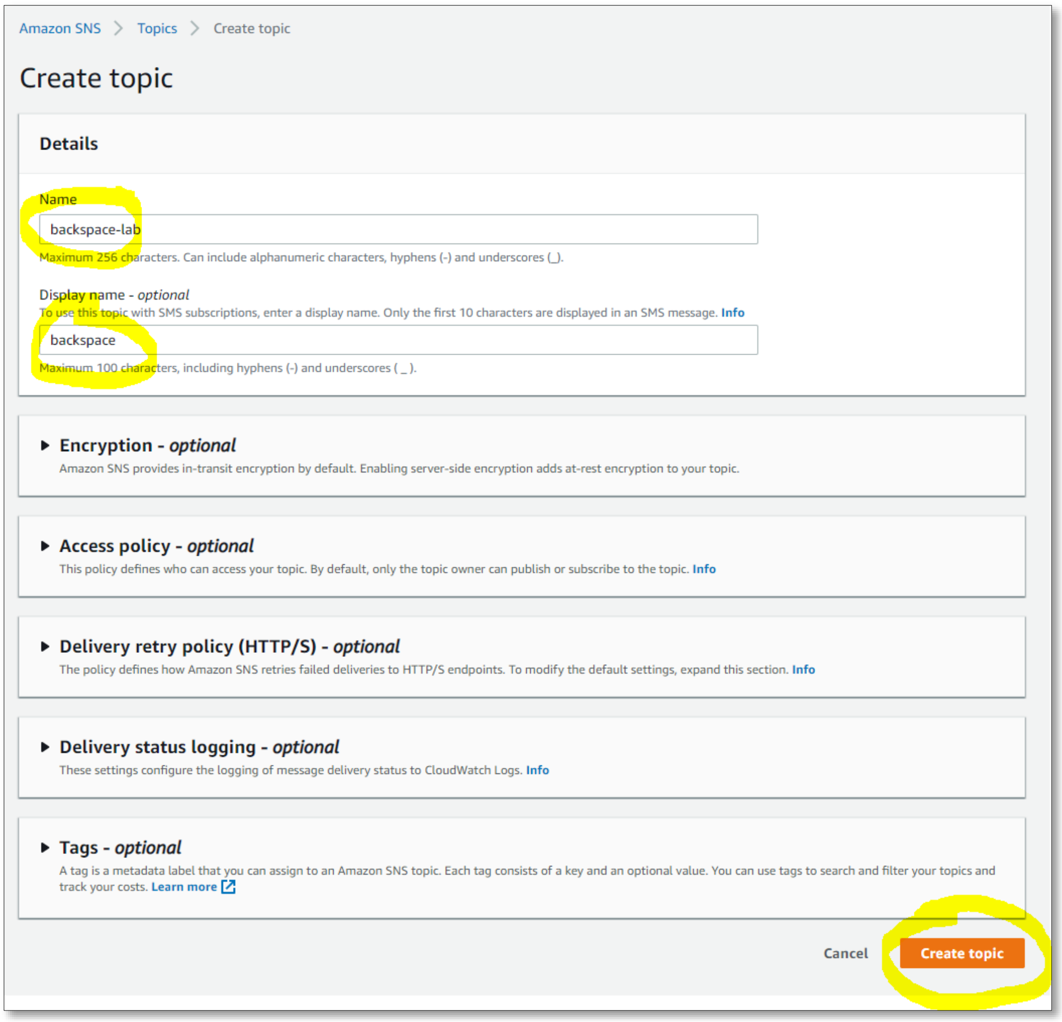
Click *Create Topic*



Name	ARN
dynamodb	arn:aws:sns:us-east-1:361919435810:dynamodb

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

Click *Create Topic*



Amazon SNS > Topics > Create topic

Create topic

Details

Name

backspace-lab

Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_).

Display name - *optional*

To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message. Info

backspace

Maximum 100 characters, including hyphens (-) and underscores (_).

Encryption - *optional*

Amazon SNS provides in-transit encryption by default. Enabling server-side encryption adds at-rest encryption to your topic.

Access policy - *optional*

This policy defines who can access your topic. By default, only the topic owner can publish or subscribe to the topic. Info

Delivery retry policy (HTTP/S) - *optional*

The policy defines how Amazon SNS retries failed deliveries to HTTP/S endpoints. To modify the default settings, expand this section. Info

