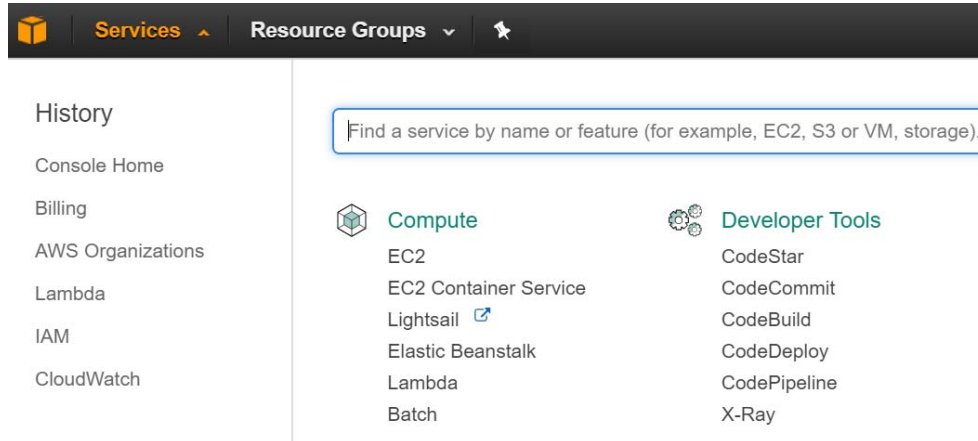


# Run codes with AWS Lambda

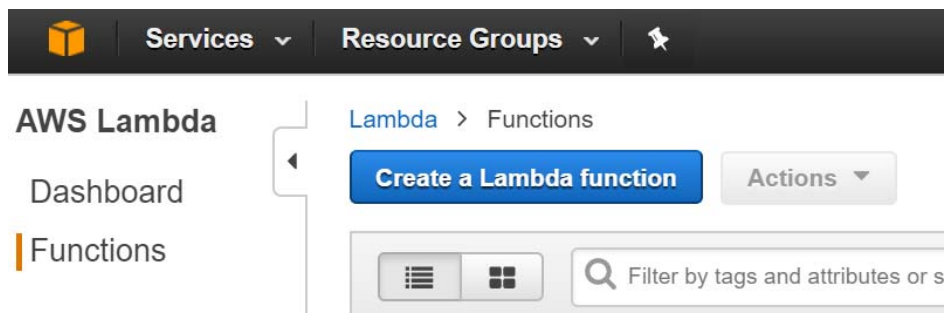
1. Python

- LambdaWithSNS

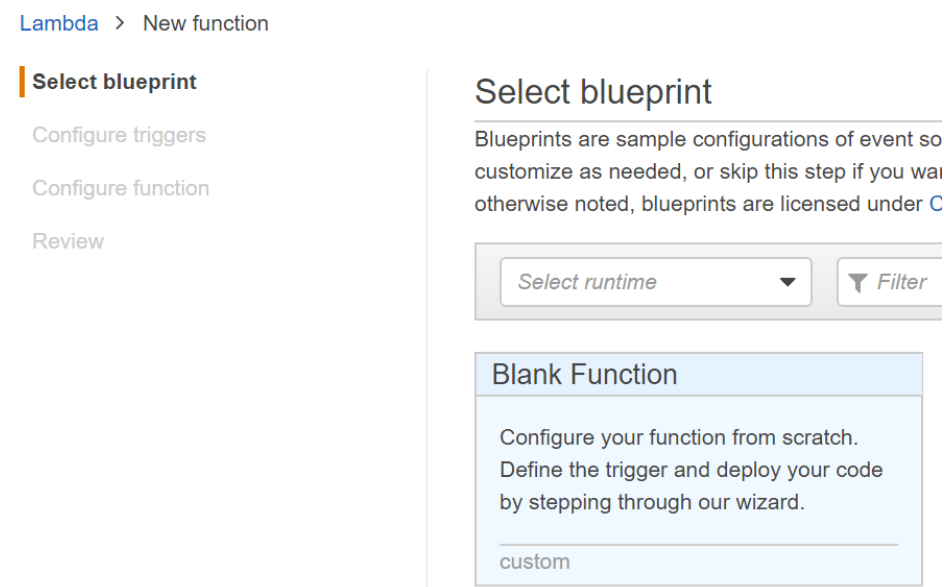
1) Log in AWS Console, Select Services -> Lambda



