Michael Griffin

06/07/2021

**Voice Foundry Interview Project**

Hi, the following documentation regards the development project which I was assigned as part of your interview process. To whomever may be reading this, I appreciate the chance to showcase my skills and possibly join your company. I had a lot of fun doing this project, and if the work is anything like the project, I believe I would be a highly engaged and valuable asset for the team.

The project prompt is included below. There was a discrepancy in the instructions where step 3 said to save the best 5 vanity numbers and step 4 says to return the 3 vanity possibilities. I decided to go with the higher number for both, just to be sure.

# Mid-Level Development Project

We expect you to build this out as you would a production project for a client, only on a small scale (tests, error handling, etc). Where you don't have the time to implement something, add comments to your code or documentation on how you would have changed or added to your implementation in the "real world".

### Deliverables

1. Git Repo with all code and documentation

2. **BONUS** - a working Amazon Connect phone number to test in your environment :-)

### Exercise

3. Create a Lambda that converts phone numbers to vanity numbers and save the best 5 resulting vanity numbers and the caller's number in a DynamoDB table. "Best" is defined as you see fit - explain your thoughts.

4. Create an Amazon Connect contact flow that looks at the caller's phone number and says the 3 vanity possibilities that come back from the Lambda function.

5. **BONUS** - Build a deployment package with AWS SAM, AWS CDK, or CloudFormation to allow a user, or assignment reviewer, to deploy your solution into their own AWS Account/Amazon Connect instance. (The contact flow can be imported manually, but include instructions!)

6. **SUPER BONUS -** a web app that displays the vanity numbers from the last 5 callers.

7. Writing and Documentation

1. Record your reasons for implementing the solution the way you did, struggles you faced and problems you overcame.

2. What shortcuts did you take that would be a bad practice in production?

3. What would you have done with more time? We know you have a life. :-)

4. **BONUS** - Please include an architecture diagram.

Show off. This is your chance to demonstrate your ability to learn, Google, and figure it out. Do your best to express your areas of expertise and ability.

**Deliverables**

Main Git repo - https://github.com/mjgriffin1113/vanity-num-generator

Amazon Connect Phone Number (generate vanity number) – (626 - 623 - 3979)

Web app Git repo - https://github.com/mjgriffin1113/vf-call-history-viewer

Web app URL - [http://vf-vanity-num-generator.s3-website-us-east-1.amazonaws.com](http://vf-vanity-num-generator.s3-website-us-east-1.amazonaws.com/)

**Documentation**

When creating and implementing my solution, my first major decision was to spend less time on the function which generates and selects the vanity numbers, and instead spend more effort on making a robust, somewhat secure, “serverless” web application which checked off all the requirements, all the bonus asks, and several new features which I felt would be appreciated by the “client” (and if not, the features could easily be modified after meeting with the client and asking if these extra enhancements are desired or not). As a result, my function to generate the vanity numbers is somewhat trivial. It generates 5 random possible vanity numbers, and then chooses the best of these 5. All 5 of these could be “bad” options though. This is the first place I would target for any future enhancements of this application, because I see a lot of room for improvement.

One such improvement would be to utilize some sort of algorithm (machine-learning??), (or find an existing package which does something similar), to parse the generated vanity number and find the longest English words which are present in each generated string. The function could start by only looking for strings with 7 letter words, and then slowly working down through the 6 and 5 letter words, and so on, until 5 options have been selected. Then the option with the longest word could be chosen. Alternatively, the customer could give us some keywords and we could try to select the vanity number strings which most closely relate to the keywords. There are a lot of possibilities with this function, and since I don’t have a client to refine the idea with, I decided to keep it simple and focus primarily on the functionality of the rest of the application.

Speaking of the application, another enhancement I would make had this been a ‘real’ project, would have probably been to separate each of the resource types into its own project file. That way the creation, management, and deployment of each serverless resource type would be independent of the others. As it currently exists, if the DynamoDB table fails to update/create, then the lambdas will fail to deploy. Decoupling these entities is the best implementation plan moving forward.