Delivery status logging - *optional*

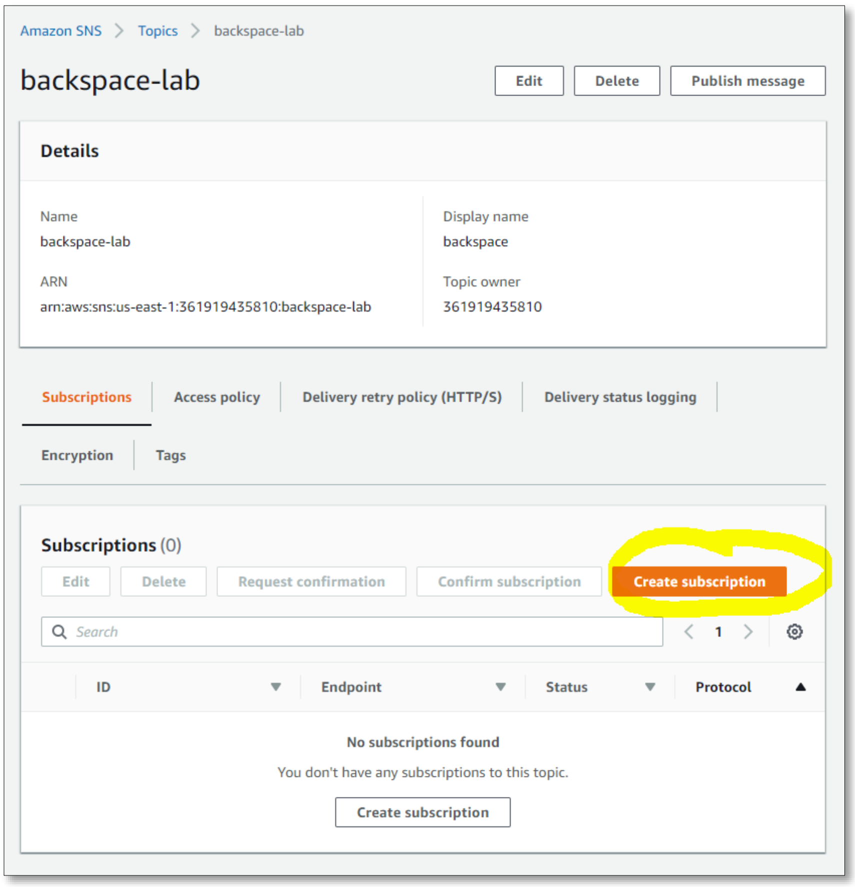
These settings configure the logging of message delivery status to CloudWatch Logs. Info

Tags - *optional*

A tag is a metadata label that you can assign to an Amazon SNS topic. Each tag consists of a key and an optional value. You can use tags to search and filter your topics and track your costs. Learn more

Subscribing an SQS Queue to an SNS Topic

Click 'Create subscription"



Amazon SNS > Topics > backspace-lab

backspace-lab

Details

Name: backspace-lab

ARN: arn:aws:sns:us-east-1:361919435810:backspace-lab

Display name: backspace

Topic owner: 361919435810

Subscriptions (0)

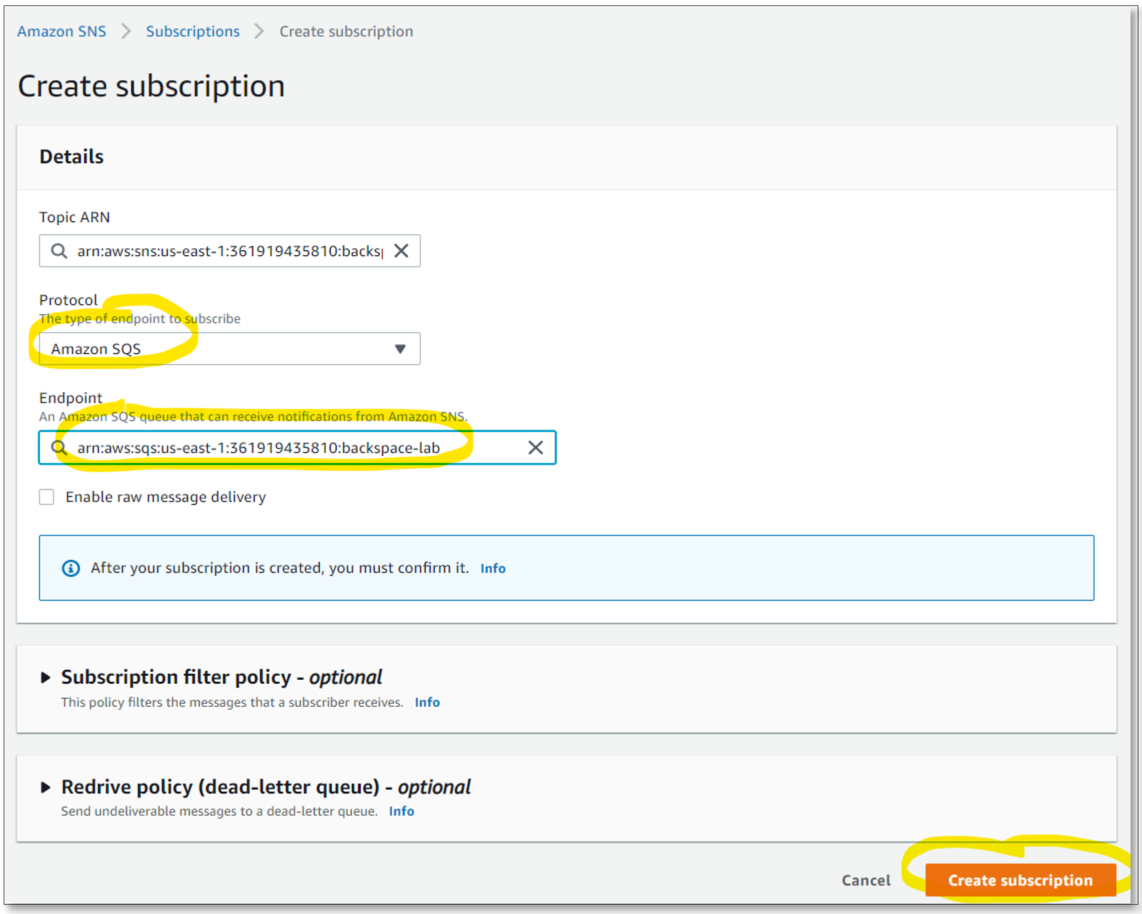
No subscriptions found

You don't have any subscriptions to this topic.