2) Create a Lambda function



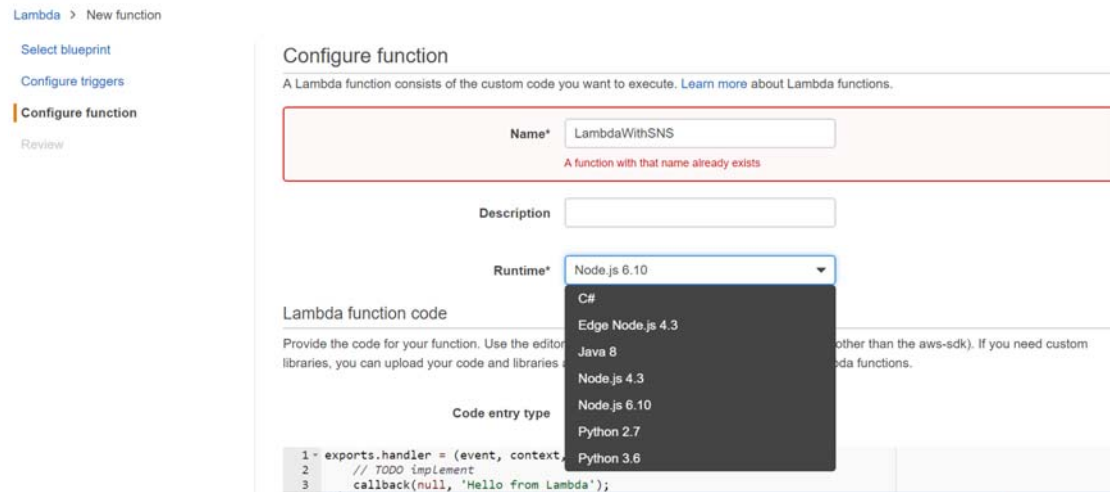
3) Select blueprint -> blank function



4) No need a trigger. Go to next step to configure the function

Name the function and choose the Runtime (Python 2.7)

AWS supports C#, Edge Node.js 4.3, Java 8, Node.js 4.3, Node.js 6.10, Python 2.7, and Python 3.6



## 5) Code input

AWS supports:

Edit code inline – Edit the code under the AWS console. Errors in the code will be shown.

Upload a .ZIP file – If there are multi files, we can zip the files and upload them. After creating the function, the codes will be shown in AWS console.

Upload a file from Amazon S3

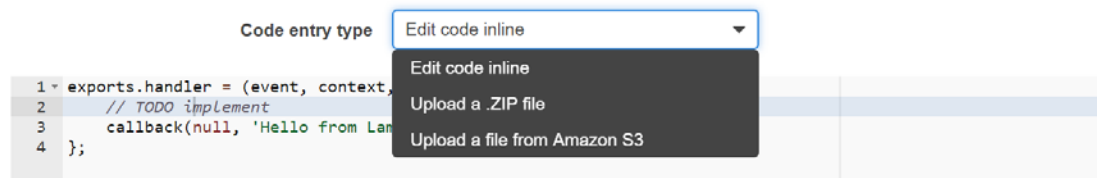
For LambdaWithSNS, we have 1 file. Choose Edit code inline and input the code.

```
from __future__ import print_function
import json
print('Loading function')
```

```
def lambda_handler(event, context):
    print("Received event: " + json.dumps(event, indent=2))
    message = event['Records'][0]['Sns']['Message']
    print("From SNS: " + message)
    return message
```

## Lambda function code

Provide the code for your function. Use the editor if your code does not require custom libraries (other than the aws-sdk). If you need custom libraries, you can upload your code and libraries as a .ZIP file. [Learn more](#) about deploying Lambda functions.



