TERBINE CONNECTION VIA AWS LAMBDA / JAVASCRIPT EXAMPLE

Last Revision: May 3, 2020
Version: 1.0.0

Contents © Terbine 2020. All rights reserved.

TABLE OF CONTENTS

Contents

Overview 2
Prerequisites 2
Serverless Programming....2
AWS Lambda 2
Terbine AWS Lambda....3
Creating the Function 3
Entering Function Code....4
Create Trigger....5
Add Cloudwatch Logging 6
Next Steps....9
Revisions 10

 Contents © Terbine 2020. All rights reserved.

OVERVIEW

This example guide provides an introduction to using AWS Lambda for accessing the Terbine API. This can be used in conjunction with the Terbine API Overview and Terbine JavaScript SDK Guide documents.

PREREQUISITES

The reader is expected to have basic understanding of JavaScript, AWS and use of the AWS Console. While there is no expectation for previous knowledge of the AWS Lambda, this example doc is not meant to be a comprehensive introduction to Serverless programming or AWS Lambda. It is a guide to show a basic introduction to calling the Terbine API via an AWS Lambda function. For further information on Serverless architecture and AWS Lambda see this guide.

SERVERLESS PROGRAMMING

Serverless computing an architecture that provides backend services on a per use basis. It allows developers to write and deploy code without needing to be concerned about the underlying infrastructure such as servers. Naturally physical servers are still used, but developers do not need to worry about configuring or maintaining them

Using Serverless computing, developers purchase on a pay as you go basis and only pay when the service is in use. For many tasks that have large fluctuations in usage dependent on user demand or are run only during certain times of the day and lie dormant for the majority of the time this can result in considerable savings.

There are other benefits of using Serverless including high availability (built-in), simplified scaling on demand, simplified code as it runs within the Serverless container and a more agile deployment process.

AWS LAMBDA

AWS Lambda is Amazon's (AWS) serverless offering that is built on the concept of functions. This is a familiar programming model for developers using the Microservice model. It runs based on a trigger, so is reactive and event driven, and can be invoked for example by a HTTP Request, a Notification via the Simple Notification Service (SNS), events coming in via Kinesis, or an object added to a S3 bucket. As explained above, the developer only pays when the Lambda function is being executed.

It has a simple programming model that allows quick turnaround on development tasks and supports a variety of languages for implementation (JavaScript, Golang (Go), Python, Java etc.).

 Contents © Terbine 2020. All rights reserved.

Each Lambda function has these components:

- Code you want to execute.
- Associated configuration how the code is executed.
- Event sources that trigger is triggered when some event happens and executes the code.

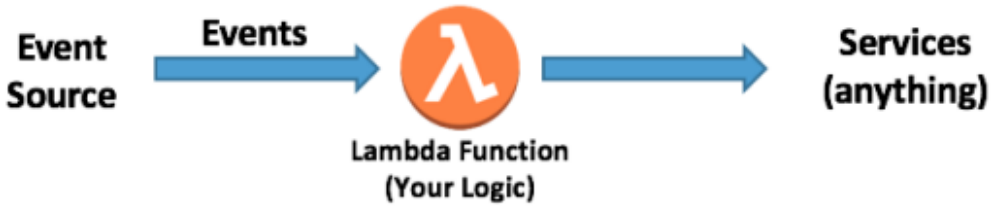

Figure 1 - Simplified View of Lambda Flow

Once configured, an event source for your function the code is invoked when the event occurs. Your code can execute any business logic, execute external web services (e.g. Terbine API), integrate with other AWS services (e.g. DynamoDB), or virtually anything else your application requires. All of the same capabilities and software architecture design strategies that are normal for whatever is your language of choice will apply when using Lambda

Key to executing a Lambda function is a special construct called the **handler**. The handler is a specific code method (e.g. in Java) or function (JavaScript, Python) that you've created and included in your code package that is invoked by the AWS container.

Once the handler is successfully invoked within the Lambda function, the runtime belongs to the code you've written. The handler can call other methods and functions within the files or classes that were uploaded. You can import 3rd party libraries are accessible or even install and execute executable binaries.

TERBINE AWS LAMBDA

This section is a step by step introduction to creating an AWS Lambda function for accessing the Terbine API. It describes creating a JavaScript based function with logging and shows the authentication and retrieval of data from Terbine.

