lab

lab title

Programming AWS Lambda

V1.00

Course title

AWS Certified Developer Associate

Table of Contents

Contents

About the Lab

These lab notes are to support the instructional videos on Programming AWS Lambda in the BackSpace AWS Certified Developer course.

In this lab we will:

- Create an AWS Lambda function
- Test the AWS Lambda function
- Invoke the Lambda function through a browser application
- Delete the Lambda function through a browser application.
- Processing Images using AWS Lambda Function with S3 Events

Please refer to the AWS JavaScript SDK documentation at:

http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/Lambda.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 and Testing an AWS Lambda Function

In this section we will create an AWS Lambda NodeJS function using the console and create an IAM role for the Lambda function and for our application to access Lambda.

Go to the Lambda console

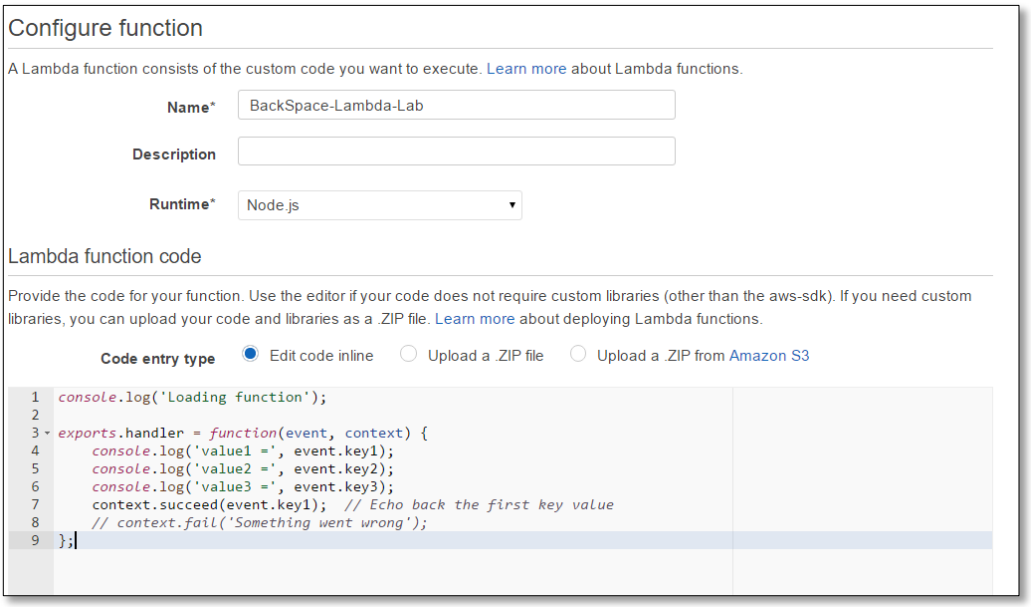
Click Get Started Now

Click Skip

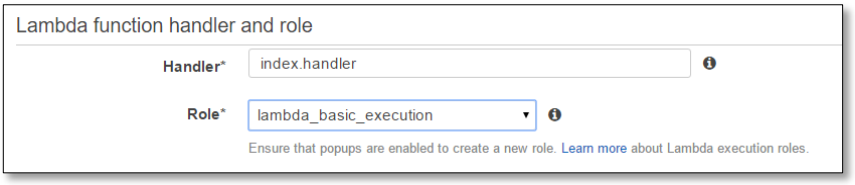
Give the function the name BackSpace-Lambda-Lab

Paste the following code:

