Prototype Big Data Archive in a Public Cloud

Group 56: Pathfinder of Big Data

Zhi Jiang, Isaac T Chan, Zhaoheng Wang

CS 463: Senior Capstone Spring 2017

Oregon State University

Abstract

This document presents a review, summary, explanation of our group's progress so far. It includes the purpose of this project and details of where we currently are, as well as how we deal with each portion of the project.

Contents

1 INTRODUCTION

1.1 Purpose

OSU campuses generate data constantly from multiples sources, including computer labs, wireless usage, student devices, and many others. This quantity of data, also known as big data, can effectively represent student behaviors for information technology. For example, analysis can be run to determine common student behaviors in order to allocate OSU resources to more often used utilities. Currently, the data is very difficult to manage because it is collected from multiple sources and is impossible to analyze. The data is neither stored in the same formats nor in the same locations, meaning it is inaccessible and useful information is unable to be extracted. The purpose of this project is to unify the data onto the consistent cloud platform of Amazon Web Services, which additionally provides utilities to manage and analyze.

1.2 Current Position

Currently, the functional part of the project is complete. We have a working database implementation in the Amazon Web Services public cloud, with appropriate storage, loading, and restricted access for sensitive data. By leveraging additional services in the AWS environment, we are able to provide rudimentary analysis, reporting, and visualization of data. From this position, we are able to present at Expo on the entire project, from beginning to end, including potential outcomes if the project were adopted by OSU. In the original project description, there were three deliverables listed. We have completed two of them: the database implementation and analysis/reporting/visualization. The third, a cost model, has yet to be complete. In order to provide an accurate cost model, we require an estimate for data amounts and are currently waiting for the estimate. This deliverable won't affect the project demonstration at Expo and is soley for the benefit of our client to decide on whether OSU will adopt our prototype.

2 DYNAMODB - ZHAOHENG WANG

2.1 Data structure organization

In general, the project will start with parsing sample data from s3. After that, the sample data will be processed and store into the DynamoDB. Finally, we will do the analysis in the QuickSight based on the sample data. Before loading data into DynamoDB, the table and the data structure should be organized. Therefore, the rst job for me is to gure out the data structure before creating table. Unlike the relational database, the ER-diagram is not suitable for organizing data structure in DynamoDB. In the DynamoDB, we treat each rows of sample data as an individual item and each column will be the attribute for the item. When we load the data into table, we are loading the items into the table in DynamoDB. For instance, we need to store the wireless information for different users based on sample data such as connecting time, disconnecting time, information of device, average usage and operating system they use. In this case, we treat each user as individual items and the wireless information for that user will be the attributes. Based on the sample data, there will be three tables in the Dynamodb. The rst table is the table which contains almost all the attributes except mac address and Onid. Instead, there will be two link tables which contains the mac address and Onid. Each table will have unique key pin and pin is actually Onid hash with some random strings.The figure 1 is showing the table structure on DynamoDB.