CREATING THE FUNCTION

Login to the AWS Console and go to the AWS Lambda dashboard by searching for Lambda. Note that often a specific IAM Role will be created specifically for creating and managing your Lambda functions, but we will not do this within this example.

 Contents © Terbine 2020. All rights reserved.

Figure 2- AWS Lambda Console Dashboard

From here you can see existing Lambda functions as well as view metrics related to your functions.

Press on Create Function button, select and Select "Author from scratch and enter the name of the function, for example **terbineApiTest**. Select Node.js 12.x as the runtime option if not already selected. At the bottom is where you can select an **Execution role** if you created one earlier. Once done, press the **Create Function** button at the bottom to actually create the Lambda function.

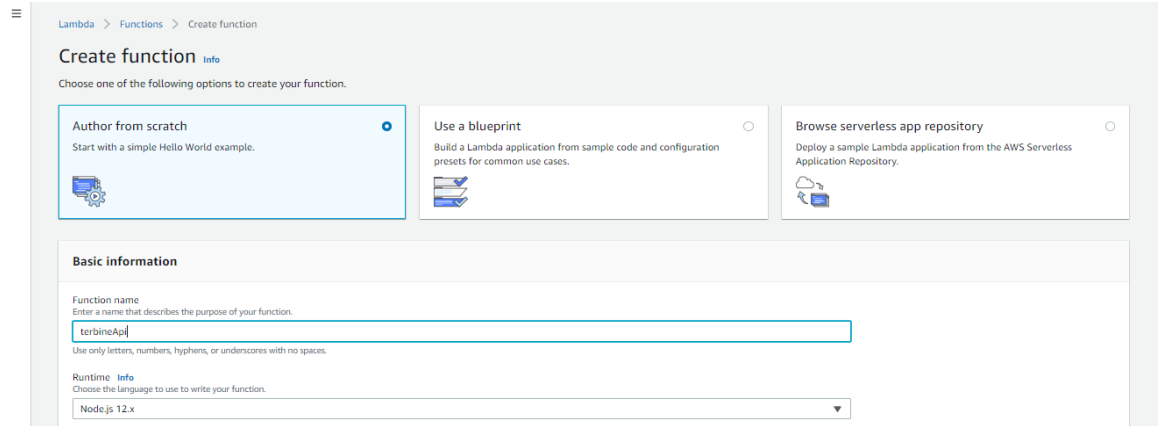

Figure 3- AWS Lambda Create Function Page

ENTERING FUNCTION CODE

Once you press Create Function you will be taken to the Function Details Page and you can enter code and configure your new Lambda Function. Before entering code to execute the Terbine API, let's setup the function with some basic code to test.

In the **Function Code** section, you can enter code inline. Enter the following code.

```
exports.handler = async(event) => {
    const response = {
          statusCode: 200,
          body: 'Current time is ' + new Date()
    };
    return response;
```

 Contents © Terbine 2020. All rights reserved.

```
};
```

This function will simply return the current date and time. After entering the text, press Save. Now that we have the code, you need a way to execute it. This is done by configuring a trigger.

CREATE TRIGGER

Scroll back to the top of the Function detail page and select + **Add Trigger**. From the list of types, select **API Gateway** and select **Create a new API** and keep all the defaults. If prompted select **HTTP API** and for Security option choose **Open**.