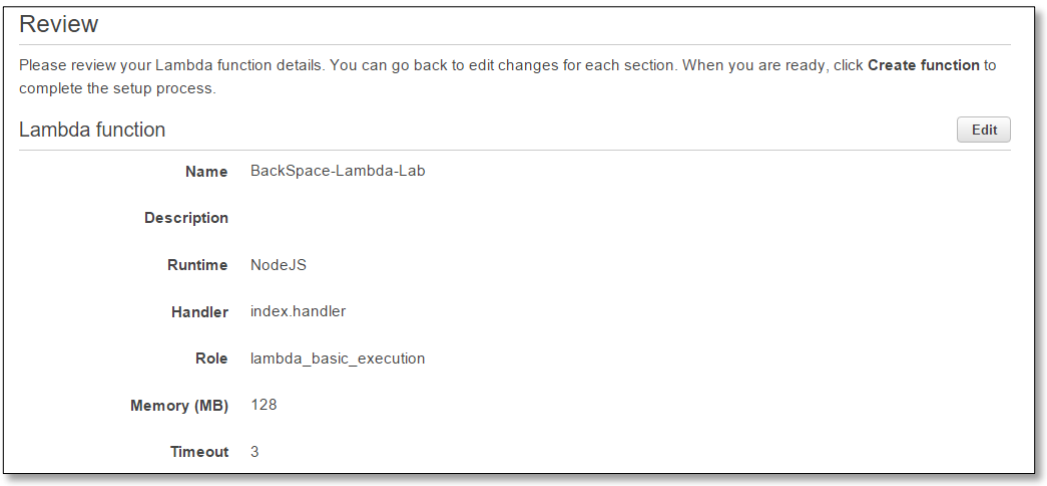
```
console.log('Loading function');

exports.handler = function(event, context) {
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    context.succeed(event.key1);  // Echo back the first key value
    // context.fail('Something went wrong');
};
```



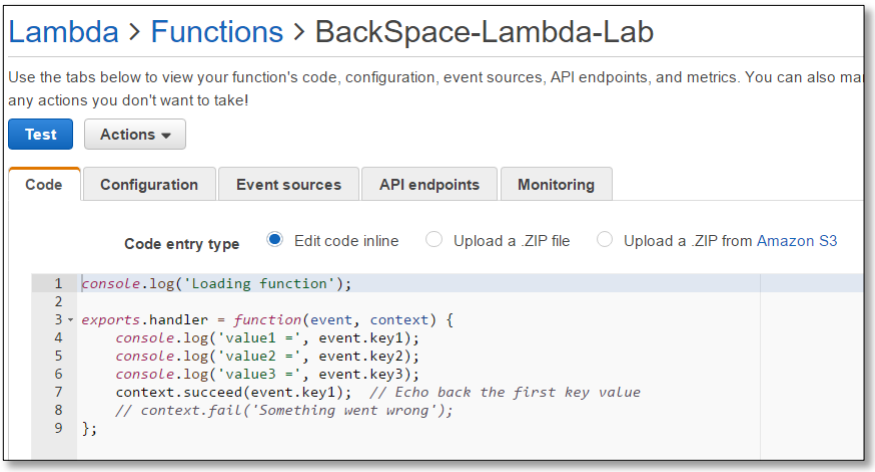
Select Create new role – Basic execution role



Click Next



Click Create Function



Click Test

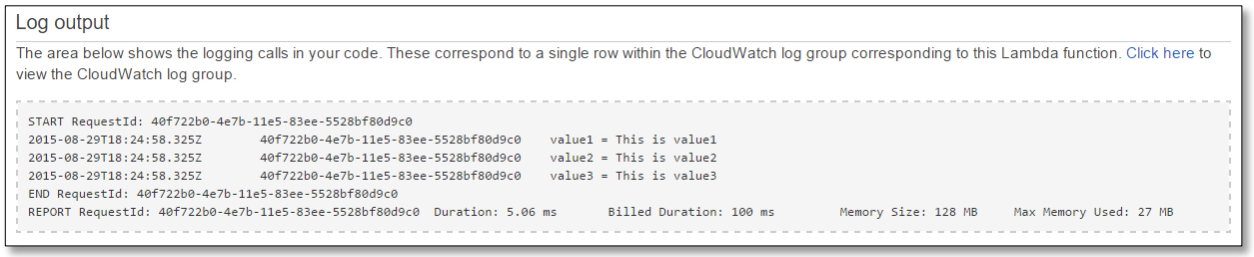
Use the Hello World sample event.

Edit it to:

```
{
  "key3": "This is value3",
  "key2": "This is value2",
  "key1": "This is value1"
}
```

Click Submit

The log output will show the console output from the Lambda code.



Creating an AWS Lambda enabled Browser Application

In this section we will create an IAM role for federated users to access AWS Lambda. We will then extend our browser application from the “Creating a Low Cost Sync Database for JavaScript Applications with AWS” lab to use the role to access Lambda.