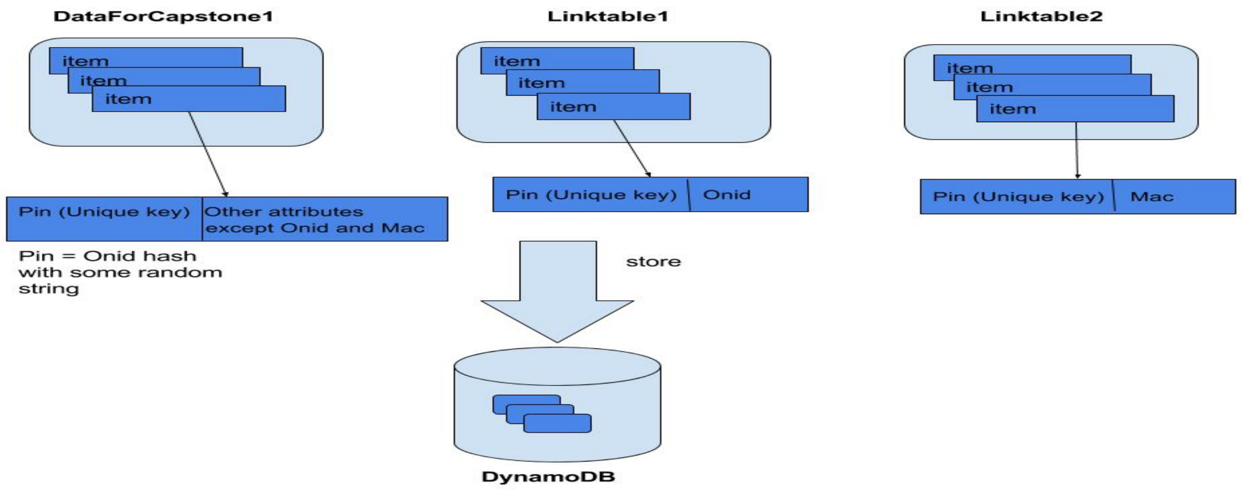


Figure 1: Table structure

2.2 Workflow of loading new data

The workflow of loading new data into DynamoDB starts with S3. At first, the system stores the new record in S3. After that, it checks whether the identifier for the device or person is new. If the unique identifiers are new, a new system identifier is generated based on a hash of the original device and student identifiers. The original identifiers and the new identifier are then added to a restricted access, DynamoDB Identity table. In the record, the original identifiers are replaced with the new system identifier, and the anonymized record is loaded into a DynamoDB data table. If the unique identifiers are not new, they are replaced with the system identifier already available in the Identity table, and the record is added to the data table.The figure 2 is workflow of loading new data.

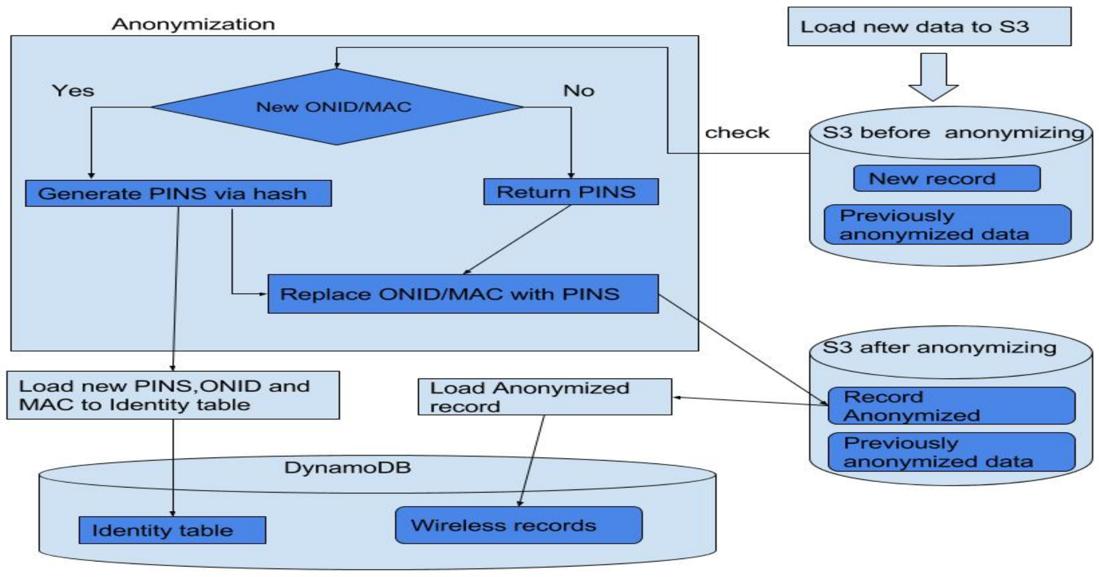


Figure 2: The workflow of loading new data

3 Importing Data from S3 to DynamoDB - Zhi Jiang

This portion describes how I import data from data storage into a database for this project. Specifically, I need to read the file containing data in S3 and then import this data into a specified table in DynamoDB. In this document, I would like to talk about how to implement this portion, what problems I encountered and how to solve them.

3.1 Create Table

When implementing this function, EC2, as a server, is used to connect S3 and DynamoDB. In other words, we can execute scripts or programs on EC2 to access data in these services and operate these data among them. In fact, there are many kinds of method to implement our functions on the server such as Python, Java or other frameworks. We chose Python because we were familiar with this kind of language. On the other hand, Amazon Web Services provide a library Boto3, which has many useful methods to deal with these services such as creating a table or accessing data.

