Tutorial 4 (Part 2): AWS Lambda and API Gateway

Objective

In this tutorial, you will create a simple API using Amazon API Gateway.

Introduction

AWS API Gateway is a complete platform for creating, managing and running APIs on the AWS platform. An API typically consists of a collection of resources, which are manipulated via a small set of methods also known as verbs or operations (Read more about resource-oriented design and API maturity modelling for more information on API designing).

In this tutorial, you create a basic API with one resource (`DynamoDBManager`) and define one method (`POST`) on it. The method is backed by a Lambda function (`LambdaFunctionOverHttps`). That is, when you call the API through an HTTPS endpoint, Amazon API Gateway invokes the Lambdafunction.

The `POST` method on the `DynamoDBManager` resource supports the following DynamoDB operations:

- Create, update, and delete an item.
- Read an item.
- Scan an item.
- Other operations (echo, ping), not related to DynamoDB, that you can use for testing.

The request payload you send in the `POST` request identifies the DynamoDB operation and provides necessary data. For example:

- The following is a sample request payload for a DynamoDB create item operation.

```
{
    "operation": "create",
    "tableName": "lambda-apigateway",
    "payload": {
        "Item": {
            "id": "1",
            "name": "Bob"
        }
    }
}
```

- The following is a sample request payload for a DynamoDB read item operation.

```
{
    "operation": "read",
    "tableName": "lambda-apigateway",
    "payload": {
        "Key": {
            "id": "1"
        }
    }
}
```

- The following is a sample request payload for an `echo` operation. You send an HTTP POST request to the endpoint, using the following data in the request body.

```
{
  "operation": "echo",
  "payload": {
    "somekey1": "somevalue1",
    "somekey2": "somevalue2"
  }
}
```

Note: API Gateway offers advanced capabilities, such as

- **Pass through the entire request** – A Lambda function can receive the entire HTTP request (instead of just the request body) and set the HTTP response (instead of just theresponse body) using the `AWS_PROXY` integration type.
- **Catch-all methods** – Map all methods of an API resource to a single Lambda functionwith a single mapping, using the `ANY` catch-all method.
- **Catch-all resources** – Map all sub-paths of a resource to a Lambda function withoutany additional configuration using the new path parameter (`{proxy+}`).

To learn more about these API Gateway features, see Configure proxy integration for a proxy resource.

Prerequisites

This tutorial assumes that you have some knowledge of basic Lambda operations and

the Lambda console. If you haven't already, follow the instructions in Getting started with AWS Lambda to create your first Lambda function.

To follow the procedures in this guide, you will need a command line terminal or shell to run commands. Commands are shown in listings preceded by a prompt symbol ($) and the name of the current directory, when appropriate.

```
~/lambda-project$ this is a command
this is output
```

For long commands, an escape character (\) is used to split a command over multiple lines.

On Linux and macOS, use your preferred shell and package manager. On Windows 10, you can install the Windows Subsystem for Linux to get a Windows-integrated version of Ubuntu and Bash.

Create an execution role (**ONLY** if you are using a personal AWS Account)

Create the execution role that gives your function permission to access AWS resources.

To create an execution role

1. Open the roles page in the IAM console.
2. Choose **Create role**.
3. Create a role with the following properties.
   - **Trusted entity** – Lambda
   - **Role name** – `LabRole`
   - **Permissions** – Custom policy with permission to DynamoDB and CloudWatch Logs.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1428341300017",
      "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
      ],
      "Effect": "Allow",
      "Resource": "*"
```