Go to the IAM console.

Create a new policy called facebook-lambda-S3

Make sure you change `YourBucketName` with your bucket name

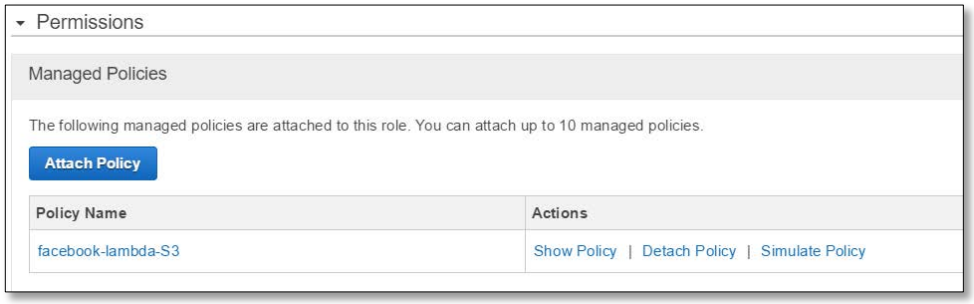
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::YourBucketName/facebook-${graph.facebook.com:id}/*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::YourBucketName"
            ],
            "Effect": "Allow",
            "Condition": {
                "StringEquals": {
                    "s3:prefix": "facebook-${graph.facebook.com:id}"
                }
```

```
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:Invoke*",
                "lambda:List*",
                "lambda:Get*"
            ],
            "Resource": "YOUR_LAMBDA_ARN"
        }
    ]
}
```

Edit your facebookBackSpace role created for the previous Facebook app lab.

Detach the current policy.

Attach the new facebook-lambda-S3 policy



Open the browser application from the “Creating a Low Cost Sync Database for JavaScript Applications with AWS” lab in the Atom IDE.

Run your application in the browser. Create and delete a shopping list to check it is still working with the new IAM role.

Now add some extra buttons to index.html test our Lambda function.

```
          <!-- Shopping List Buttons -->
          <p>
            <button type="button" class="btn btn-success btn-lg" data-toggle="modal" data-
target="#createModal">
              Create List
            </button>
            <button type="button" class="btn btn-primary btn-lg" data-toggle="modal" data-
target="#readModal">
              Read List
            </button>
```

```
            <button type="button" class="btn btn-primary btn-lg" id="btnSync">
              Sync with Cloud
            </button>
            <button type="button" class="btn btn-danger btn-lg" data-toggle="modal" data-
target="#deleteModal">
              Delete List
            </button>
            <hr>
            <div>
              <button id="btnInvoke" type="button" class="btn btn-success btn-lg">
                Invoke Lambda Function
              </button>
              <button id="btnGet" type="button" class="btn btn-default btn-lg">
                Get Lambda Function
              </button>
            </div>
            <div id= "pageSpinner" class="alert alert-info" role="alert">
              <span class="glyphicon glyphicon-refresh glyphicon-refresh-animate"></span>
Working, please wait...
            </div>
          </p>
```

Now add an AWS.Lambda object declaration to app.js after the S3 object declaration.

```
var S3 = new AWS.S3({
    params: {
        Bucket: bucketName
    }
});
var lambda = new AWS.Lambda();
```

Now add add events for our new buttons to the button events.

```
/*******************  Button Events   ************************************/

$('#btnCreate').on('click', function (event) {
  createShoppingList();
});

$('#btnSync').on('click', function (event) {
  console.log('Starting sync...');
  syncShoppingList();
});
```

```
$('#btnDelete').on('click', function (event) {
  deleteShoppingList($('#dropdownDeleteList')[0].selectedIndex);
});

$('#dropdownReadList').on('click', function (event) {
  readShoppingList($('#dropdownReadList')[0].selectedIndex);
});

$('#btnInvoke').on('click', function (event) {
  invokeLambda();
});

$('#btnGet').on('click', function (event) {
  getLambda();
});
```

Now add lambda credentials object after the S3 credentials object.