Select ‘Amazon SQS” for protocol

Select our SQS queue ARN.

Click *Create subscription*



Amazon SNS > Subscriptions > Create subscription

Create subscription

Details

Topic ARN

arn:aws:sns:us-east-1:361919435810:backs|

Protocol

The type of endpoint to subscribe

Amazon SQS

Endpoint

An Amazon SQS queue that can receive notifications from Amazon SNS.

arn:aws:sqs:us-east-1:361919435810:backspace-lab

Enable raw message delivery

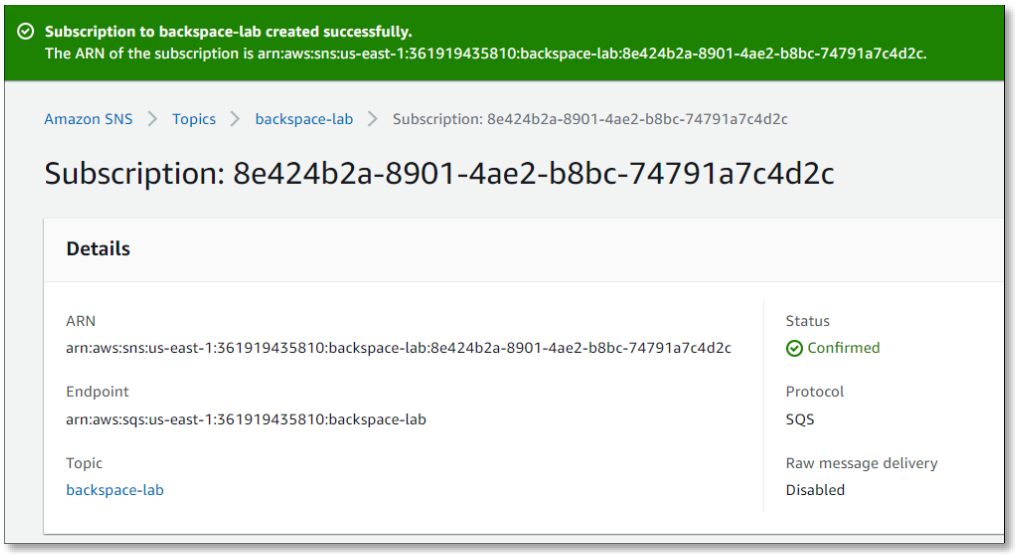
After your subscription is created, you must confirm it. Info

Subscription filter policy - *optional*

This policy filters the messages that a subscriber receives. Info

Redrive policy (dead-letter queue) - *optional*

Send undeliverable messages to a dead-letter queue. Info



Subscription to backspace-lab created successfully.
The ARN of the subscription is arn:aws:sns:us-east-1:361919435810:backspace-lab:8e424b2a-8901-4ae2-b8bc-74791a7c4d2c.

Amazon SNS > Topics > backspace-lab > Subscription: 8e424b2a-8901-4ae2-b8bc-74791a7c4d2c

Subscription: 8e424b2a-8901-4ae2-b8bc-74791a7c4d2c

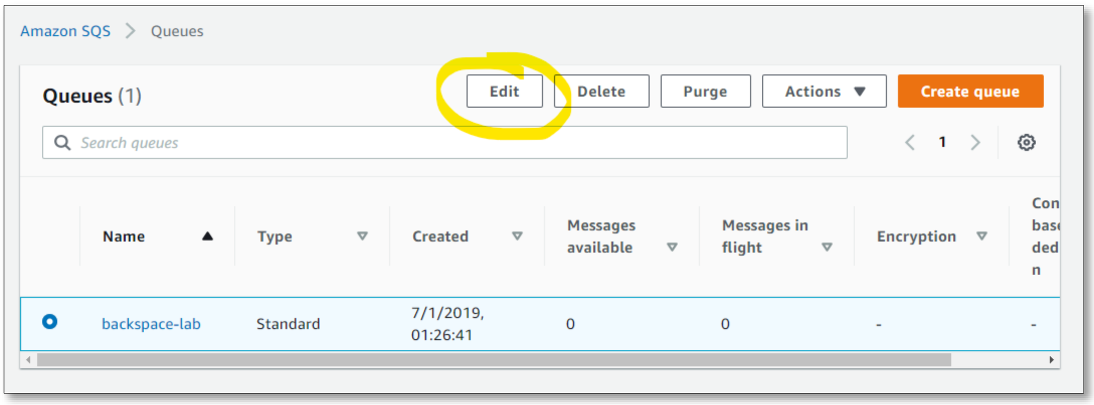
Details

ARN arn:aws:sns:us-east-1:361919435810:backspace-lab:8e424b2a-8901-4ae2-b8bc-74791a7c4d2c	Status Confirmed
Endpoint arn:aws:sqs:us-east-1:361919435810:backspace-lab	Protocol SQS
Topic backspace-lab	Raw message delivery Disabled

Granting SNS Permission to send messages to SQS

Now go back to the SQS console.

