



lab title

Programming AWS Lambda V1.00



Course title

AWS Certified Developer Associate



Table of Contents

Contents

Table of Contents.....	1
About the Lab	1
Creating and Testing an AWS Lambda Function	1
Creating an AWS Lambda enabled Browser Application	1
Processing Images using AWS Lambda Function with S3 Events	1

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 latest 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');
};
```

Configure function

A Lambda function consists of the custom code you want to execute. [Learn more](#) about Lambda functions.

Name* BackSpace-Lambda-Lab

Description

Runtime* Node.js

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.

Code entry type ☒ Edit code inline ☐ Upload a .ZIP file ☐ Upload a .ZIP from Amazon S3

```
1 console.log('Loading function');
2
3 exports.handler = function(event, context) {
4   console.log('value1 =', event.key1);
5   console.log('value2 =', event.key2);
6   console.log('value3 =', event.key3);
7   context.succeed(event.key1); // Echo back the first key value
8   // context.fail('Something went wrong');
9 };
```

Select Create new role – Basic execution role

Lambda function handler and role

Handler* index.handler

Role* lambda_basic_execution

Ensure that popups are enabled to create a new role. [Learn more](#) about Lambda execution roles.

Click Next

Review

Please review your Lambda function details. You can go back to edit changes for each section. When you are ready, click **Create function** to complete the setup process.

Lambda function

Edit

Name BackSpace-Lambda-Lab

Description

Runtime NodeJS

Handler index.handler

Role lambda_basic_execution

Memory (MB) 128

Timeout 3

Click Create Function

Lambda > Functions > BackSpace-Lambda-Lab

Use the tabs below to view your function's code, configuration, event sources, API endpoints, and metrics. You can also make any actions you don't want to take!

Test

Actions

Code

Configuration

Event sources

API endpoints

Monitoring

Code entry type ☒ Edit code inline ☐ Upload a .ZIP file ☐ Upload a .ZIP from Amazon S3

```
1 console.log('Loading function');
2
3 exports.handler = function(event, context) {
4     console.log('value1 =', event.key1);
5     console.log('value2 =', event.key2);
6     console.log('value3 =', event.key3);
7     context.succeed(event.key1); // Echo back the first key value
8     // context.fail('Something went wrong');
9 };
```

Click Test

Use the Hello World sample event.

Edit it to:

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

Input sample event

Use the editor below to enter a sample event to test your function with (please remember that this will actually execute the code!).

Sample event template

Hello World

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

Click Submit

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

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: 40f722b0-4e7b-11e5-83ee-5528bf80d9c0
2015-08-29T18:24:58.325Z      40f722b0-4e7b-11e5-83ee-5528bf80d9c0    value1 = This is value1
2015-08-29T18:24:58.325Z      40f722b0-4e7b-11e5-83ee-5528bf80d9c0    value2 = This is value2
2015-08-29T18:24:58.325Z      40f722b0-4e7b-11e5-83ee-5528bf80d9c0    value3 = This is value3
END RequestId: 40f722b0-4e7b-11e5-83ee-5528bf80d9c0
REPORT RequestId: 40f722b0-4e7b-11e5-83ee-5528bf80d9c0  Duration: 5.06 ms    Billed Duration: 100 ms    Memory Size: 128 MB    Max Memory Used: 27 MB
```

▶ 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}"
        }
      }
    }
  ]
}
```

```

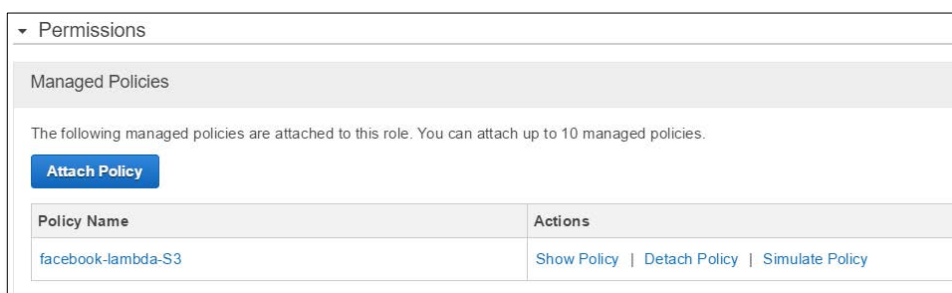
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "lambda:Invoke*",
      "lambda:List*",
      "lambda:Get*"
    ],
    "Resource": "YOUR_LAMBDA_ARN"
  }
]
}

```

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.');
```

```

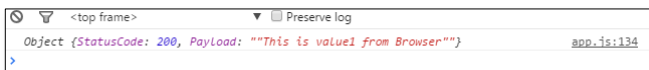
  }
});
}
```

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.

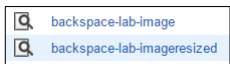
```
Object {Code: Object, Configuration: Object} 800.15:15
  Code: Object
  Configuration: Object
    CodeSize: 308
    Description: ""
    FunctionArn: "arn:aws:lambda:us-east-1:802694931986:function:BackSpace-Lambda-Lab"
    FunctionName: "BackSpace-Lambda-Lab"
    Handler: "index.handler"
    LastModified: "2015-08-29T18:14:56.683+0000"
    MemorySize: 128
    Role: "arn:aws:iam::802694931986:role/lambda_basic_execution"
    Runtime: "nodejs"
    Timeout: 3
  __proto__: Object
  __proto__: Object
```

▶ 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(err) {
        // ...
    });
};

```

