Jamie Gentry
For **EngagedMD, Inc.**
December 29, 2021

Phone Number Formatter Case Study

For this project, I have created a program which formats a U.S. phone number, and I show how it can be deployed as an API on AWS Lambda.

The entire project can be found on GitHub.

I describe the parts which went into the implementation, from the ground up. A simple architecture diagram can be found below.

The formatter itself

The main logic of the formatter is the **formatPhoneNumber** method, written in JavaScript on Node.js. I chose Node.js because it is simple to deploy to Amazon Lambda (and also, selfishly, just to get more experience with it myself). A JVM-based language (such as Java or Kotlin) would likely be just as effective, so the main consideration here would be to consider the preferences and the conventions of the organization who would build and maintain the software.

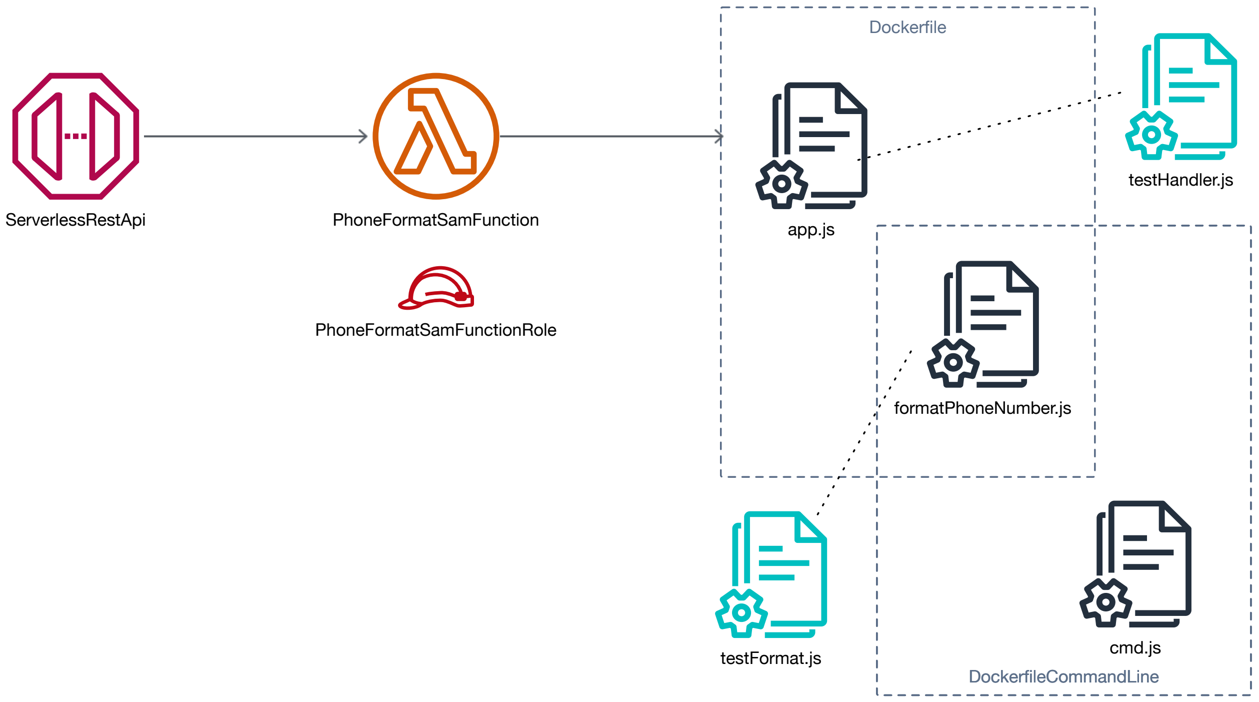
The logic of the formatter is quite simple:

```
const formatPhoneNumber = (input) => {
  if (input.length != 10)
    throw new RangeError(`Bad format ${input}`)
  else
    return input.slice(0, 3) + '-' +
      input.slice(3, 6) + '-' +
      input.slice(6)
}
```

The formatter is set to fail if the string passed in is not ten characters.

➡ *In real life, with more knowledge of requirements, we would probably want to do better or different validation.* For example, if the function is to be used to format a phone number *as it is being typed* by an end user, then we would probably want the function to accept partial phone numbers. Also, the function might want to flag clearly incorrect input (such as non-numeric characters), and it might want to succeed even if the input already contains dashes.

There are simple unit tests for the format method in **test-format.js**. These tests can be run via *npm test* (after first performing an *npm install*).



The overall architecture

The standalone Docker container

The formatter code can be packaged up in a Docker image using the **DockerfileCommandLine** Docker file, using the following command (run from within the *formatter* directory);

```
docker build -f DockerfileCommandLine -t format .
```

As requested, the Docker image accepts the string to be formatted from an environment variable named *PHONE*. The **cmd.js** module provides the entry point for the Docker container; it grabs the value of *PHONE* from the environment, calls the formatter function, and writes the results to standard output. The docker image can be run via a command like:

```
docker run -e PHONE="1234567890" format
```

Deployment to AWS

The phone number formatter can be deployed to AWS as an API, through AWS Lambda. I used the AWS Serverless Application Model (SAM) to set this up.[1]

The template used for this application specifies the lambda function as well as a gateway endpoint API. Among other things, SAM produces the **ServerlessRestApi** gateway, the **PhoneFormatSamFunction** lambda function, and the **PhoneFormatSamFunctionRole** IAM role from the template.

- ➡ Some of the naming here is admittedly awkward, and would want to be cleaned up and standardized at some point.

I decided to upload the application as a Docker image—in order to keep to the requirements for how the standalone Docker image is to be invoked (*e.g.* using an environment variable for the input), I needed to create a separate **Dockerfile** for this SAM image. Included in this image is a "lambda handler" Node.JS method in **app.js**, used to translate the inputs expected by the AWS Lambda interface to the inputs expected by the (platform-agnostic) *formatPhoneNumber* method. (There is also a unit test, **testHandler**, which exercises the logic in app.js.)

SAM makes it significantly easier to create and deploy a basic Lambda function, and it provides methods to allow you to deploy the function locally for testing purposes.

The actual deployed end point to the function may be found at *https://8cgfyuipvd.execute-api.us-east-2.amazonaws.com/Prod/format*. Here is a valid invocation:

```
   https://8cgfyuipvd.execute-api.us-east-2.amazonaws.com/Prod/
format?PHONE=6175551234
```

… and here is an invalid invocation (using an invalid phone number):

```
   https://8cgfyuipvd.execute-api.us-east-2.amazonaws.com/Prod/
format?PHONE=12345
```

- ➡ At the moment, this gateway is open to everyone in the world—depending on how it would be used in real life, we would want to lock this down more.

[1] I decided to go beyond the requirements here and perform a real deployment. Though I have used AWS extensively in my past work, I have never actually set up a function in Lambda, and I wanted to try it out.

Performance considerations

The function and deployment as implemented is quite simple. In the real world, we might make different design choices based on what we predict about the future direction of this function.

If we expected the function to remain this simple, then it might make sense to move it into the browser (assuming that it is being used to prettify text input by the user). However, if we expected the function to take on more responsibility—such as formatting international phone numbers or more sophisticated validation, then it would likely want to remain on the server.

We would want to consider scaling requirements—if we expected the use of the function to fluctuate, then we would want to set up schedules for scaling or to set up autoscaling. And of course, we would want to keep an eye on AWS metrics to make sure that (for example) the function is not being invoked more than we think it should be.