The first step is to create a table. boto3 has provided useful methods to complete this portion. In following codes, this is a standard template to create a table. First of all, we access our dynamoDB and then create a table object. We set the table name and some properties of an attribute such as name and type. We still set capacity for this table.

```python
dynamodb = boto3.client('dynamodb')

# create table
table = dynamodb.create_table(
    # set name of table
    TableName = tname,
    # set information about attribute
    KeySchema = [
        {
            'AttributeName': 'PIN',
            'KeyType': 'HASH'
        }
    ],
    AttributeDefinitions = [
        {
            'AttributeName': 'PIN',
            'AttributeType': 'S'
        }
    ],
    ProvisionedThroughput = {
        'ReadCapacityUnits': 5,
        'WriteCapacityUnits': 5
    }
)
```

Listing 1: create table

3.2 Read Data

The second step is we should read data of the original file from S3, but we used an incorrect method to read data at the beginning. The following codes are an improper way to access data in S3. We were not familiar with this SDK, so we through each file should be regarded as an object, and then we just need to read the content of this object. But eventually, we failed to read data from variable content. In fact, we did not completely understand the role of an object type in this SDK. Object type is a complex constructor, and it contains much information.

```
s3 = boto3. resource('s3')
obj = s3.Object('bucket_name', 'file_name')
content = obj.get()['Body'].read()
```

Listing 2: inappropriate method to access S3 and read data

We believe that the optimal solution is the data is stored in the txt file, so we read document Boto3 document again, then we found the following method. The difference between this two method is the file we want to process is loaded into a temporary file and extension of its text. The benefit of this approach is it made this problem more familiar to us because we have done many similar tasks, which is I/O for a text file.

```
s3 = boto3.resource('s3')
s3.download_file("bucket_name", "file_name", "tmp.txt")
```

Listing 3: appropriate method to access S3 and read data

3.3 Import Data

So now what we should do was using a conventional method to read this temporary file in Python. In this implementation, We used method `table.batch writer()` to for do it. The purpose of this approach is you can both speed up the process and reduce the number of write requests made to the service if you are loading a lot of data at a time, so it is very appropriate to this project related big data.On the other hand, according to our client's requirement, we need to separate the data into the table. One is used to store Mac address of the user, one is used to store ONID of the user, and one is used to rest of data. The purpose of this method is our client would like to anonymize user's private information. In this part, Zhaoheng designed a new kind of structure of the database, which there is one common attribute 'PIN' used to connect these three tables. This attribute 'PIN' is created by hash function with user's ONID, so 'PIN' can be treated as a primary key. The basic idea to create a table is that it must contain a primary key. The purpose of this way is to make each row unique. The following codes show how to import data to a table without Mac address and ONID.

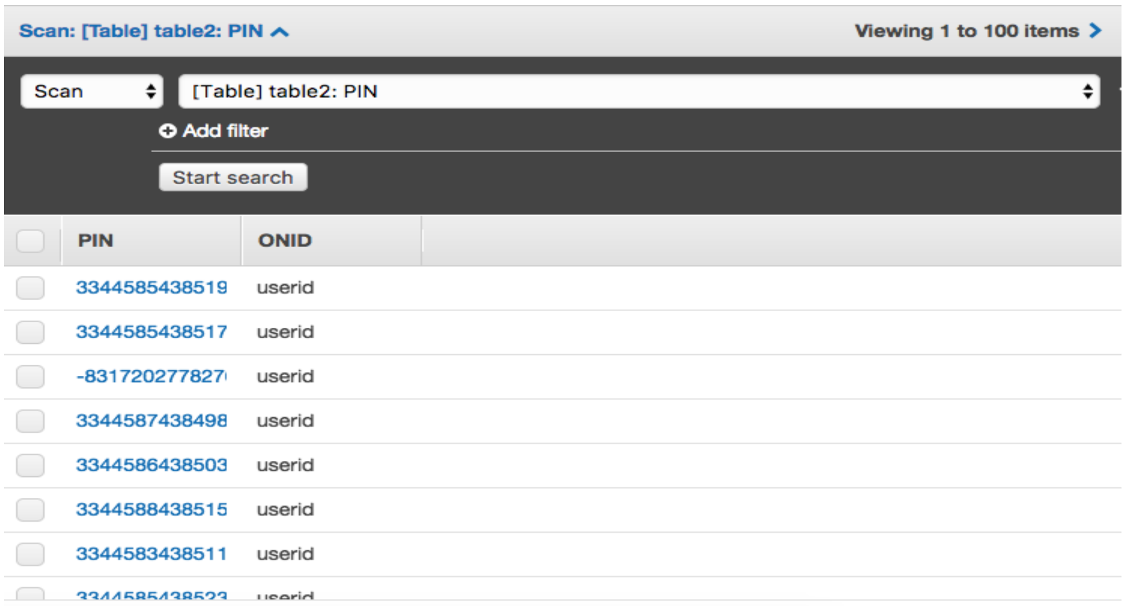
```
# open this file contains content of CSV
f = open('tmp.txt')

# set a variable as primary key
num = 0

# use table.batch_writer() method to load a lot of data
with table.batch_writer() as batch:
  for line in f:
    num += 1
     list_line = line.split(',')