```

        },
        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();
  }
};

```

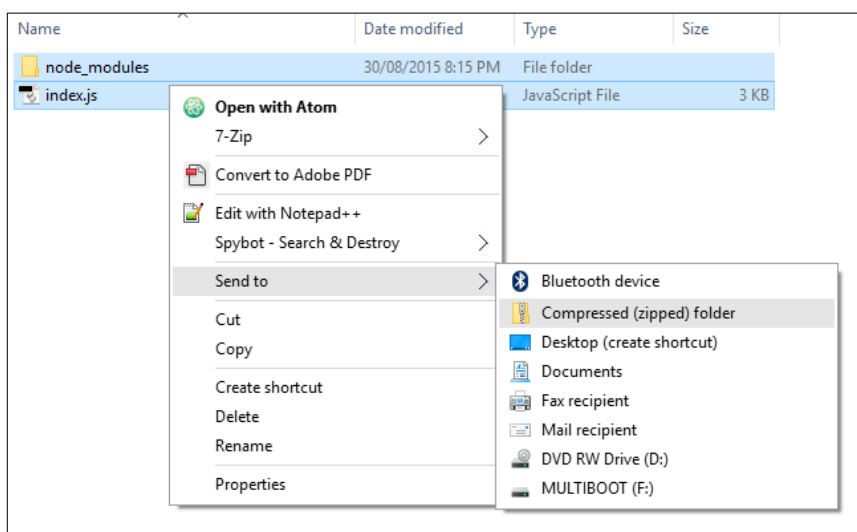
Your directory will now look like:

```

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.

Configure function

A Lambda function consists of the custom code you want to execute. [Learn more](#) about Lambda functions.

Name*

Description

Runtime*

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.

Code entry type ☐ Edit code inline ☒ Upload a .ZIP file ☐ Upload a .ZIP from [Amazon S3](#)

For .ZIP files larger than 10 MB, consider uploading via S3.

index.zip

Lambda function handler and role

Handler* ⓘ

Role* ⓘ

Ensure that popups are enabled to create a new role. [Learn more](#) about Lambda execution roles.

Advanced settings

These settings allow you to control the code execution performance and costs for your Lambda function. Changing your resource settings (by selecting memory) or changing the timeout may impact your function cost. [Learn more](#) about how Lambda pricing works.

Memory (MB)* ⓘ

Timeout* sec

Click Next

Click Create Function

Lambda > Functions > backspace-lambda-s3-lab

Use the tabs below to view your function's code, configuration, event sources, API endpoints, and metrics. You can also perform actions you don't want to take!

Congratulations! Your Lambda function "backspace-lambda-s3-lab" has been successfully created.

Code

It looks like your Lambda function "backspace-lambda-s3-lab" is unable to be edited inline, so you need to re-upload your function right now.

Code entry type ☒ Upload a .ZIP file ☐ Upload a .ZIP from [Amazon S3](#)

For .ZIP files larger than 10 MB, consider uploading via S3.

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": "FMMyUVURIY8/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.

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.

```

bucket:
  { name: 'backspace-lab-image',
    ownerIdentity: { principalId: 'A3NL1KOZZKExample' },
    arn: 'arn:aws:s3:::backspace-lab-image' },
object:
  { key: 'astronaut.jpg',
    size: 1024,
    eTag: 'd41d8cd98f00b204e9800998ecf8427e',
    versionId: '096fKKXTRtl3on89fVO.nfljtsv6qko' } } } }
2015-08-30T11:52:47.687Z      9832c001-4f0d-11e5-8481-e9efa3a46891      Successfully resized backspace-lab-image/astronaut.jpg and uploaded to backspace-lab-imag
eresized/resized-astronaut.jpg
END RequestId: 9832c001-4f0d-11e5-8481-e9efa3a46891
REPORT RequestId: 9832c001-4f0d-11e5-8481-e9efa3a46891  Duration: 15473.67 ms   Billed Duration: 15500 ms   Memory Size: 128 MB   Max Memory Used: 128 MB

```

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

All Buckets / backspace-lab-image				
	Name	Storage Class	Size	Last Modified
	astronaut.jpg	Standard	1.1 MB	Sun Aug 30 16:38:52

All Buckets / backspace-lab-imageresized				
	Name	Storage Class	Size	Last Modified
	resized-astronaut.jpg	Standard	3.4 KB	Sun Aug 30 21:52

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

Static Website Hosting

Logging

Events

Event Notifications enable you to send alerts or trigger workflows. Notifications can be sent via Amazon Simple Notification Service (SNS) or Amazon Simple Queue Service (SQS) or to a Lambda function (depending on the bucket location).

Name: S3NotificationForLambda ⓘ

Events: ObjectCreated (All) x ⓘ

Prefix: e.g. images/ ⓘ

Suffix: jpg ⓘ

Send To: ☐ SNS topic ☐ SQS queue ☒ Lambda function ⓘ

Lambda function: backspace-lambda-s3-lab ▼

S3 will add the necessary permissions to invoke your Lambda function from this source bucket. See the [Developer Guide](#).

[Save](#) [Cancel](#)

Click Save

Events

Event Notifications enable you to send alerts or trigger workflows. Notifications can be sent via Amazon Simple Notification Service (SNS) or Amazon Simple Queue Service (SQS) or to a Lambda function (depending on the bucket location).

Name	Event(s)	Filter	Type
S3NotificationForLambda	ObjectCreated (All)	jpg	Lambda ⓘ x

[Add Notification](#)

Now upload another jpg image to the bucket.

All Buckets / backspace-lab-image

	Name	Storage Class	Size
<input type="checkbox"/>	astronaut.jpg	Standard	1.1 MB
<input type="checkbox"/>	astronaut2.jpg	Standard	1.4 MB

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

All Buckets / backspace-lab-imageresized

	Name	Storage Class	Size
<input type="checkbox"/>	resized-astronaut.jpg	Standard	3.4 KB
<input type="checkbox"/>	resized-astronaut2.jpg	Standard	5.6 KB