Also, if there had been a bit more time, I would have set up an automated deployment pipeline, where any merge/commit to the main branch would kick off an automated build/deploy. I set this up for the Call History web application, but wasn’t able to get to it for the main app.

I tried not to take too many “shortcuts” but there is certainly an endless amount of configuration which could be done, which I mostly ignored as there was no reason to customize anything too much since there was no business use case for it. For example, I selected all the default settings when creating a DynamoDB table. I could have also chosen to create and use custom IAM policies for the Lambda function roles, instead of using default AWS managed ones, which technically give more access to the Lambda functions than is needed to run this application. I tried to make the application relatively secure by limiting access to the DynamoDB table and setting a bucket policy on the deploy bucket in s3, but there is always room for improvement in regards to security, and I’m sure there are additional security features and enhancements of which I am not yet aware. I am aware, however, that having no layer of authorization on my Call History API (exposed lambda function) is not a best practice. However, the only endpoint that is exposed is a GET, so I figured this shortcut wasn’t too detrimental (it does expose customer phone numbers, so that is obviously a security flaw).

A few additional enhancements I would make would be to create multiple levels of environments, rather than only having one. I would then have a separate DynamoDB table for each environment level. I would also have a different Amazon Connect instance for each environment. I would address/acknowledge any vulnerabilities in currently used npm packages (Github tells me there is 1 existing vulnerability with the node-notifier package). I also do not really have true unit tests, and would certainly add them and mock the external requests (DB and API requests) if I had a bit more time. I do have functional integration tests. However, thses are not the best because they write fake data to the only table currently used for the app and they do not clean the data up after testing, and some of the tests rely on the data inserted by the others. It would be best to insert and cleanup the data required for each integration test separately.

As for the actual contact flow, there are several enhancements I would have implemented if I had a bit more time. The first is getting rid of the spaces in the suggested vanity number, so that Joanna doesn’t say the word “space.” I would also enable the Kinesis data streams from the Connect instance and implement a listener (Lambda) to read the stream events for the vanity number topic, and then store this data in a long-term storage/data warehouse such as Snowflake. To make management easier, prompts/messages could be stored in a database and then dynamically referenced in the prompt action block. A Lambda function can look up and correctly set the prompt text variable so that it can be referenced. This is a needed enhancement to reduce management complexity, and a good step towards total adoption of “infrastructure-as-code” processes. Another fun feature I wanted to add, but am not sure it’s possible on the current Connect platform, is to ask the caller to state their name, record it, and then use this recording to greet the customer at the end of the call, and on their subsequent calls to the number. It seems that call recording is only available when an agent is on the line with a customer.

I certainly did face a few challenges while working on this project. Coming in, I had light experience with AWS Cloudformation, and none with AWS SAM specification. While it wasn’t a completely new challenge, I had never used a “sam” command before, so there was a small learning curve as I figured out how to successfully configure, build, and deploy the application into my AWS environment. I also was not familiar with “permissions boundaries” and I got stuck for a minute until I realized that you need to manually update the existing permissions boundary when adding new entities/resources to the application. I re-discovered that fact that nested responses are not allowed from Lambda to Connect, and I definitely battled with IAM for a bit. Not everything was a chore, however. Some challenges I had faced previously in my time writing and deploying Lambda functions to production have been solved over the last year or so. The AWS toolkit and the ability to easily run and test the various resources in Docker eliminated a lot of wasted time spent waiting to deploy minor fixes to AWS just so that they could be tested in a real environment. Local Lambda development was a lot more enjoyable than I remember it being, as I wasn’t having to constantly zip and upload my function just to test and iterate on small changes.

It feels like I could think of enhancements all day, so I will stop here. I will include all the deployment instructions I can think of in the project readme. Hopefully everything works for you as well as it does for me with minimal massaging on your end. Again I want to thank you for all of your time, and reiterate how much I actually enjoyed working on this project. Thanks.