Select the queue and click *Edit*



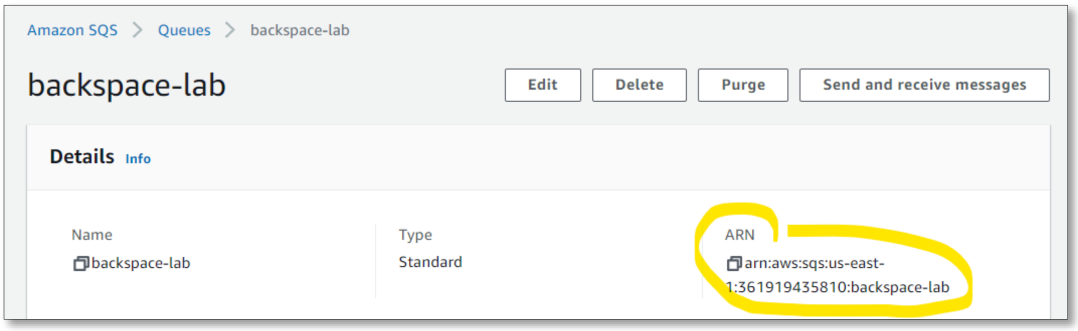
Amazon SQS > Queues

Queues (1)

	Name	Type	Created	Messages available	Messages in flight	Encryption	Con base ded n
●	backspace-lab	Standard	7/1/2019, 01:26:41	0	0	-	-

Paste in the following Access policy.

Make sure your copy and paste in your SQS queue ARN and SNS Topic (not the subscription) ARN.

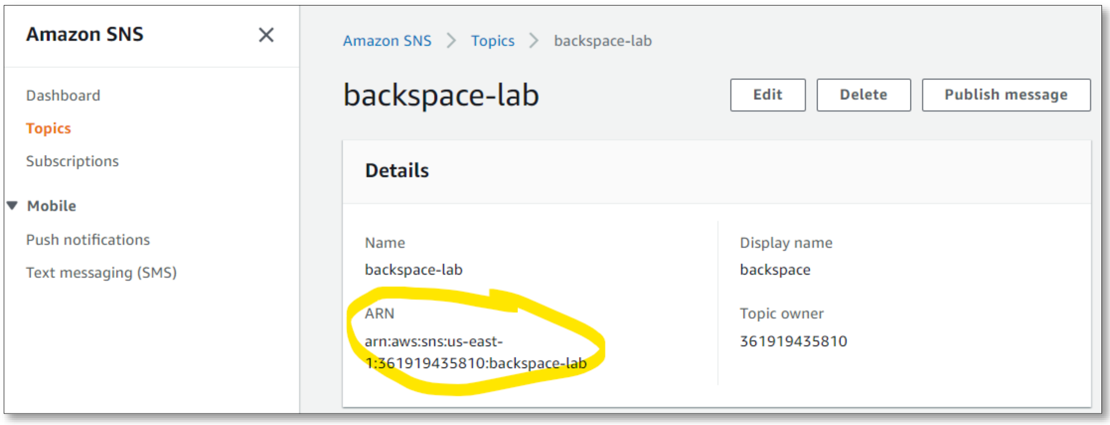


Amazon SQS > Queues > backspace-lab

backspace-lab

Details Info

Name	Type	ARN
backspace-lab	Standard	arn:aws:sqs:us-east-1:361919435810:backspace-lab