## 6) Lambda function handler and role

Handler: The filename.handler-method value in your function. For example, "main.handler" would call the handler method defined in main.py.

If the method name in the code is not lambda\_handler shown in below figure, need change either one to keep constant.

Role:

Choose an existing role: Defines the permissions of your function. Note that new roles may

not be available for a few minutes after creation. Learn more about Lambda execution roles.  
Existing role: need create a role before using 'Choose an existing role'. Here use AWSPowerUser.

#### Lambda function handler and role

Handler\*

Role\* Choose an existing role

Existing role\*

Tags

Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. [Learn more](#) about Lambda execution roles.

7) Then create the function and test it.

The default test event is Hello World. Need choose different one to edit according to your needs.

Input test event

Use the editor below to enter an event to test your function with. You can edit the event again by choosing **Configure test event** in the Actions list. Note that changes to the event will only be saved locally.

Sample event template Hello World

```
1 {  
2   "key3": "value3",  
3   "key2": "value2",  
4   "key1": "value1"  
5 }
```

If there is any error, AWS console will report it.

Input test event

There is an error in your JSON event. Please correct it before saving.

Use the editor below to enter an event to test your function with. You can edit the event again by choosing **Configure test event** in the Actions list. Note that changes to the event will only be saved locally.

Sample event template Hello World

```
1 {  
2   "first_name": "Y",  
3   "last_name": "Du",  
4 }
```

For LambdaWithSNS, select SNS as the Sample event template. We can modify the event as needed.

```
{  
  "Records": [  
    {  
      "EventVersion": "1.0",  
      "EventSubscriptionArn": "arn:aws:sns:EXAMPLE",  
      "EventSource": "aws:sns",
```

```

    "Sns": {
      "SignatureVersion": "1",
      "Timestamp": "1970-01-01T00:00:00.000Z",
      "Signature": "EXAMPLE",
      "SigningCertUrl": "EXAMPLE",
      "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
      "Message": "Hello from SNS!",           // message = event['Records'][0]['Sns']['Message']
      "MessageAttributes": {
        "Test": {
          "Type": "String",
          "Value": "TestString"
        },
        "TestBinary": {
          "Type": "Binary",
          "Value": "TestBinary"
        }
      },
      "Type": "Notification",
      "UnsubscribeUrl": "EXAMPLE",
      "TopicArn": "arn:aws:sns:EXAMPLE",
      "Subject": "TestInvoke"
    }
  }
}
]
}