```
    },
    {
      "Sid": "",
      "Resource": "*",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Effect": "Allow"
    }
  ]
}
```

The custom policy has the permissions that the function needs to write data to DynamoDB and upload logs. Note the Amazon Resource Name (ARN) of the role for later use.

Create the function

The following example code receives an API Gateway event input and processes the messages that it contains. For illustration, the code writes some of the incoming event data to CloudWatch Logs.

Note: For sample code in other languages, see Sample function code.

Example index.js

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

var AWS = require('aws-sdk');
var dynamo = new AWS.DynamoDB.DocumentClient();

/**
 * Provide an event that contains the following keys:
 *
 *   - operation: one of the operations in the switch statement below
 *   - tableName: required for operations that interact with DynamoDB
 *   - payload: a parameter to pass to the operation being performed
 */
```

```
exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));

    var operation = event.operation;

    if (event.tableName) {
        event.payload.TableName = event.tableName;
    }

    switch (operation) {
        case 'create':
            dynamo.put(event.payload, callback);
            break;
        case 'read':
            dynamo.get(event.payload, callback);
            break;
        case 'update':
            dynamo.update(event.payload, callback);
            break;
        case 'delete':
            dynamo.delete(event.payload, callback);
            break;
        case 'list':
            dynamo.scan(event.payload, callback);
            break;
        case 'echo':
            callback(null, "Success");
            break;
        case 'ping':
            callback(null, "pong");
            break;
        default:
            callback('Unknown operation: ${operation}');
    }
};
```

To create the function

1. Copy the sample code into a file named `index.js`.
2. Create a deployment package.

```
$ zip function.zip index.js
```

3. Create a Lambda function with the `create-function` command.

```
$ aws lambda create-function --function-name LambdaFunctionOverHttps \
--zip-file fileb://function.zip --handler index.handler --runtime
nodejs12.x \
--role arn:aws:iam::<your-account-id>:role/LabRole
```

Test the Lambda function

Invoke the function manually using the sample event data. We recommend that you invoke the function using the console because the console UI provides a user-friendly interface for reviewing the execution results, including the execution summary, logs written by your code, and the results returned by the function (because the console always performs synchronous execution—invokes the Lambda function using the `RequestResponse` invocation type).

To test the Lambda function

1. Copy the following JSON into a file and save it as `input.txt`.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

2. Execute the following `invoke` command:

```
$ aws lambda  invoke --function-name LambdaFunctionOverHttps \
--payload fileb://input.txt outputfile.txt
```

Create an API using Amazon API Gateway

In this step, you associate your Lambda function with a method in the API that you created using Amazon API Gateway and test the end-to-end experience. That is, when an HTTP request is sent to an API method, Amazon API Gateway invokes your Lambda function.

First, you create an API (`DynamoDBOperations`) using Amazon API Gateway with one resource (`DynamoDBManager`) and one method (`POST`). You associate the `POST` method with your Lambda function. Then, you test the end-to-end experience.

Create the API

Run the following `create-rest-api` command to create the `DynamoDBOperations` API for this tutorial.

```
$ aws apigateway create-rest-api --name DynamoDBOperations
The output will look like:
{
    "id": "bs8fqo6bp0",
    "name": "DynamoDBOperations",
    "createdDate": 1539803980,
    "apiKeySource": "HEADER",
    "endpointConfiguration": {
        "types": [
            "EDGE"
        ]
    }
}
```

Save the API ID for use in further commands. You also need the ID of the API root resource. To get the ID, run the `get-resources` command.

```
$ API=bs8fqo6bp0
$ aws apigateway get-resources --rest-api-id $API
The output will look like:
{
    "items": [
        {
            "path": "/",
            "id": "e8kitthgdb"
        }
    ]
}
```

At this time, you only have the root resource, but you add more resources in the next step.

Create a resource in the API

Run the following `create-resource` command to create a resource (`DynamoDBManager`) in the API that you created in the preceding section.

```
$ aws apigateway create-resource --rest-api-id $API  --path-part
DynamoDBManager \
--parent-id  e8kitthgdb
The output will look like:
{
    "path": "/DynamoDBManager",
    "pathPart": "DynamoDBManager",
    "id": "iuig5w",
    "parentId": "e8kitthgdb"
}
```

Note the ID in the response. This is the ID of the `DynamoDBManager` resource that you created.