Amazon SNS > Topics > backspace-lab

backspace-lab

Details

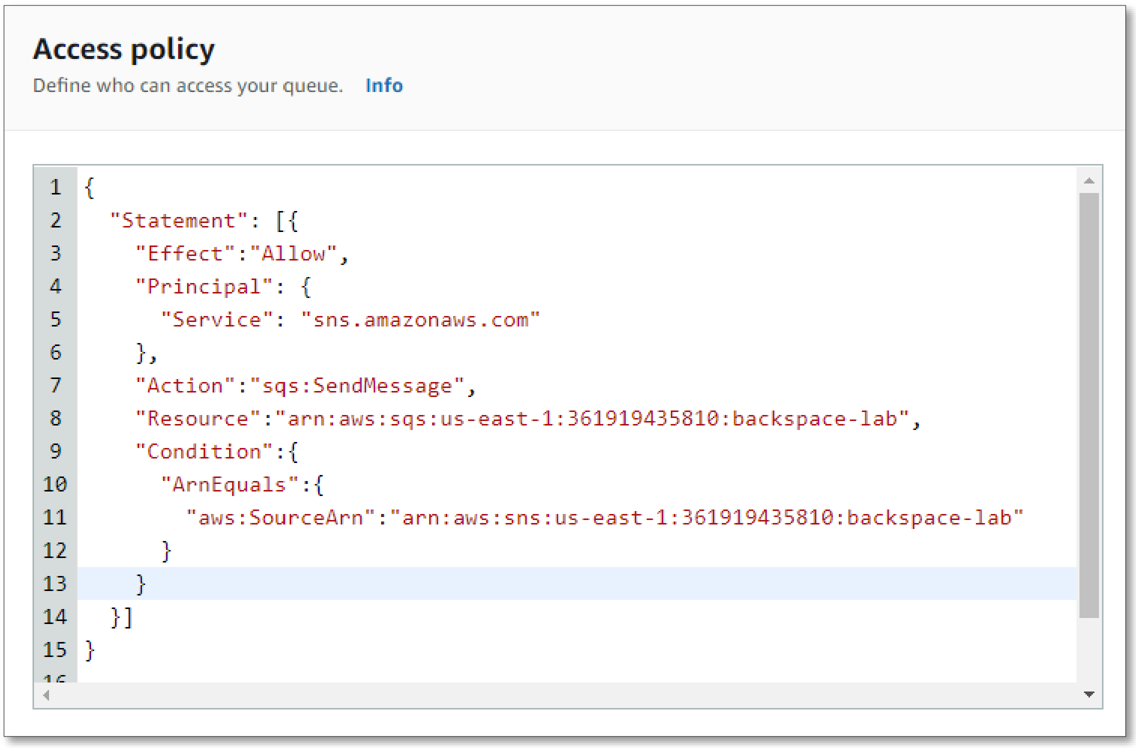
Name: backspace-lab

Display name: backspace

ARN: arn:aws:sns:us-east-1:361919435810:backspace-lab

Topic owner: 361919435810

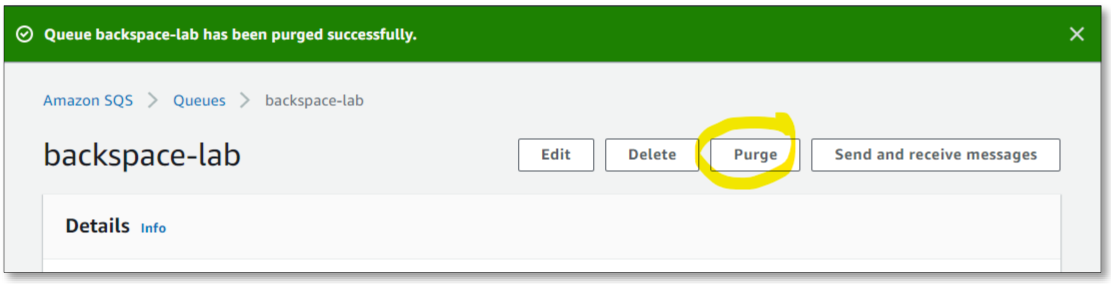
```
{
  "Statement": [{
    "Effect":"Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action":"sqs:SendMessage",
    "Resource":"YOUR_SQS_QUEUE_ARN",
    "Condition":{
      "ArnEquals":{
        "aws:SourceArn":"YOUR_SNS_TOPIC_ARN"
      }
    }
  }]
}
```



```
{
  "Statement": [{
    "Effect":"Allow",
    "Principal": {
      "Service": "sns.amazonaws.com"
    },
    "Action":"sqs:SendMessage",
    "Resource":"arn:aws:sqs:us-east-1:361919435810:backspace-lab",
    "Condition":{
      "ArnEquals":{
        "aws:SourceArn":"arn:aws:sns:us-east-1:361919435810:backspace-lab"
      }
    }
  }]
}
```

Click *Save*

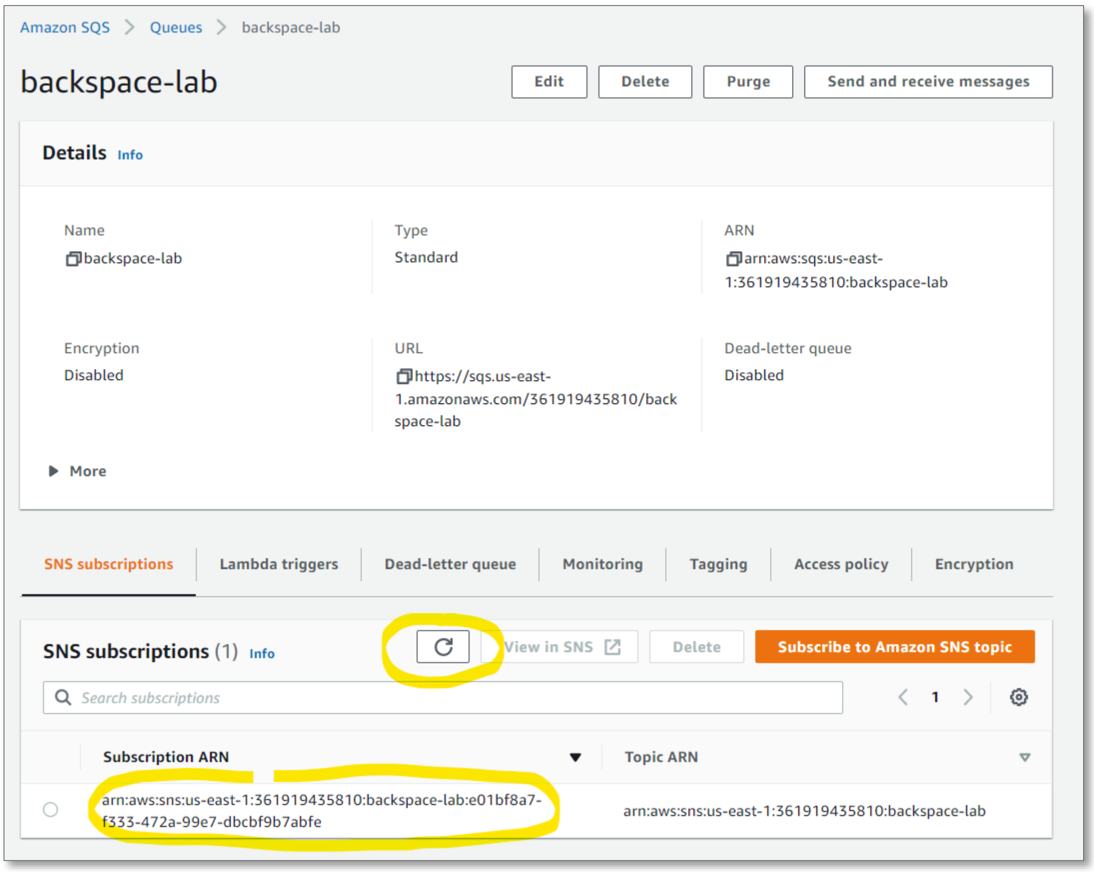
Purge any messages from the *backspace-lab* queue