```

```
        # create a new item and put it into a table
        batch.put_item(
          Item = {
            'PIN': str(hash(list_line[1]+str(num))),
            'attribute_name0': list_line[0],
            'attribute_name1': list_line[1],
            ...
            ...
            ...
          }
        )
    f.close()
```

Listing 4: data.py

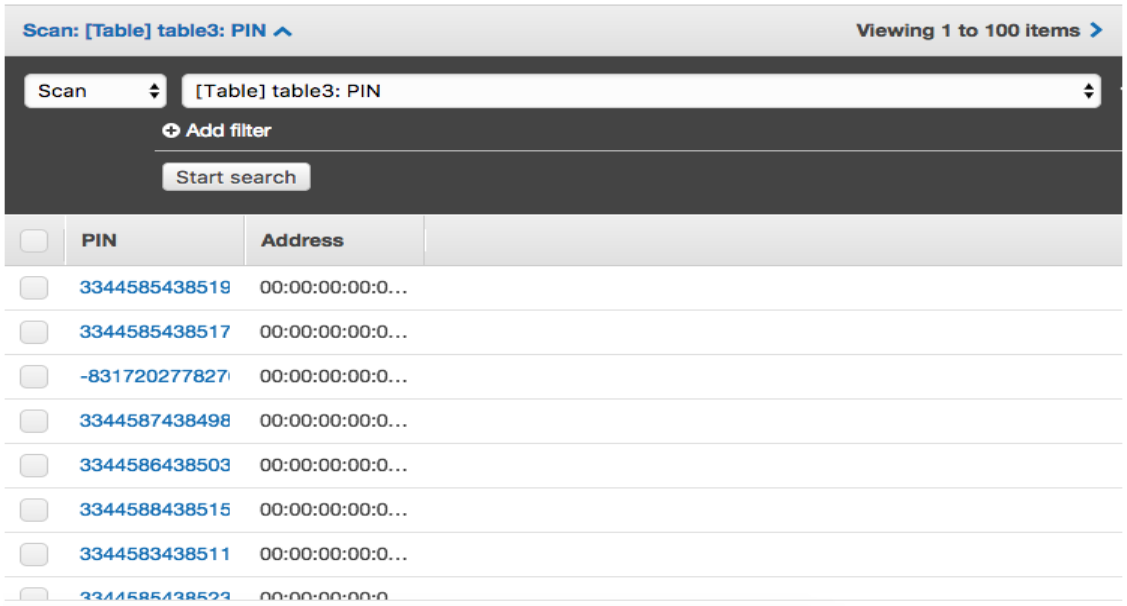
3.4 Results

We went to check results after we run this script. Fortunately, these results met our expectations. The following pictures are showing the result of each table. The first table shows the result of ONID and the second table is shows result of Mac address. The last table shows other data such as time and type and company of device. Overall, we successfully imported data from S3 into DynamoDB.