Create POST method on the resource

Run the following `put-method` command to create a `POST` method on the `DynamoDBManager` resource in your API.

```
$ RESOURCE=iuig5w
$ aws apigateway put-method --rest-api-id $API --resource-id $RESOURCE \
--http-method POST --authorization-type NONE
The output will look like:
{
    "apiKeyRequired": false,
    "httpMethod": "POST",
    "authorizationType": "NONE"
}
```

We specify `NONE` for the `--authorization-type` parameter, which means that unauthenticated requests for this method are supported. This is fine for testing but in production you should use either the key-based or role-base authentication.

Set the Lambda function as the destination for the POST method

Run the following command to set the Lambda function as the integration point for the `POST` method. This is the method Amazon API Gateway invokes when you make an HTTP request for the `POST` method endpoint. This command and others use ARNs that include your account ID and region. Save these to variables (you can find your account ID in the role ARN that you used to create the function).

```
$ REGION=us-east-2
$ ACCOUNT=123456789012
$ aws apigateway put-integration --rest-api-id $API --resource-id
$RESOURCE \
--http-method POST --type AWS --integration-http-method POST \
--uri arn:aws:apigateway:$REGION:lambda:path/2015-03-
31/functions/arn:aws:lambda:$REGION:$ACCOUNT:function:LambdaFunctionOverHt
tps/invocations
The output will look like:
{
    "type": "AWS",
    "httpMethod": "POST",
    "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-
31/functions/arn:aws:lambda:us-east-
2:123456789012:function:LambdaFunctionOverHttps/invocations",
    "passthroughBehavior": "WHEN_NO_MATCH",
    "timeoutInMillis": 29000,
    "cacheNamespace": "iuig5w",
    "cacheKeyParameters": []
}
```

`--integration-http-method` is the method that API Gateway uses to communicate with AWS Lambda. `--uri` is unique identifier for the endpoint to which Amazon API Gateway can send request.

Set `content-type` of the `POST` method response and integration response to JSON as follows:

- Run the following command to set the `POST` method response to JSON. This is the response type that your API method returns.

```
$ aws apigateway put-method-response --rest-api-id $API \
--resource-id $RESOURCE --http-method POST \
--status-code 200 --response-models application/json=Empty
```

```
The output will look like:
{
    "statusCode": "200",
    "responseModels": {
        "application/json": "Empty"
    }
}
```

- Run the following command to set the `POST` method integration response to JSON. This is the response type that Lambda function returns.

```
$ aws apigateway put-integration-response --rest-api-id $API \
--resource-id $RESOURCE --http-method POST \
--status-code 200 --response-templates application/json=""
The output will look like:
{
    "statusCode": "200",
    "responseTemplates": {
        "application/json": null
    }
}
```

Deploy the API

In this step, you deploy the API that you created to a stage called `prod`.

```
$ aws apigateway create-deployment --rest-api-id $API --stage-name prod
The output will look like:
{
    "id": "20vgsz",
    "createdDate": 1539820012
}
```

Grant invoke permission to the API

Now that you have an API created using Amazon API Gateway and you've deployed it, you can test. First, you need to add permissions so that Amazon API Gateway can invoke your Lambda function when you send HTTP request to the `POST` method.

To do this, you need to add a permission to the permissions policy associated with your Lambda function. Run the following `add-permission` AWS Lambda command to grant

the Amazon API Gateway service principal (`apigateway.amazonaws.com`) permissions to invoke your Lambda function (`LambdaFunctionOverHttps`).

```
$ aws lambda add-permission --function-name LambdaFunctionOverHttps \
--statement-id apigateway-test-2 --action lambda:InvokeFunction \
--principal apigateway.amazonaws.com \
--source-arn "arn:aws:execute-
api:$REGION:$ACCOUNT:$API/*/POST/DynamoDBManager"
The output will look like:
{
    "Statement": "{\"Sid\":\"apigateway-test-
2\",\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"apigateway.amazonaws
.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda
:us-east-
2:123456789012:function:LambdaFunctionOverHttps\",\"Condition\":{\"ArnLike
\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-east-
2:123456789012:mnh1yprki7/*/POST/DynamoDBManager\"}}}"
}
```

You must grant this permission to enable testing (if you go to the Amazon API Gateway and choose **Test** to test the API method, you need this permission). Note the `--source-arn` specifies a wildcard character (*) as the stage value (indicates testing only). This allows you to test without deploying the API.

