Part 2
Node.js on AWS

What's in this guide?

This short guide provides an intro to the following:

Step 1 How to install and set up Node.js on your AWS ec2 instance

Step 2 How to create a simple HTTP server on AWS using Node.js

Step 3 Connecting to DynamoDB from JavaScript code on AWS

Prerequisites

You are expected have already done the following:

Step 0.1 Signed into your AWS academy Learner Lab (or AWS free tier) account

Step 0.2 Instantiated a t2.micro ec2 instance with Ubuntu 20.04 (allowing all TCP traffic in the network settings)

Step 0.3 Signed into the ec2 instance's console via PuTTY or Terminal

Step 0.4 Installed a file transfer client such as FileZilla and connected it to your ec2 instance to transfer files to/from your local machine.

If you have not yet completed these steps, guides to them have been provided separately.

Step 0.5 Go through the guide *Basics of JavaScript*, which has been provided separately.

Step 1 How to install and set up Node.js on your AWS ec2 instance

The AWS documentation lists a simple 4-step process to install Node.js on AWS. With an Ubuntu 22.04 image installed on your instance, this should be straight forward. To execute these 4 steps, it is assumed that you are logged into your AWS instance via PuTTY or terminal.

Follow the steps specified on this AWS documentation page.

Step 2 How to create a simple HTTP server on AWS using Node.js

We have previously been able to communicate information between client and server using bare TCP sockets. That method is good while we are learning how communication takes place on the internet under the hood. However, it slows down development as it requires management of sockets manually. It is better to use HTTP (or HTTPS): the application layer protocol used by the Web.

Sarim Baig
May 2022

Recall that HTTP is a stateless request-response protocol, where each request and response message has a specific format.

Instead of manually constructing these messages and sending them through TCP sockets, we will use Node.js: a JavaScript runtime which allows us to write an web server to respond to HTTP requests. The runtime will manage the sockets for us under the hood. Moreover, it will provide us a programming interface to specify the various elements of HTTP responses and requests.

In Step 1, we installed Node.js on AWS. Now let us write a very simple Node.js based webserver and run it on the ec2 instance.

Here is the web server code. It should be stored in a .js type file. A good name for the file could be mywebserver.js

mywebserver.js

```
var http = require('http');
var server = http.createServer(function(req, res){
   res.writeHead(200, {'Content-Type':'text/plain'})
   res.end('You have reached your AWS EC2 web server...');
});

console.log('Server is running on port 3000');
server.listen(3000,'0.0.0.0');
```

On your local machine, the listen function should use 127.0.0.1, which is the localhost. You can then run the server by clicking run from the menu on Visual Studio Code.

The guide 'Basics of JavaScript' describes how to download, install and use VS code to write Node.js code on your local computer.

What's the code in *myserbserver.js* doing?

-- require('http'): includes the http module to the code

-- **http.createServer** creates a server object. It accepts a JavaScript function as its parameter. This function is set as one of the properties of the server object and is fired every time an HTTP request arrives at the server. The **req** parameter of the function is a reference to an object containing the incoming HTTP request. The **res** parameter is a reference to an object containing the outgoing HTTP response.

-- functions **res.writeHead** and **res.end** are properties of the response object. They are used to specify values of the various fields of the HTTP response message. Recall that an HTTP response header has a status code field, and 200 means OK. The HTTP response message also has a Content-Type field, set to 'text/plain' in this case. This means the response being sent to the client's browser comprises plain text.

-- **res.end** simply sets the Entity body field of the HTTP response message.

-- **server.listen** simply instructs the server to start listening for incoming HTTP requests on the given port number and IP address.

Now you may take the following steps to execute this code on ASW and test it from your machine using the browser:

- Using FileZilla, transfer *mywebserver.js* to the ubuntu folder of the ec2 instance.
- Using PuTTY, run the following command on the ec2 command line:
  $node mywebserver.js
- On your local machine, open the browser to the following address:
  your-ec2-instance-public-IP:3000
  For example, the public IP of my ec2 instance is 54.174.192.231, so I will enter the following in the address bar:
  54.174.192.231:3000

  The browser window should now show the following:
  You have reached your AWS EC2 web server…

Side Note: Node.js v Python
The client server programming we have previously done was all in Python. We could continue to code our webserver in Python as well, however that would have robbed us of the opportunity to explore another language and framework. Having said that, both choices have their own merits. Here is a nice comparison between Node.js and python.

Step 3 Connecting to DynamoDB from JavaScript code on the AWS

Note: the use of DynamoDB as described in the following documentation requires some basic use of JavaScript. So, it is recommended that you go through the elementary JavaScript guide, which has been provided separately, before you proceed below.

There is a detailed guide on how to use DynamoDB on AWS provided separately (you might be familiar with it from Information Processing). That guide contains coding examples in Python. Naturally, the same DynamoDB functionalities can also be used from JavaScript. Here is the AWS documentation on how to use DynamoDB from JavaScript code.

Before you use DynamoDB as explained above, install the AWS JavaScript SDK as explained in documentation here.