PIN	ONID
3344585438519	userid
3344585438517	userid
-831720277827	userid
3344587438498	userid
3344586438503	userid
3344588438515	userid
3344583438511	userid
3344585438523	userid

Figure 3: result of ONID table



Scan: [Table] table3: PIN

Viewing 1 to 100 items

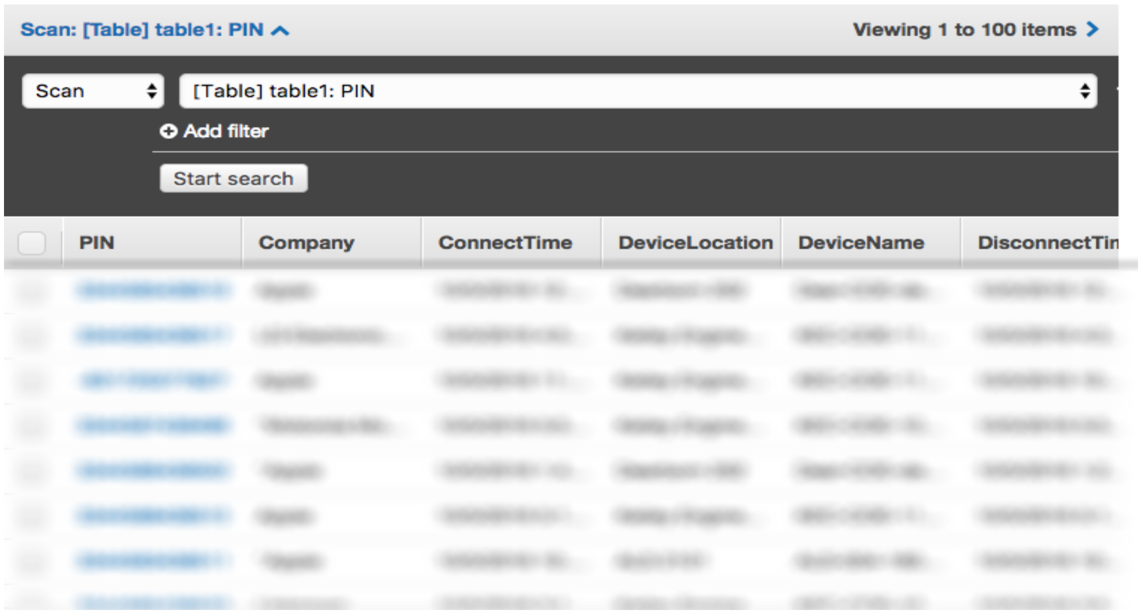
Scan | [Table] table3: PIN

Add filter

Start search

PIN	Address
3344585438519	00:00:00:00:0...
3344585438517	00:00:00:00:0...
-831720277827	00:00:00:00:0...
3344587438498	00:00:00:00:0...
3344586438503	00:00:00:00:0...
3344588438515	00:00:00:00:0...
3344583438511	00:00:00:00:0...
3344585438523	00:00:00:00:0

Figure 4: result of Mac address table



Scan: [Table] table1: PIN

Viewing 1 to 100 items

Scan | [Table] table1: PIN

Add filter

Start search

PIN	Company	ConnectTime	DeviceLocation	DeviceName	DisconnectTin
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Figure 5: result of Data table

4 QuickSight - Zhaoheng Wang

4.1 Steps For generating QuickSight Analysis

The first step is to generate a data source based on the processed data. This step can be easily done by clicking the export button on DynamoDB console. These data source are very important for Quicksight because the QuickSight requires these data source for creating data set and doing the analysis. Then, the Quicksight will create the data set by choose the data source on QuickSIght console. After the data set has been created, the visual can be generated by the data set. There are several types of visual on the Quicksight and we can choose one to generate the suitable graph for the analyzing results.

4.2 Screenshot For QuickSight Analysis

The figure 6 shows proportion of different OS which users use to access.Based on the graph, most of users are using Mac OS or IOS to access the wireless.

