Rick Farina
rickrfarina@gmail.com
linkedin.com/in/rickrfarina
(210) 896-1837

AWS Demo

April 20, 2018

OVERVIEW

This application is intended to show how modernization can be achieved by gradually implementing new applications in the Cloud vs. a large scale Lift and Shift approach that can take years and cost several millions of dollars with no guarantee of a successful outcome.

It also shows how costs can be mitigated by taking advantage of AWS Managed Services such as EC2, API GW, Lambda, and DynamoDB. Additionally, it eliminates the need to purchase additional disk, cpu, memory and the corresponding increase in maintenance costs.

GOALS

1. Other than an EC2 instance to house a nodejs website, the application will be developed using Serverless (fully managed) AWS Services.
2. The application's intent is simply to paint a picture of what can be accomplished and highlight the benefits of implementation in the cloud vs. developing on-prem.
3. Emphasize how an organization looking to modernize can utilize small applications to become familiar with Cloud technologies in addition to beginning the onboarding process with new developers that may not be available within its current pool of resources.

APPLICATION SCENARIO

This application is based on a Logistics company that handles and delivers freight to its customers, much like FedEx, UPS, and Amazon. For demo purposes we will assume that the organization has a fleet of delivery trucks, and that each driver is equipped with a

mobile device that allows for route navigation, and reporting delivery completion back to the system of record.

However, the company is running into customer service issues as the demand for real-time ETA information is growing. The company recognizes that they need to provide this service but are concerned about the added costs associated with storing and querying the data.

After reviewing the options, the company decides to implement this application by taking advantage of AWS Cloud Services.

APPLICATION SPECIFICS

Because we don't have an actual fleet of trucks and drivers that we can use to track data in real-time, we did the next best thing. A web based application was written that reports geographic location points continuously and records them in DynamoDB. The points were collected by starting the application on a mobile device and driving from point A to point B; with point A representing the origin and point B representing the destination. The data is keyed by OrderId and Transaction Time Stamp.

These records were then used to "simulate" an actual driver making a delivery from origin A to destination B.

The application performs a very simple function. The browser component of the application requests the geographic data points in a loop via a nodejs web application running on an AWS EC2 instance. The nodejs application in turn retrieves the data via an API GW ➔ Lambda ➔ DynamoDB invocation, and returns the data as a json object. The returned object contains the latitude/longitude points, which are used to track the shipment's movements on a map via the use of the Google MAP API.

The application can be run here http://54.200.121.119:5001. Simply select an Order Id and click the Track Shipment button. Note that Order Id "12345" is valid; while Order Id "xxxxx" is invalid and is included to show how errors are generally handled.

PROGRAMMING NOTES

Because this was always intended to be a demo, it was written procedurally with all the Html, CSS, and Javascript in the same index.html file. This was done for ease of

development and quickly getting the application working. With the proper refactoring, the CSS and JavaScript will eventually be split out into separate files and housed in “styles” and “scripts” folders respectively.

Additionally, there is more work to be done on the Google Map to plot out the route, and to adjust the direction of the vehicle as it moves along the route.

It’s also worth noting that the browser application could have directly invoked the API GW without going through the nodejs API. But this would have required AWS credentials to be stored in browser app and would represent a security risk. By allowing the nodejs application to be the broker, the credentials are not required as the nodejs app is running on an EC2 instance that has assumed an Identity Access Management Role (IAM); which provides it with the authority to invoke the API GW, run the Lambda function, access DynamoDB, and write to the CloudWatch logs.

Another security enhancement would be to implement an API GW Authorizer to ensure that only valid clients are allowed access.

With respect to git commits, they will not be allowed if errors are detected by ESLint. This is accomplished via the use of the pre-commit module. Specific .eslintrc.json files have been established for Server (nodejs) and Client (javascript in the browser) for purposes of implementing environment-dependent rules.

With respect to the actual Lambda function, the source code has been included in a folder named “lambda” for review. However, the code was manually written in the Lambda Console. Another enhancement would be to implement the Serverless.com Framework for deployment.

A final note is that this application is intended to be used to demonstrate how an organization can quickly build a Cloud based application with little to no impact to their existing system of record, and begin the process of Digital Transformation using incremental steps rather than using the big bang alternative of Lift and Shift.

End of Document