```
 S3.config.credentials = new AWS.WebIdentityCredentials({
      ProviderId: 'graph.facebook.com',
      RoleArn: roleArn,
      WebIdentityToken: fbToken
  });
  lambda.config.credentials = new AWS.WebIdentityCredentials({
      ProviderId: 'graph.facebook.com',
      RoleArn: roleArn,
      WebIdentityToken: fbToken
  });
```

Now create the invokeLambda function in app.js

```
function invokeLambda(){
  var params = {
    FunctionName: 'BackSpace-Lambda-Lab', /* required */
    InvocationType: 'RequestResponse',
    LogType: 'Tail',
    Payload: '{"key3": "This is value3 from Browser","key2": "This is value2 from
Browser","key1": "This is value1 from Browser"}'
  };
  lambda.invoke(params, function(err, data) {
    if (err) {
      console.log(err, err.stack); // an error occurred
```

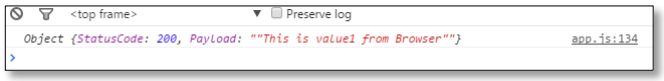
```
      growl('danger', 'AWS Lambda Error', 'Failed to invoke AWS Lambda function.');
    }
    else {
      console.log(data);           // successful response
      growl('success','AWS Lambda','Invoked AWS Lambda function.');
    }
  });
}
```

Upload app.js to S3 again.

Refresh the app browser screen. Press F12 to see console output.

Click on Invoke Lambda Function

You will get a success notification and console output of the response received from Lambda.



Now create the getLambda function

```
function getLambda(){
  var params = {
    FunctionName: 'BackSpace-Lambda-Lab' /* required */
  };
  lambda.getFunction(params, function(err, data) {
    if (err) {
      console.log(err, err.stack); // an error occurred
      growl('danger', 'AWS Lambda Error', 'Failed to get AWS Lambda function.');
    }
    else {
      console.log(data);           // successful response
      growl('success','AWS Lambda','Got AWS Lambda function.');
    }
  });
}
```

Upload app.js to S3 again.

Refresh the app browser screen. Press F12 to see console output.

Click on Get Lambda Function

You will get a success notification and console output of the response received from Lambda.

Processing Images using a Lambda Function with S3 Events

In this section we will create a NodeJS application using the ImageMagick / GraphicsMagick package to process images uploaded to S3. We will then upload it to AWS Lambda to create a Lambda function. The process will be automated using S3 events.

Go to the S3 console.

Create two buckets. The second bucket has the same name with resized on the end:

Upload an image file to the first bucket.

Go to https://nodejs.org/download/

Install NodeJs.

Open the NodeJS Command Prompt

Create a directory for your Lambda app

Install GraphicsMagick and Async npm modules

```
npm install async gm
```

Your directory will now look like:

```
/node_modules/gm
/node_modules/async
```

The AWS Javascript SDK is already installed on Lambda so we don't need to install it.

Now we will create the Lambda function code.

Copy the following code into Atom IDE and save in the directory as index.js

```
// dependencies
var async = require('async');
var AWS = require('aws-sdk');
var gm = require('gm')
            .subClass({ imageMagick: true }); // Enable ImageMagick integration.
var util = require('util');

// constants
var MAX_WIDTH  = 100;
var MAX_HEIGHT = 100;

// get reference to S3 client
var s3 = new AWS.S3();
exports.handler = function(event, context) {
        // Read options from the event.
        console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
        var srcBucket = event.Records[0].s3.bucket.name;
        // Object key may have spaces or unicode non-ASCII characters.
    var srcKey    =
    decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
        var dstBucket = srcBucket + "resized";
        var dstKey    = "resized-" + srcKey;

        // Sanity check: validate that source and destination are different buckets.
        if (srcBucket == dstBucket) {
                console.error("Destination bucket must not match source bucket.");
                return;
        }

        // Infer the image type.
        var typeMatch = srcKey.match(/\.([^.]*)$/);
        if (!typeMatch) {
                console.error('unable to infer image type for key ' + srcKey);
                return;
        }
        var imageType = typeMatch[1];
        if (imageType != "jpg" && imageType != "png") {
                console.log('skipping non-image ' + srcKey);
                return;
        }