Queue backspace-lab has been purged successfully.

Amazon SQS > Queues > backspace-lab

backspace-lab

Scroll down to see the SNS subscription has been added.


Amazon SQS > Queues > backspace-lab

backspace-lab

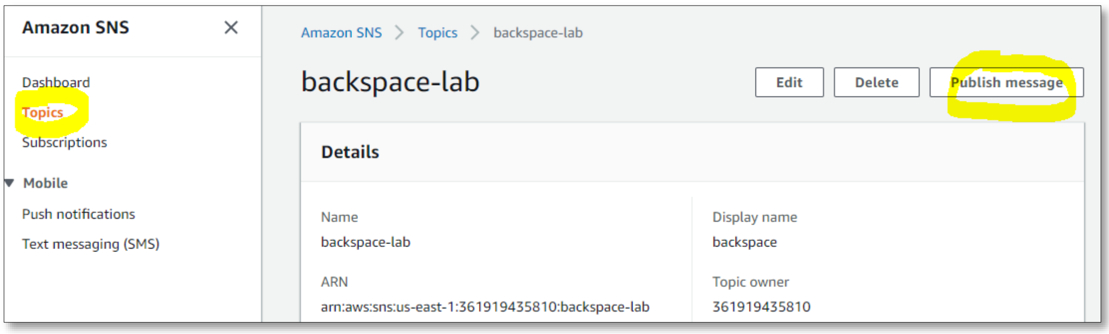
Name backspace-lab	Type Standard	ARN arn:aws:sqs:us-east-1:361919435810:backspace-lab
Encryption Disabled	URL https://sqs.us-east-1.amazonaws.com/361919435810/backspace-lab	Dead-letter queue Disabled

Subscription ARN	Topic ARN
arn:aws:sns:us-east-1:361919435810:backspace-lab:e01bf8a7-f333-472a-99e7-dbcbf9b7abfe	arn:aws:sns:us-east-1:361919435810:backspace-lab

Now go back to the SNS console.