```

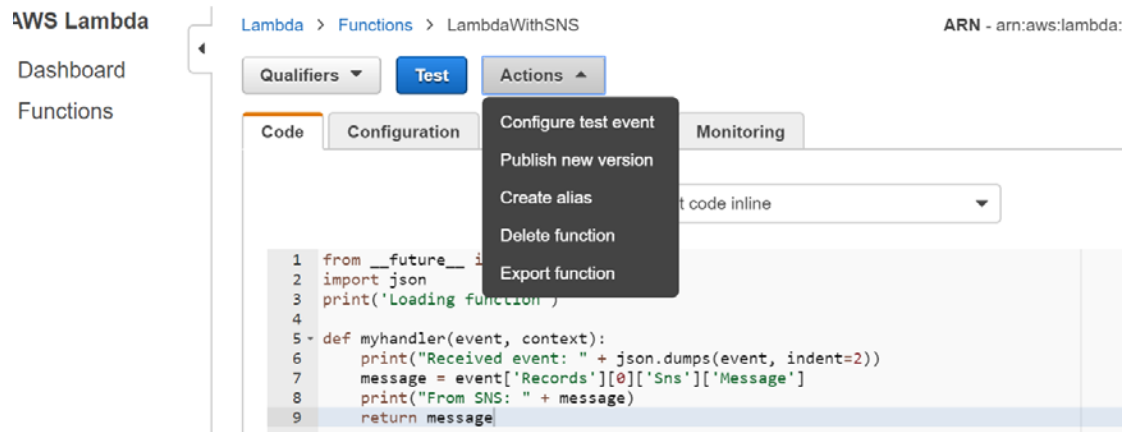
Sample event template SNS

```

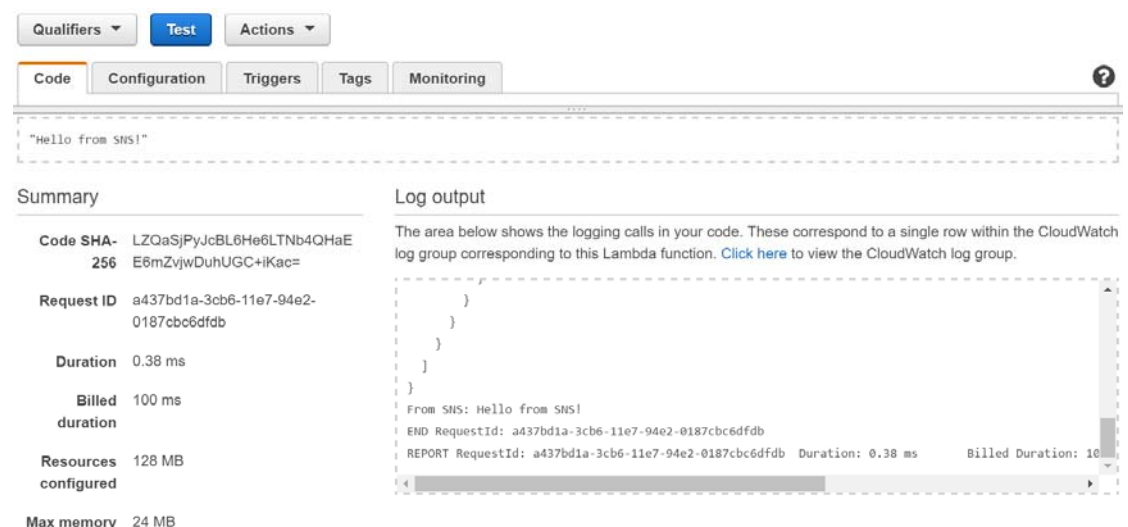
1  {
2  "Records": [
3  {
4    "EventVersion": "1.0",
5    "EventSubscriptionArn": "arn:aws:sns:EXAMPLE",
6    "EventSource": "aws:sns",
7    "Sns": {
8      "SignatureVersion": "1",
9      "Timestamp": "1970-01-01T00:00:00.000Z",
10     "Signature": "EXAMPLE",
11     "SigningCertUrl": "EXAMPLE",
12     "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
13     "Message": "Hello from SNS!",
14     "MessageAttributes": {
15       "Test": {
16         "Type": "String",
17         "Value": "TestString"
18       },
19       "TestBinary": {
20         "Type": "Binary",
21         "Value": "TestBinary"
22       }
23     },
24     "Type": "Notification",
25     "UnsubscribeUrl": "EXAMPLE",
26     "TopicArn": "arn:aws:sns:EXAMPLE",
27     "Subject": "TestInvoke"
28   }
29 }
30 ]

```

8) Before running the test, we can re-configure the test event.



- 9) After running the test, the test results will be shown to list the duration, memory usage and the log output.