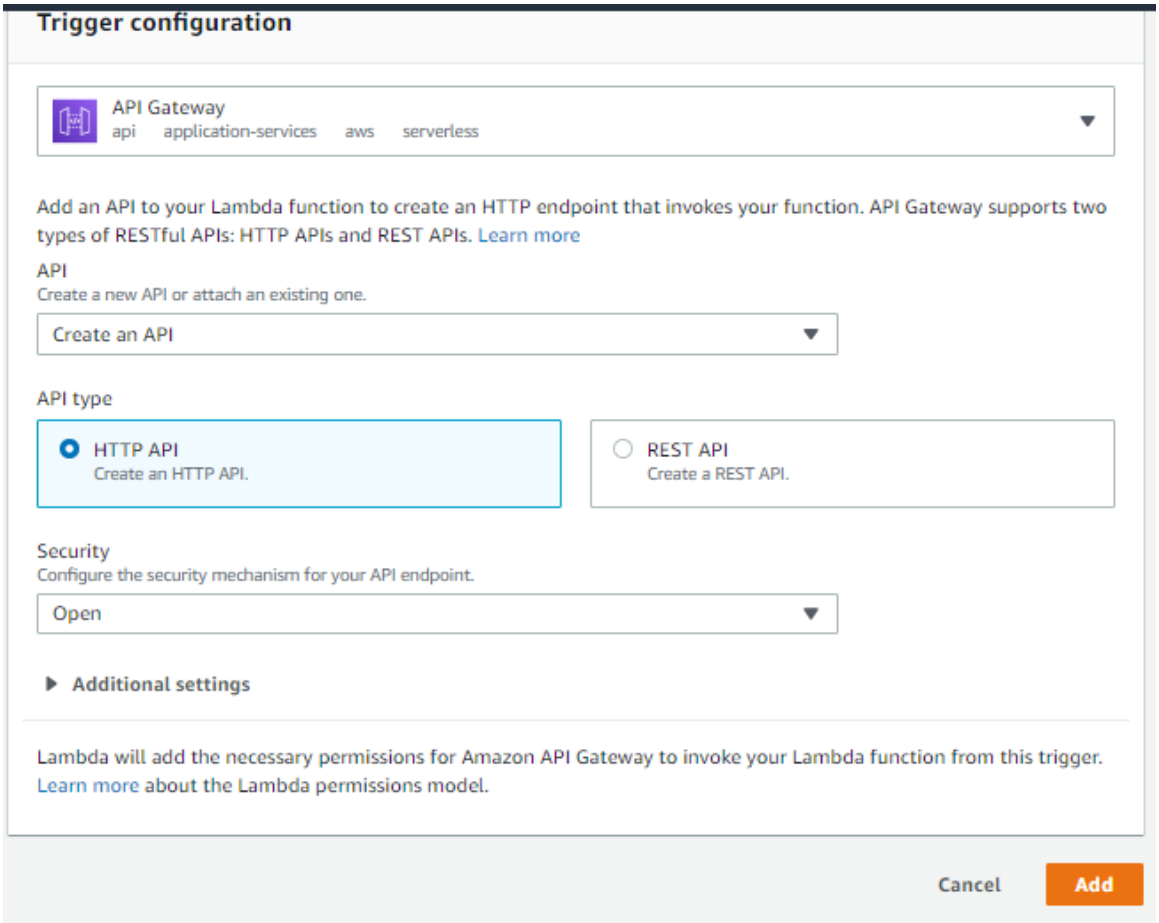

Figure 4 - Create Trigger Page

Once done, press the Add button and the trigger using **API Gateway** will be created.

You will then be presented with an API Endpoint, copy this into Postman or equivalent and you can execute your Function.

You should see the following after executing:

 Contents © Terbine 2020. All rights reserved.

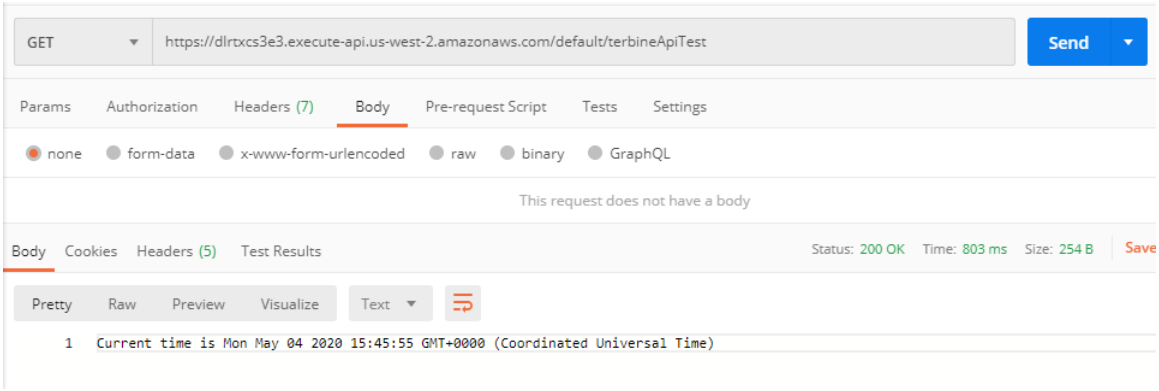

Figure 5 - Execute Sample Lambda

ADD CLOUDWATCH LOGGING

One way to gain insight into your function is by using **Amazon Cloudwatch**. To do this go to the Cloudwatch service within the Console. Select the **Log Groups** and since we executed the function one time we will see our function.