        // Download the image from S3, transform, and upload to a different S3 bucket.
        async.waterfall([
                function download(next) {
                        // Download the image from S3 into a buffer.
                        s3.getObject({
                                        Bucket: srcBucket,
                                        Key: srcKey
```

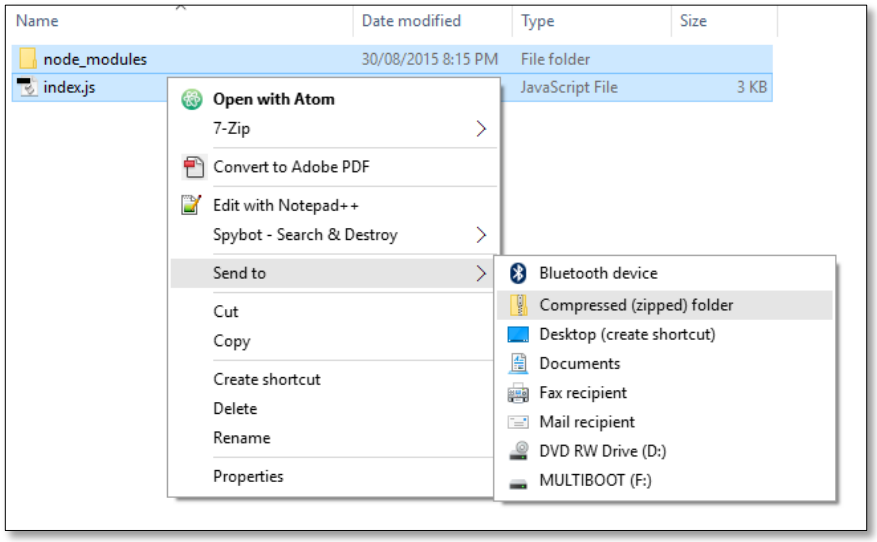
```
                                },
                                next);
                        },
                function transform(response, next) {
                        gm(response.Body).size(function(err, size) {
                                // Infer the scaling factor to avoid stretching the image
unnaturally.
                                var scalingFactor = Math.min(
                                        MAX_WIDTH / size.width,
                                        MAX_HEIGHT / size.height
                                );
                                var width  = scalingFactor * size.width;
                                var height = scalingFactor * size.height;

                                // Transform the image buffer in memory.
                                this.resize(width, height)
                                        .toBuffer(imageType, function(err, buffer) {
                                                if (err) {
                                                        next(err);
                                                } else {
                                                        next(null, response.ContentType,
buffer);
                                                }
                                        });
                        });
                },
                function upload(contentType, data, next) {
                        // Stream the transformed image to a different S3 bucket.
                        s3.putObject({
                                        Bucket: dstBucket,
                                        Key: dstKey,
                                        Body: data,
                                        ContentType: contentType
                                },
                                next);
                        }
                ], function (err) {
                        if (err) {
                                console.error(
                                        'Unable to resize ' + srcBucket + '/' + srcKey +
                                        ' and upload to ' + dstBucket + '/' + dstKey +
                                        ' due to an error: ' + err
                                );
                        } else {
                                console.log(
                                        'Successfully resized ' + srcBucket + '/' + srcKey +
                                        ' and uploaded to ' + dstBucket + '/' + dstKey
                                );
```

```
                        }
                        context.done();
                }
        );
};
```

Your directory will now look like:

