Assignment 3 Report *15324740*

Task 1

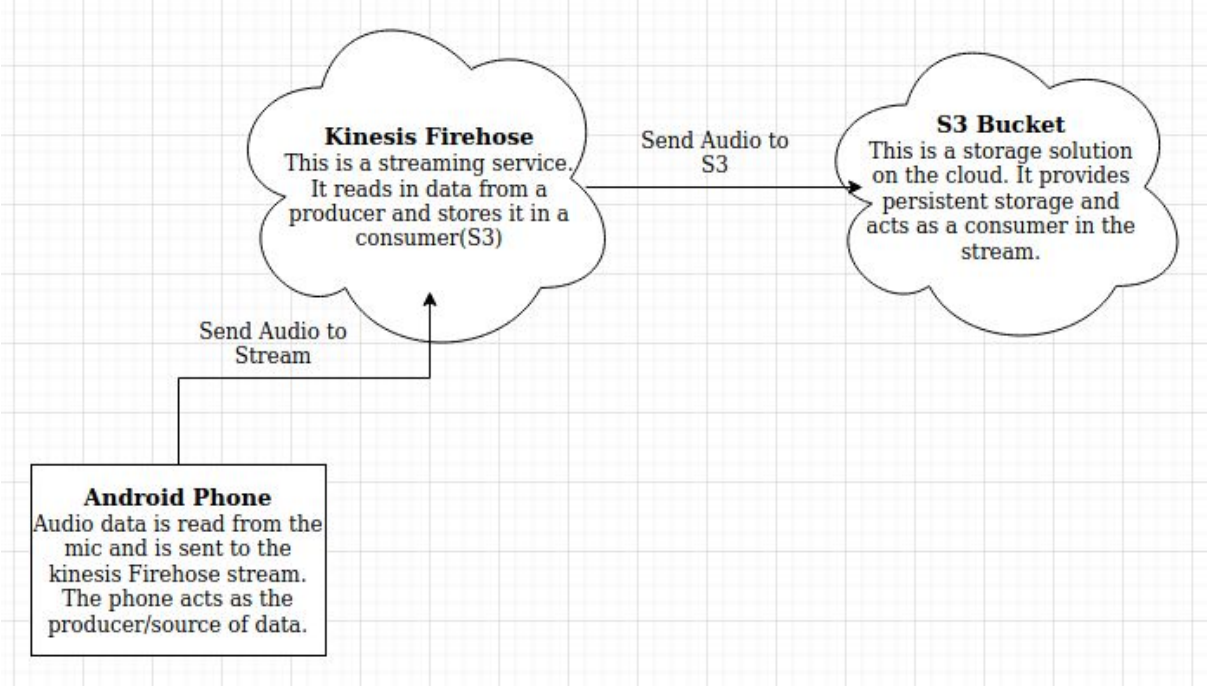

Fig 1.1 This is a technical diagram of the Task 1 system

I added the dependencies for AWS Amplify to my android application. This has recently replaced the role of the AWS Android SDK. The Amplify libraries create the backend in the development environment. This removes the need to set up connections manually in the application code. When configuring AWS Amplify I had to sign into AWS and create an IAM user for amplify. I added the IAM user to a group that had all the permissions granted to work with AWS Kinesis Firehose and S3. I then passed the keys needed to start programming into the command-line interface.

In my cloud integration I set up an AWS Kinesis Firehose stream. The Kinesis service was configured to directly connect to data sources like phones, IoT devices, etc. The data that was streamed to the Kinesis service was piped into an S3 bucket. S3 is a service that provides scalable, replicated (available and redundant) persistent storage. Storage can also be provided by managed database services like AWS RDS. I did not have the need for transactions and fast queries. If the need to process or transform the data before it is stored is felt, I can add a lambda function trigger to the Kinesis Firehose stream. AWS lambda is a serverless solution. It is cheap because you only pay for the number of CPU cycles you use. It may be slow to start because you might invoke a cold lambda server.

The data is generated from my phone's microphone in a separate thread from the main. The data is sent real time to Kinesis Firehose as it is generated. If I was to integrate this app with a lambda function or an RDS database I would send data in bigger batches. Bigger batches would increase the throughput of the system due to the increased continuity of operations in an RDS/lambda instance; but it might introduce latency. Firehose streams the data in real-time to S3 where it is stored in a default S3 bucket.