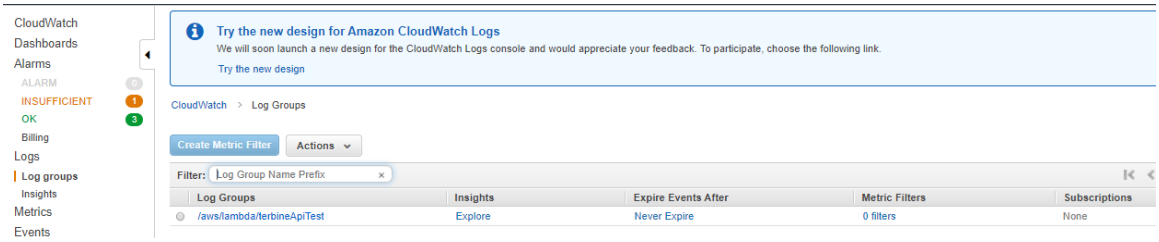

Figure 6 - Log Group, Select Function

Selecting the function, you will be taken to logging which among other items shows the pricing information for the execution. Any logging, such as any console.log statements will be shown here. This is very useful during development.

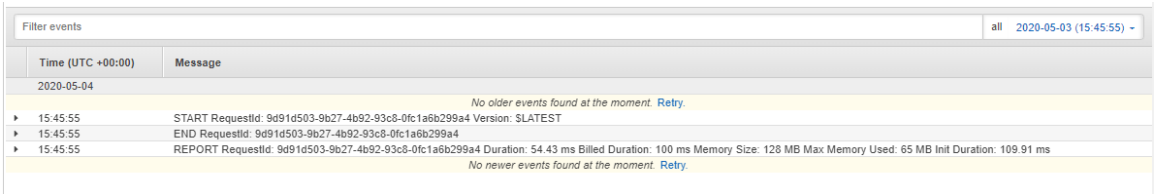

Figure 7 - Amazon Cloudwatch Details

Accessing Terbine

Now that the function is working with a trivial example, we are able to trigger it via our configured API Gateway and we can add code to access the Terbine API. Replace the existing code with the following:

 Contents © Terbine 2020. All rights reserved.

const https = require('https')

```
exports.handler = async(event) => {
    var token;
        var orgID;

    const loginResponse = await new Promise((resolve, reject) => {

        const data = JSON.stringify({
          userName: '<username>',
          password: '<password>'
        })

        const options = {
          hostname: 'terbine.io',
          port: 8443,
          path: '/api/auth/login',
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            'Content-Length': data.length
          }
        }

        const req = https.request(options, (res) => {

        let dataString = '';

        res.on('data', chunk => {
            dataString += chunk;
        }).on('end', () => {
              var json = JSON.parse(dataString, null, 4);
              token = json.token;
              orgID = json.orgid;
              resolve({
                  statusCode: 200,
                  body: JSON.stringify(json)
              });
        });
      })

      req.on('error', (e) => {
          console.error('Error during login: ' + e);
          reject({
              statusCode: 500,
              body: 'Something went wrong Login!',
          });
        });

      req.write(data)
      req.end();
    });

    const response = await new Promise((resolve, reject) => {
```

 Contents © Terbine 2020. All rights reserved.

```
        const data = JSON.stringify({
          text: 'Air Temperature'
        })

        const options = {
          hostname: 'terbine.io',
          port: 8443,
          path: '/api/search/v2/metadata?pageNum=1&pageSize=20',
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            'Content-Length': data.length,
            'Authorization': 'bearer ' + token
          }
        }

        const req = https.request(options, (res) => {
            let dataString = '';
            res.on('data', chunk => {
                dataString += chunk;
            }).on('end', () => {
                  var json = JSON.parse(dataString, null, 4);
                  resolve({
                      statusCode: 200,
                      body: JSON.stringify(json),
                      headers: {
                        'x-page-num': res.headers['x-page-num'],
                        'x-page-size': res.headers['x-page-size'],
                        'x-tracking-id': res.headers['x-tracking-id'],
                        'x-processed-num': res.headers['x-processed-
num'],
                        'x-total-count': res.headers['x-total-count']

                      }
                  });
            });
        })

        req.on('error', (e) => {
              console.error('Error during search: ' + e);
              reject({
                  statusCode: 500,
                  body: 'Something went wrong search!'
              });
          });

        req.write(data)
        req.end();
    });

    return response;
};
```

Within the code, you will need to replace the **<username>** and **<password>** values with your own Terbine credentials.