Select *Topics*

Select the topic and click *Publish message*



backspace-lab

Details

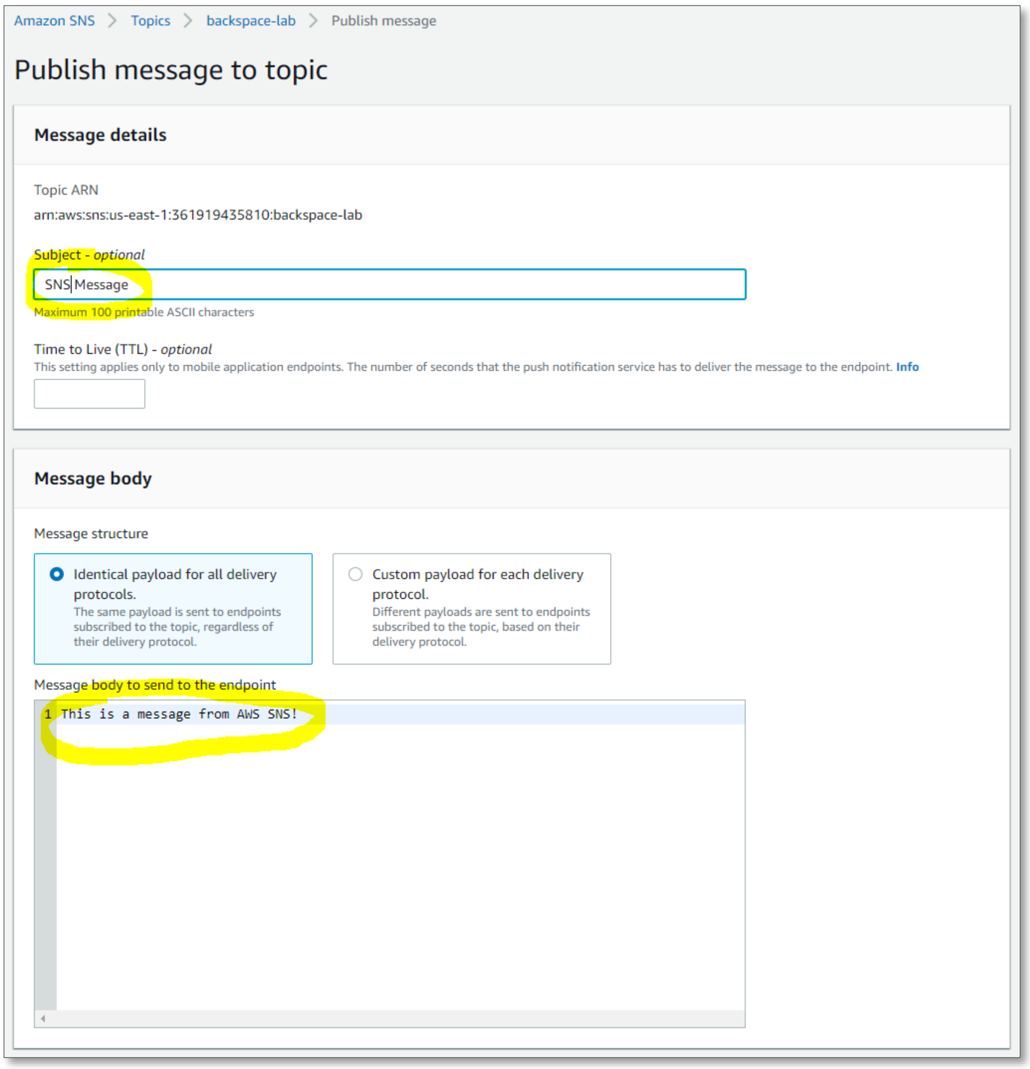
Name
backspace-lab

Display name
backspace

ARN
arn:aws:sns:us-east-1:361919435810:backspace-lab

Topic owner
361919435810

Create a subject and message.



Publish message to topic

Message details

Topic ARN
arn:aws:sns:us-east-1:361919435810:backspace-lab

Subject - *optional*
SNS Message
Maximum 100 printable ASCII characters

Time to Live (TTL) - *optional*
This setting applies only to mobile application endpoints. The number of seconds that the push notification service has to deliver the message to the endpoint. Info

Message body

Message structure

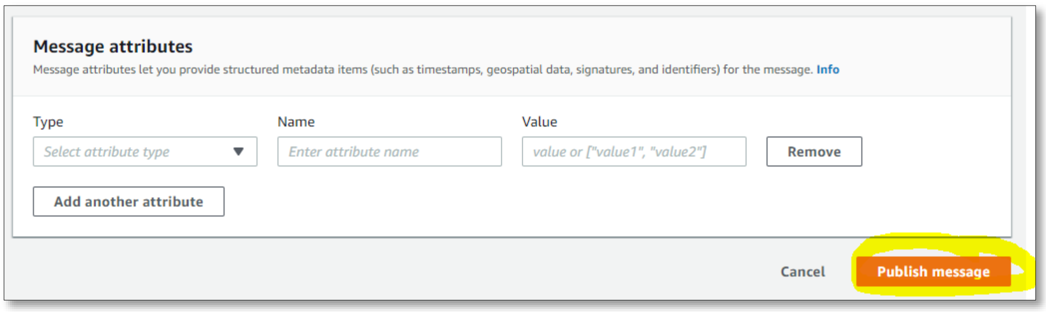
● Identical payload for all delivery protocols.
The same payload is sent to endpoints subscribed to the topic, regardless of their delivery protocol.

○ Custom payload for each delivery protocol.
Different payloads are sent to endpoints subscribed to the topic, based on their delivery protocol.

Message body to send to the endpoint

```
This is a message from AWS SNS!
```

Click *Publish message*



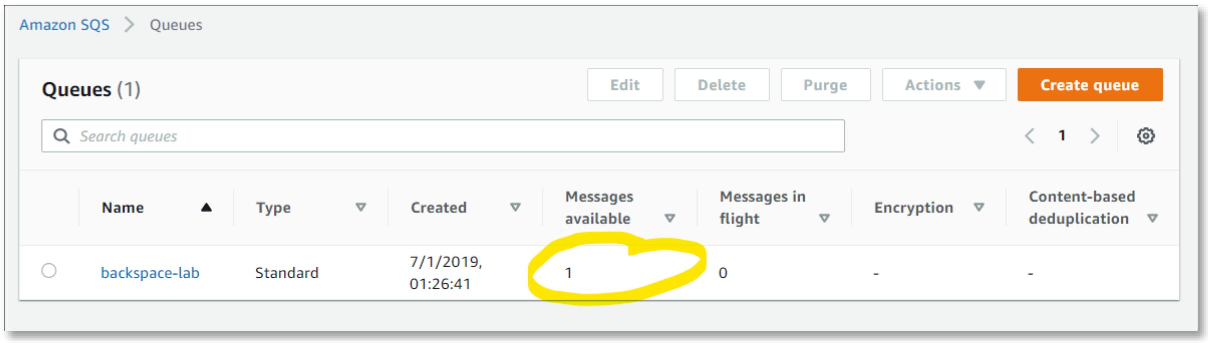
Message attributes

Message attributes let you provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) for the message. Info

Type	Name	Value	
Select attribute type	Enter attribute name	value or ["value1", "value2"]	Remove