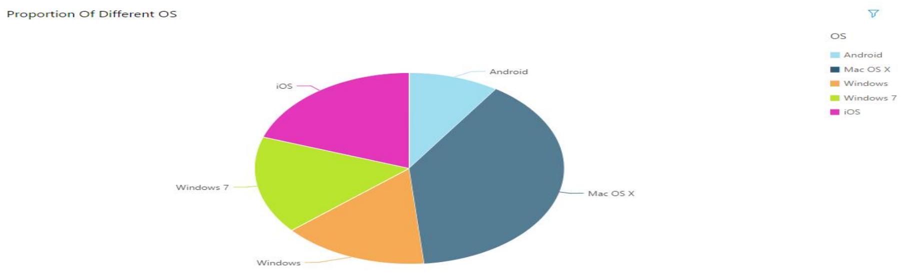


Figure 6: Proportion of Different OS

The figure 7 shows the relationship of total traffic in different connecting time and average usage in different Connecting Time. based on the graph, there are many users access to the server between 1pm to 4pm in 10/5/2016.

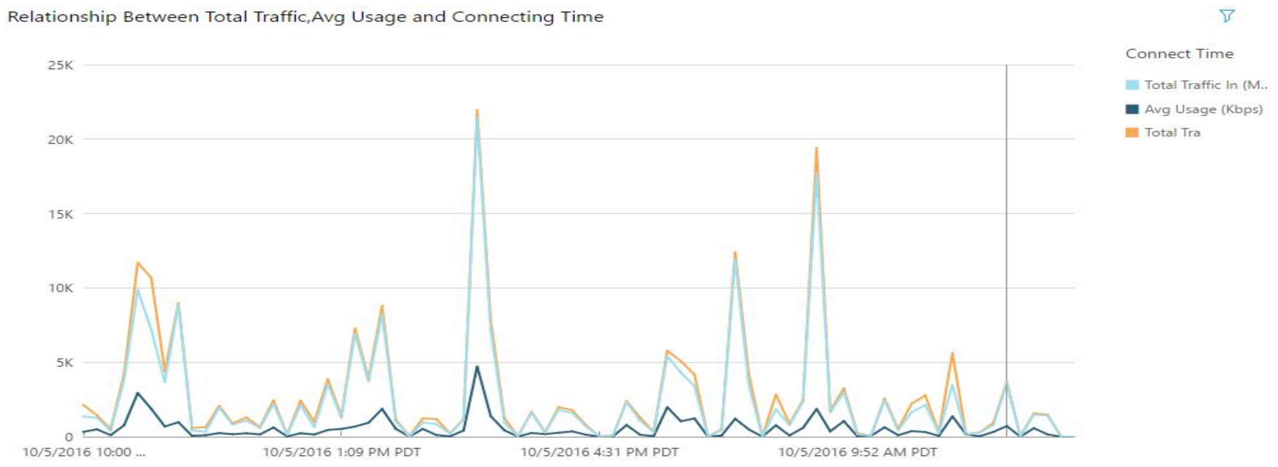


Figure 7: Total Traffic and Avg Usage in Different Connecting Time

The figure 8 shows Proportion of Different connection model that users use to access the server. Based on the graph, most of users are using 802.11n (5GHZ).