 Contents © Terbine 2020. All rights reserved.

Once this is done, press **Save** and you can execute the code.

You will receive a response for the search that was executed for the term ***Air Temperature*** as was defined in the code.

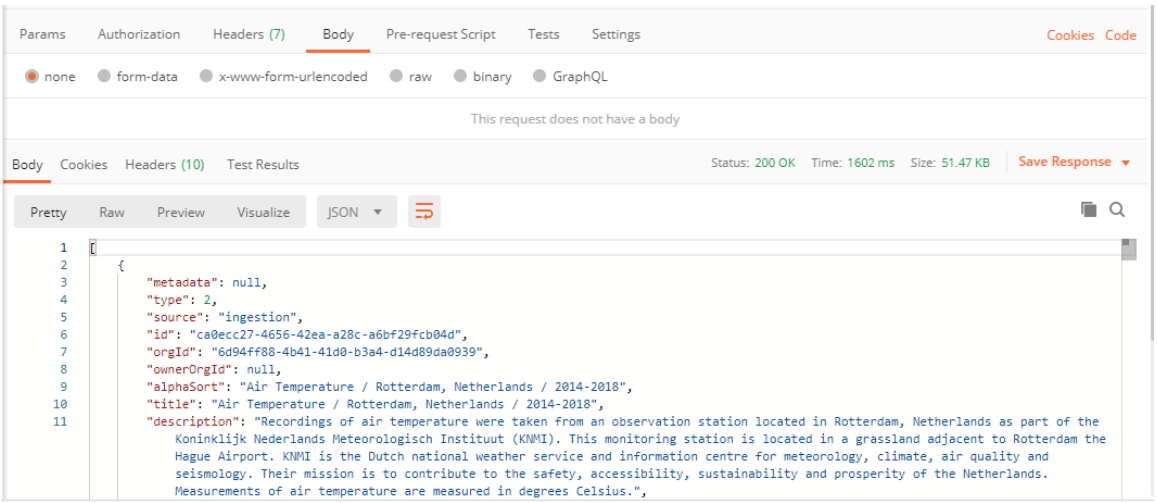

Figure 8 - API Response For Terbine Search

Note also, within the code the custom Terbine header values are passed through to the API Gateway response when processing the response from the Terbine API:

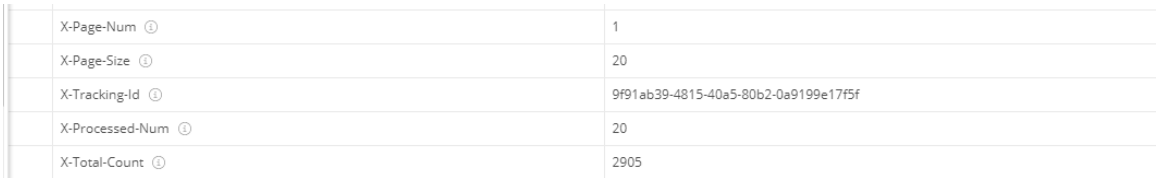

Figure 9 - Terbine Custom Header Values

In this case, you see there were 20 results returns (as defined in the request) and there are a total of 2905 Terbine feeds that fit the search criteria.

NEXT STEPS

The Terbine API Overview and Terbine JavaScript SDK Guide documents can provide further information. For instance using the above search results shown in the response, a feed could be added to the workspace using the Feed GUID value, locked and the data accessed. From there, the Lambda function could be used to store the data in S3 for further processing.

 Contents © Terbine 2020. All rights reserved.

REVISIONS

Version	Date	Name	Description
1.0.0	2020/05/03	Brian Enochson	Initial Release

 Contents © Terbine 2020. All rights reserved.