Now go back to the SQS console

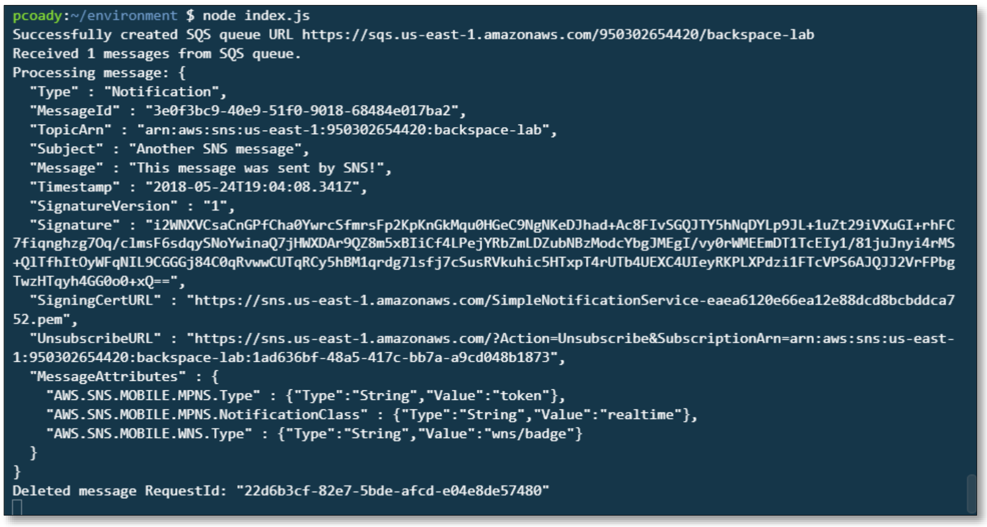
You will now see that the SNS message has been sent to the SQS queue



	Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication
○	backspace-lab	Standard	7/1/2019, 01:26:41	1	0	-	-

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 Python SDK

Uncomment the createMessages call from createQueue

Remove *sqs_queue_url* from the call to *create_messages* .

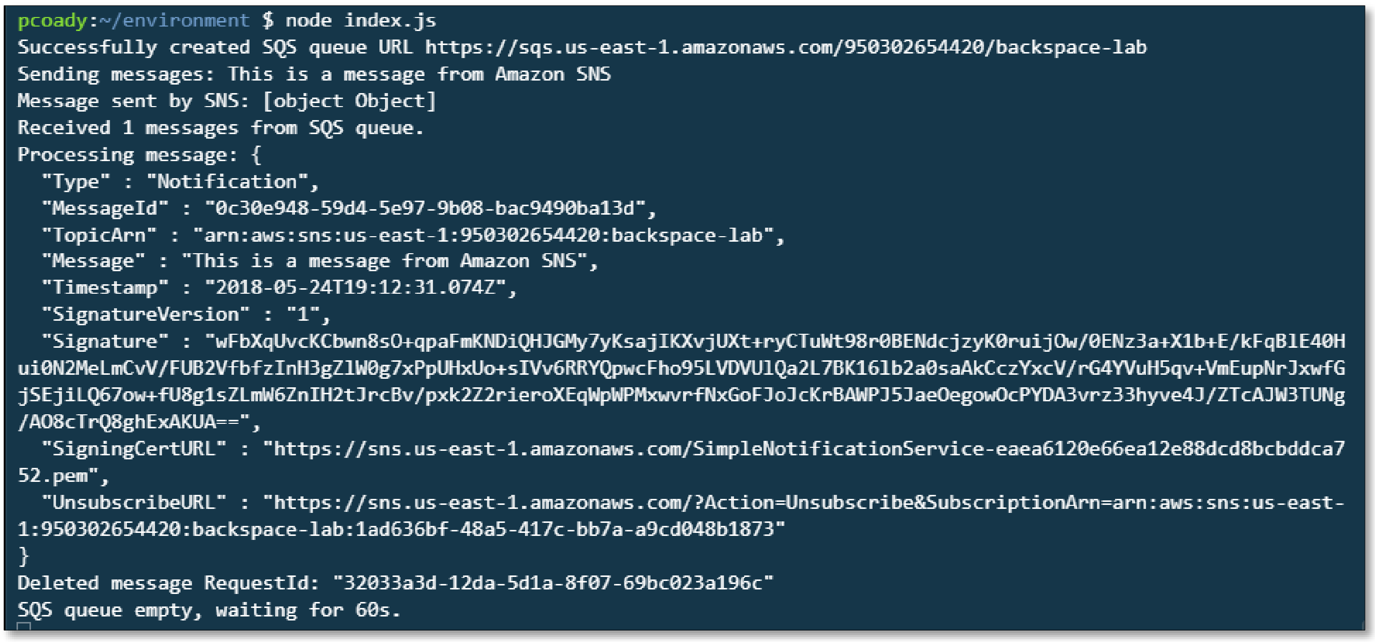
```
# Main program
def main():
    sqs_queue_url = create_sqs_queue('backspace-lab')
    print('Successfully created SQS queue URL '+ sqs_queue_url )
    create_messages()
    print('Successfully created messages')
    receive_messages(sqs_queue_url)
```

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

```
# Create an SNS message
sns = boto3.client('sns', region_name='us-east-1')

def create_messages():
    try:
        data = sns.publish(
            TargetArn = "YOUR_SNS_ARN",
            Subject = "SNS Message",
            Message = "This a message from Amazon SNS!"
        )
        return data['MessageId']
    # An error occurred
    except ParamValidationError as e:
        print("Parameter validation error: %s" % e)
    except ClientError as e:
        print("Client error: %s" % e)
```

Click Ctrl + S 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+sIVv6RRYQpwcFho95LVDVUlQa2L7BK16lb2a0saAkCczYxcV/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 Ctrl + C to stop the application.