6. Writing and Documentation

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

There is no end for any of the projects in this world, still MS Excel is growing ☺. So scope of the project is important. As the timeline are already defined, I decided to go with the MVP of the project in all aspects like functionalities, user experience etc. I took last seven digits of the caller and implemented the solution as requested. Now if you want to proceed further it will be easy to add other requirements like last 6 digits, last 5 digits, in between any number of digits. As all the logics are available, it will be easy for the team to enhance further.

When comes to the challenges, I was not aware of the concept of vanity. So I understood the concept, learned and started the designing. Another one, initially I was using nltk package in python to fetch the words from the dictionary. It was giving lot of problems while including in lambda and consuming 4 MB of lambda space. After spending half of the day only on this, I replaced this package with another library, English Words.

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

There is enough scope to use orchestrator like Step functions to handle multiple lambdas and let the vanity generator run asynchronously. It will avoid the timeout error, if the search goes beyond 8 seconds.

In the webpage there is no AWS Cognito implemented. Currently considering all are authenticated users.

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

There is a looping option in the current vanity generation. Currently due to due constraint I used the expanded condition. That could be achieved.

Other options of vanity in 6 digits, 5 digits and in-between

Step function

Access key implementation for API calls

Possibilities are directly proportional to time ☺

4. What other considerations would you make before making our toy app into something that would be ready for high volumes of traffic, potential attacks from bad folks, etc.

When we think big, obviously we have to use the advantages of cloud and other architecture rules. As we already use the power of server-less in lambda and Amazon Connect components, I list out the other areas

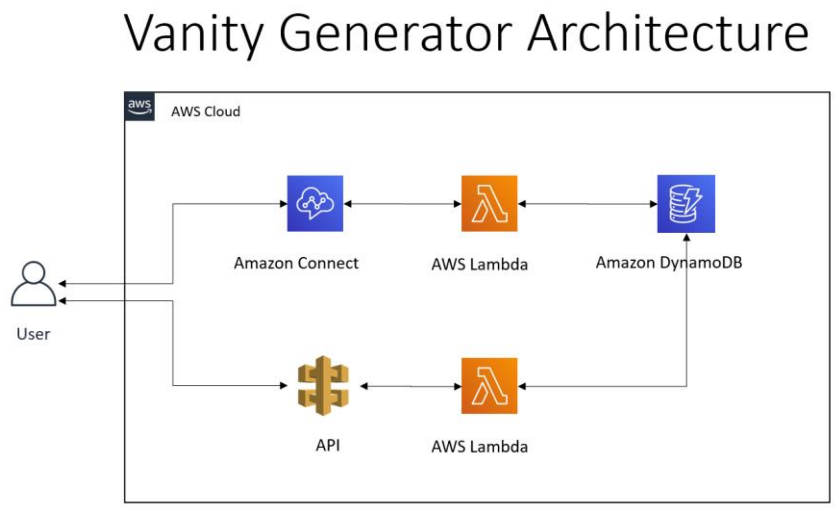
High-availability and Resilience for other than server-less services like webpage and DB. By implementing Application Load Balancer for web application and implementing read replicas we can ensure the high availability

Even though we are implementing ALB, we have to plan for the resiliency properly.

Security of the resources in cloud, ALB will take care of the basic level threats like DDOS. For access, we need to provide only for required roles. Access keys and tokens can be enabled for secured API calls. Encryption will help us to keep the data safe in transit and rest. According to the requirement, we can go for managed or KMS or client managed for the data security

From the application side: To reduce the vanity generation time, instead of finding the vanity for every call, it can be run in offline and the DB may be filled with all the permutation combinations of vanity numbers. At the time of user request, it can check the DB before starting vanity generation. It will reduce the call time and increase the customer satisfaction

5. Please include an architecture diagram.


Vanity Generator Architecture
AWS Cloud
Amazon Connect
AWS Lambda
Amazon DynamoDB
User
API
AWS Lambda