```
index.js
/node_modules/gm
/node_modules/async
```

Select the file and directory and compress into a single zip file (don't compress the folder, compress the files/folder inside the folder).



Go to the Lambda console

Click Create Lambda Function

Click Skip

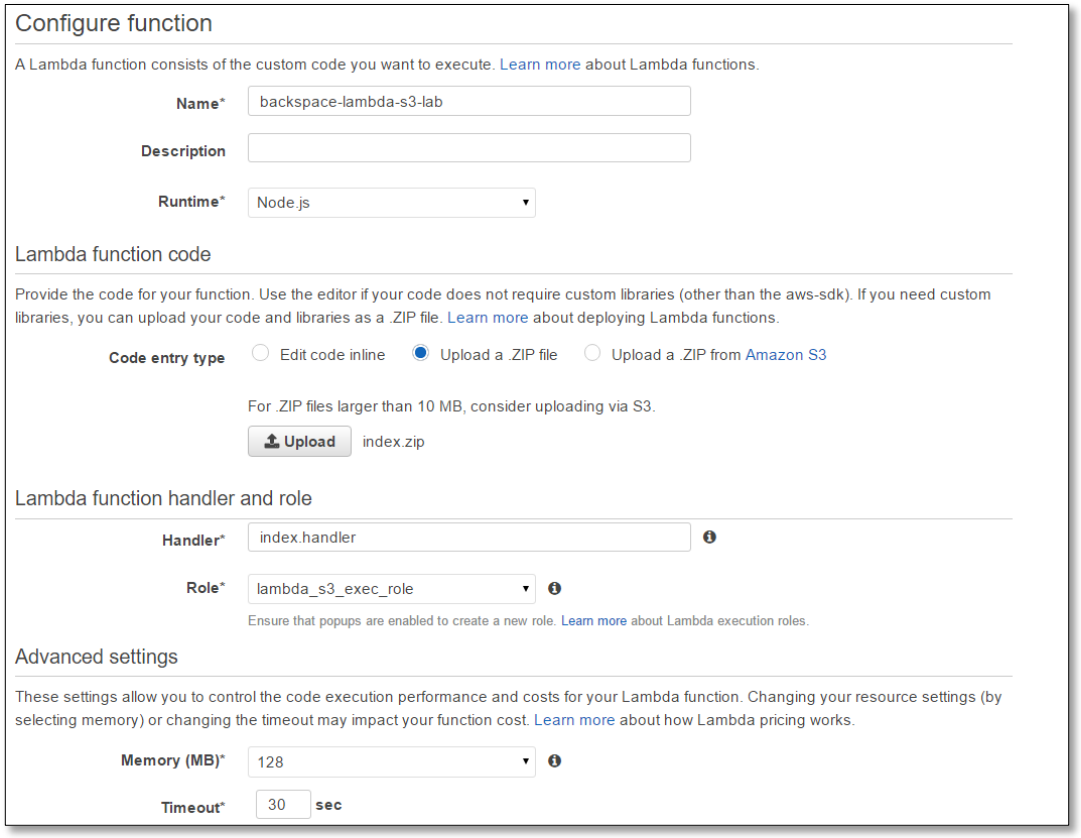
Give it name backspace-lambda-s3-lab

Select Upload a zip file

Upload the zip file

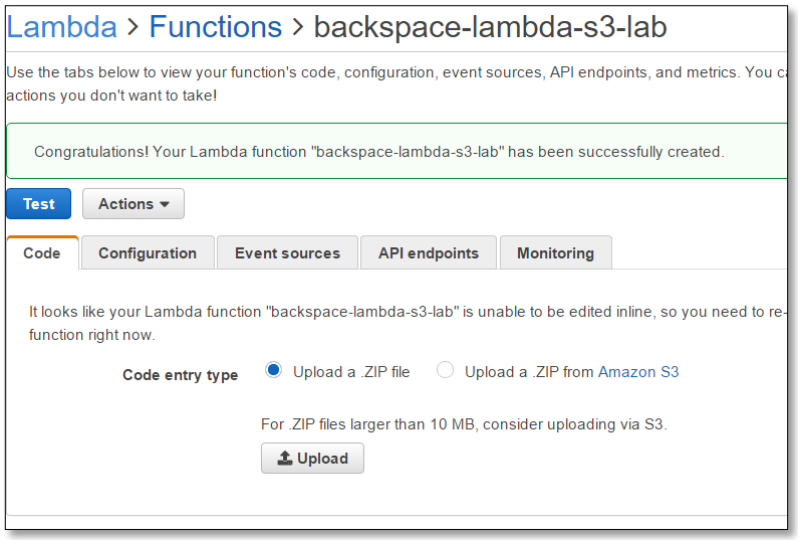
Select the lambda S3 execution role.

Set timeout to 30 seconds.



Click Next

Click Create Function



Click Test

Use the following test object. Change the image filename and bucket name.