The benefit of using AWS kinesis is that it will automatically scale to thousands or millions of users if need be. This would be hard to implement and manage myself.

Task 2

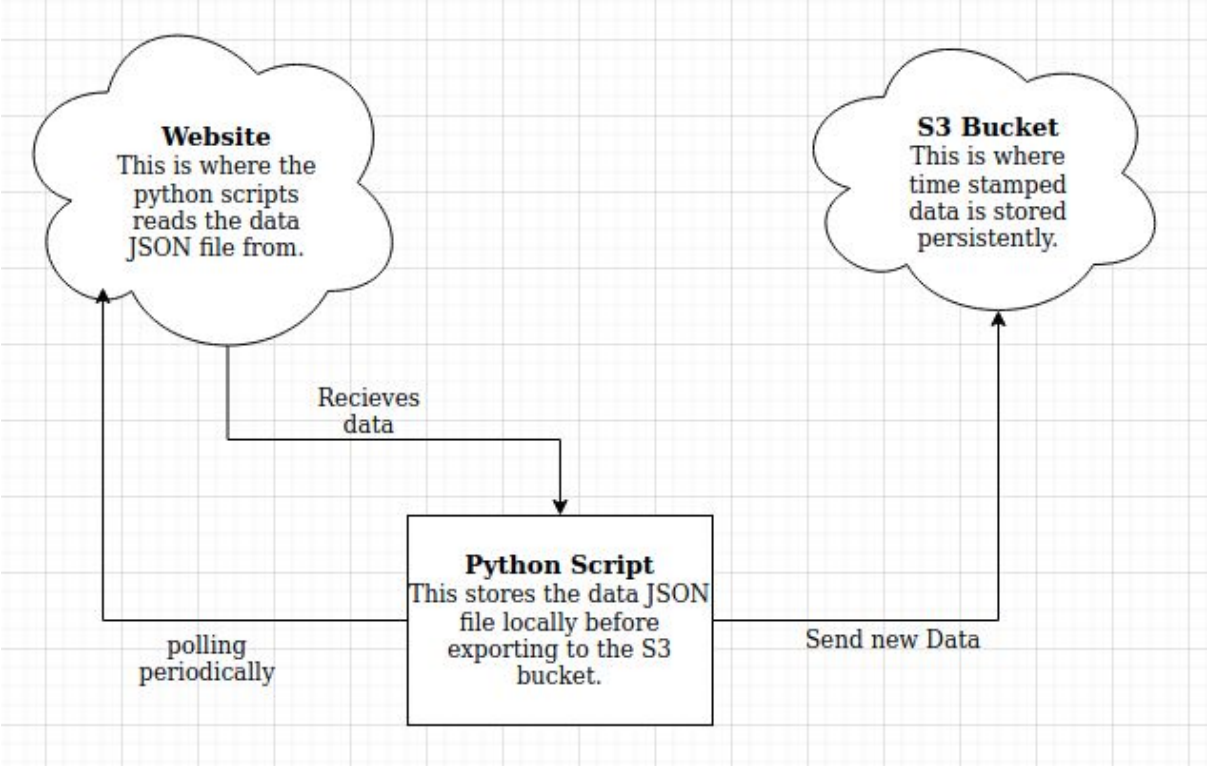

Fig 2.1 This is the technical diagram for the Task 2 implementation

I created a python script that accessed a data source on parking space availability in Amsterdam. The data was provided in the form of a JSON file. The file was being updated in real time. The update frequency was roughly 10-20 seconds. I downloaded the file using the “urllib” library. The data was written into a file and it was hashed.

The hash provided a means for checking if the information had been updated quickly. In the case that the website was updated, the python script would be able to write the new data to the data file and it could then send the file to an S3 bucket.

The data was sent to an S3 bucket using the boto3 package available through the PIP repository. The data.json file was uploaded to an S3 object with the timestamp of the data as the name. This allows historical stored data to be easily found.

The python script constantly polls the website for the data. Therefore the data is collected and stored automatically in real time.

Data Source Link: http://opd.it-t.nl/data/amsterdam/ParkingLocation.json