Note: If your function and API are in different regions, the region identifier in the source ARN must match the region of the function, not the region of the API.

Now, run the same command again, but this time you grant to your deployed API permissions to invoke the Lambda function.

```
$ aws lambda add-permission --function-name LambdaFunctionOverHttps \
--statement-id apigateway-prod-2 --action lambda:InvokeFunction \
--principal apigateway.amazonaws.com \
--source-arn "arn:aws:execute-
api:$REGION:$ACCOUNT:$API/prod/POST/DynamoDBManager"
The output will look like:
{
```

```
    "Statement": "{\"Sid\":\"apigateway-prod-
2\",\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"apigateway.amazonaws
.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda
:us-east-
2:123456789012:function:LambdaFunctionOverHttps\",\"Condition\":{\"ArnLike
\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-east-
2:123456789012:mnh1yprki7/prod/POST/DynamoDBManager\"}}}"
}
```

You grant this permission so that your deployed API has permissions to invoke the Lambda function. Note that the `--source-arn` specifies a `prod` which is the stage name we used when deploying the API.

Create an AWS DynamoDB table

Create the DynamoDB table that the Lambda function uses.

To create a DynamoDB table

1. Open the AWS DynamoDB Console.
2. Choose **Create table**.
3. Create a table with the following settings.
   - **Table name** – `lambda-apigateway`
   - **Primary key** – `id` (string)
4. Choose **Create**.

Trigger the function with an HTTP request

In this step, you are ready to send an HTTP request to the `POST` method endpoint. You can use either Curl or a method (`test-invoke-method`) provided by Amazon API Gateway.

You can use Amazon API Gateway CLI commands to send an HTTP `POST` request to the resource (`DynamoDBManager`) endpoint. Because you deployed your Amazon API Gateway, you can use Curl to invoke the methods for the same operation.

The Lambda function supports using the `create` operation to create an item in your DynamoDB table. To request this operation, use the following JSON:

Example create-item.json

```
{
    "operation": "create",
    "tableName": "lambda-apigateway",
    "payload": {
        "Item": {
            "id": "1234ABCD",
            "number": 5
        }
    }
}
```

Save the test input to a file named `create-item.json`. Run the `test-invoke-method` Amazon API Gateway command to send an HTTP `POST` method request to the resource (`DynamoDBManager`) endpoint.

```
$ aws apigateway test-invoke-method --rest-api-id $API \
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \
--body file://create-item.json
```

Or, you can use the following Curl command:

```
$ curl -X POST -d "{\"operation\":\"create\",\"tableName\":\"lambda-
apigateway\",\"payload\":{\"Item\":{\"id\":\"1\",\"name\":\"Bob\"}}}"
https://$API.execute-api.$REGION.amazonaws.com/prod/DynamoDBManager
```

To send request for the `echo` operation that your Lambda function supports, you can use the following request payload:

Example echo.json

```
{
  "operation": "echo",
  "payload": {
    "somekey1": "somevalue1",
    "somekey2": "somevalue2"
  }
}
```

Save the test input to a file named `echo.json`. Run the `test-invoke-method` Amazon API Gateway CLI command to send an HTTP `POST` method request to the resource (`DynamoDBManager`) endpoint using the preceding JSON in the request body.

```
$ aws apigateway test-invoke-method --rest-api-id $API \
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \
--body file://echo.json
```

Or, you can use the following Curl command:

```
$ curl -X POST -d
"{\"operation\":\"echo\",\"payload\":{\"somekey1\":\"somevalue1\",\"someke
y2\":\"somevalue2\"}}" https://$API.execute-
api.$REGION.amazonaws.com/prod/DynamoDBManager
```