```
{
   "Records":[
      {
         "eventVersion":"2.0",
         "eventSource":"aws:s3",
         "awsRegion":"us-east-1",
         "eventTime":"1970-01-01T00:00:00.000Z",
         "eventName":"ObjectCreated:Put",
         "userIdentity":{
            "principalId":"AIDAJDPLRKLG7UEXAMPLE"
         },
         "requestParameters":{
            "sourceIPAddress":"127.0.0.1"
         },
         "responseElements":{
            "x-amz-request-id":"C3D13FE58DE4C810",
            "x-amz-id-2":"FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWerMUE5JgHvANOjpD"
         },
         "s3":{
            "s3SchemaVersion":"1.0",
            "configurationId":"testConfigRule",
            "bucket":{
               "name":"YOUR_BUCKET",
               "ownerIdentity":{
                  "principalId":"A3NL1KOZZKExample"
               },
               "arn":"arn:aws:s3:::YOUR_BUCKET"
            },
            "object":{
               "key":"YOUR_IMAGE.jpg",
               "size":1024,
               "eTag":"d41d8cd98f00b204e9800998ecf8427e",
               "versionId":"096fKKXTRTtl3on89fVO.nfljtsv6qko"
            }
         }
      }
   ]
}
```

In the Log Output you will see the image has been resized and saved in the second bucket.

Now go to the two buckets and see the original and resized images

Now we will automate the process using S3 events.

Go to the source bucket Properties and click Events

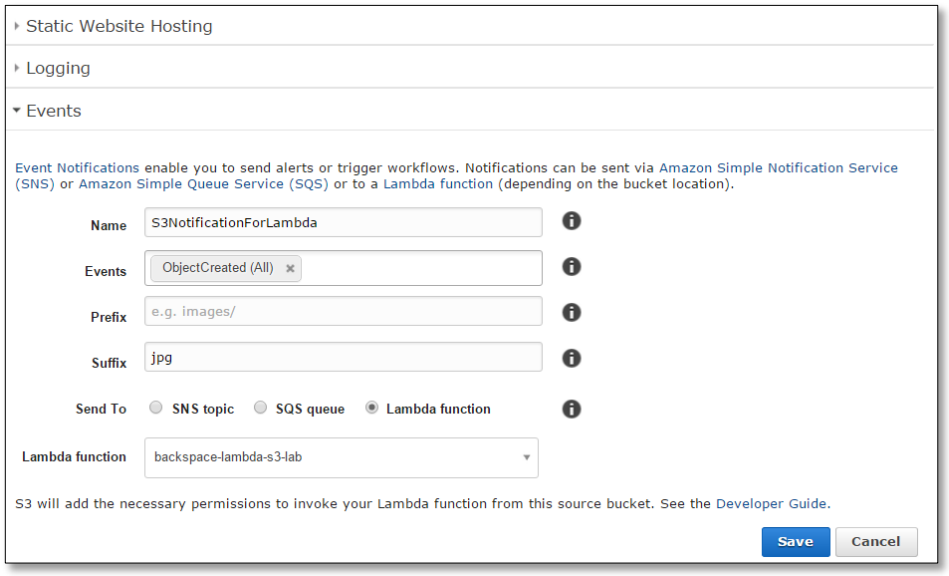
Give the event a name

Select Object Created All

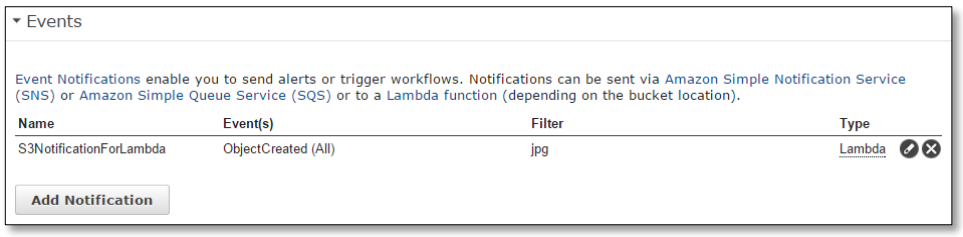
Limit objects with Suffix jpg

Send to Lambda function

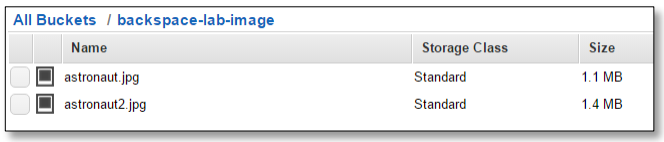
Select Lambda function



Click Save



Now upload another jpg image to the bucket.



After about 30 seconds the new resized image will be created in the other bucket.