## ● ProcessDynamoDBStream

### 1) Code input

```
from __future__ import print_function
```

```
def lambda_handler(event, context):
    for record in event['Records']:
        print(record['eventID'])
        print(record['eventName'])
        print('Successfully processed %s records.' % str(len(event['Records'])))
```

### 2) DynamoDB event

```
{
    "Records": [
        {
            "eventID": "1",
            "eventVersion": "1.0",
            "dynamodb": {
                "Keys": {
                    "Id": {
```

```

        "N": "101"
      }
    },
    "NewImage": {
      "Message": {
        "S": "New item!"
      },
      "Id": {
        "N": "101"
      }
    },
    "StreamViewType": "NEW_AND_OLD_IMAGES",
    "SequenceNumber": "111",
    "SizeBytes": 26
  },
  "awsRegion": "us-west-2",
  "eventName": "INSERT",
  "eventSourceARN":
"arn:aws:dynamodb:us-west-2:account-id:table/ExampleTableWithStream/stream/2015-06-27T0
0:48:05.899",
  "eventSource": "aws:dynamodb"
},
...
]
}

```

### 3) Result

Summary		Log output
<b>Code SHA-256</b>	mTCRt6ZHq8zjoD7g9pkiyvLHi5AqUGyVo90BCOI0VJc=	<p>The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. <a href="#">Click here</a> to view the CloudWatch log group.</p> <pre> START RequestId: ba22d30d-3cb9-11e7-95dc-439194c687bd Version: \$LATEST 1 INSERT 2 MODIFY 3 REMOVE Successfully processed 3 records. </pre>
<b>Request ID</b>	ba22d30d-3cb9-11e7-95dc-439194c687bd	
<b>Duration</b>	12.97 ms	
<b>Billed duration</b>	100 ms	
<b>Resources configured</b>	128 MB	
<b>Max memory used</b>	22 MB	

## 2. Node.js

### 1) Code input

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

```
exports.handler = function(event, context, callback) {
```

```
  var bucketName = process.env.S3_BUCKET;
```

```
  callback(null, bucketName);
```

```

    }
2) S3 put event
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "s3": {
        "configurationId": "testConfigRule",
        "object": {
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901",
          "key": "index.js",
          "size": 1024
        },
        "bucket": {
          "arn": "arn:aws:s3:::mybucket",
          "name": "Nodejs",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          }
        },
        "s3SchemaVersion": "1.0"
      },
      "responseElements": {
        "x-amz-id-2":
"EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH",
        "x-amz-request-id": "EXAMPLE123456789"
      },
      "awsRegion": "us-east-1",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "eventSource": "aws:s3"
    }
  ]
}
3) Result

```

### Summary

Code SHA-256

16QrT8xyqHvszHF7sINzpeBJpQaKkw/fjaeNVt677WQ=

Request ID

74e3e4e1-3cbc-11e7-85f2-95b3202e9f85

Duration

21.83 ms

Billed duration

100 ms

Resources configured

128 MB

Max memory used

32 MB

### Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```

START RequestId: 74e3e4e1-3cbc-11e7-85f2-95b3202e9f85 Version: $LATEST
END RequestId: 74e3e4e1-3cbc-11e7-85f2-95b3202e9f85
REPORT RequestId: 74e3e4e1-3cbc-11e7-85f2-95b3202e9f85  Duration: 21.83 ms    Billed Duration: 100 ms

```

### 3. Create an Identity and Access Management (IAM) role



## Security, Identity & Compli...

- IAM
- Inspector
- Certificate Manager
- Directory Service
- WAF & Shield
- Compliance Reports

#### 1) Select AWS Lambda

### Create role

Step 1 : Select role type

Step 2 : Establish trust

Step 3 : Attach policy

Step 4 : Set role name and review

### Select role type

• AWS Service Role

Amazon EC2

Allows EC2 instances to call AWS services on your behalf.

Select

AWS Directory Service

Allows AWS Directory Service to manage access for existing directory users and groups to AWS services.

Select

AWS Lambda

Allows Lambda Function to call AWS services on your behalf.

Select

#### 2) Search and select PowerUserAccess to create a new role

### Attach Policy

Select one or more policies to attach. Each role can have up to 10 policies attached.

Filter: Policy Type

power

Showing 6 res

	Policy Name	Attached Entities	Creation Time	Edited Time
<input type="checkbox"/>	 PowerUserAccess	2	2015-02-06 10:39 PDT	2016-12-06 10:11 PDT