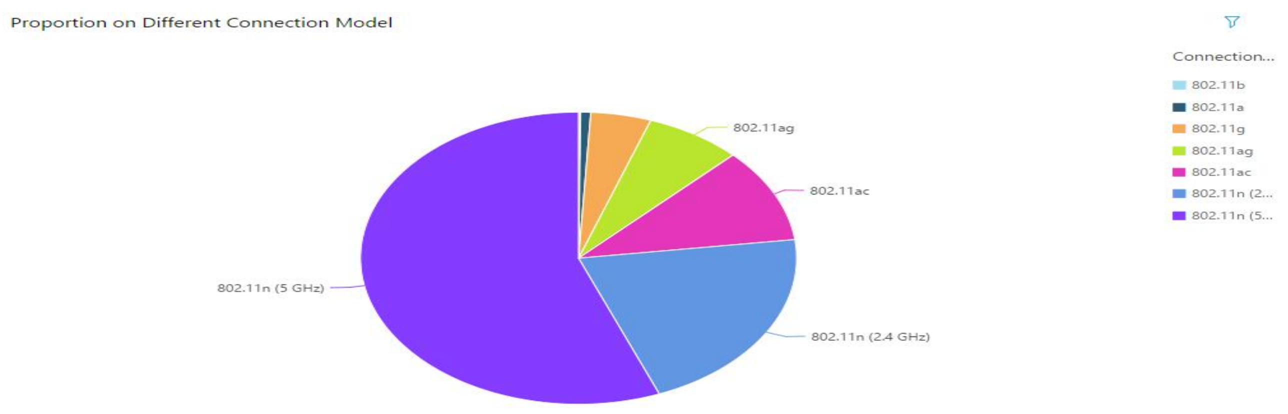


Figure 8: Proportion of Different Connection Model

4.3 Job And Things Left

My Job is to design the data structure for organizing the table and the workflow for loading data into the DynamoDB. Besides, I am also responsible for creating rudimentary analysis on QuickSight.Most of my job are finished on last term. In this term, I update the workflow based on the suggestion from the client. Furthermore, I update the graph on QuickSight because some of the data in sample data are not filtered. My group plan to schedule an appointment with our client for checking the whole project.

5 Tests for Functionality - Isaac Chan

5.1 Original Analysis

Tests for functionality ensures that users are able to interact with the system. They will need to use different utilities for different interactions, such as monitoring resources, monitoring performance, managing data, and performing different methods of analysis. As shown previously, AWS Quicksight is able to build visualizations from the data. However, its unable to perform more complex analysis, such as machine learning. Since the primary users of the implemented database will be data analyzers, our original test for functionality was to run basic data analysis code from AWS EMR, or Elastic MapReduce, to ensure data analysis is possible. For example, we can run analysis scripts to determine the relationship between time a user stays on a specific wireless network compared with their location on campus.

5.2 Data Standardization

The current data analyst tested our implemented solution and shifted the requirement. The database worked well for analysis purposes. However, she had concerns about loading raw data into the database. The raw data had certain contents that were not standardized, such as duplicates and fields that didnt work under the column. Previously, the analyst would pull some raw data, cleanse it locally on her machine, and run analysis on a small sample. This was not a scalable solution. For real use, cleansing all the data would not be sustainable for extremely large data sets.

Instead of running data analysis code on AWS EMR, we would run data cleansing code on EMR. EMR was chosen because it had many analysis tools preinstalled and has options for permissions to connect to the other Amazon tools we are using. It was a long process of working with the data analyst to give me all the permissions required to even start up the cluster. AWS EMR runs on top of an EC2 cluster, that we have already used. I had to obtain permissions to connect EMR to EC2. Understanding those permissions took a lot of working closely with the data analyst, who served as our AWS administrator.

The code was written in Scala and used Apache Spark as a processing engine. In order to run the Scala code on EMR, I had to package it locally in SBT, a dependency manager, and build it into a JAR, which could then be run as a Spark job on EMR.

```
spark-submit \
  dataprep_2.11-1.0.jar \
  fakeTerm s3://isaac-data/mybox-selected/
```

Listing 5: Spark Submit

After running it on sample data, the code outputted a split series of CSV files to Amazon S3, which could then be loaded into DynamoDB.

Analysis on data in DynamoDB is almost always the same process, because the data format inside is always consistent. However, in order to efficiently process big data in the Amazon cloud, we had to shift requirements to cleanse and load data all within the AWS environment. This demonstrated a path forward for future usage of our project, from start to finish of standardizing raw data all the way to loading it into the database.

6 Performance Optimizations - Isaac Chan

Performance metrics for database functionality are important to ensure the usability of our implemented solution. We would assess performance from typical database operations: data inserts, updates, and reads. From this we can obtain a baseline for the database performance, and examine how varying data and query loads compare to the baseline.

After meeting with the data analyst, we found out that the scenario they are expecting is only loading approximately 80 megabytes of data per day, which is an incredibly small amount and loads easily into our database using the Amazon services. Therefore, we eliminated this task in favor for functionality tests.

7 Database Security - Isaac Chan

For database security, unfortunately NoSQL databases do not support external encryption tools. Therefore we used a combination of methods to ensure security of our solution, by restricting access and preventing malicious utilization of the data. The options we chose are AWSs user authentication policies, and encrypting sensitive data fields. Because of the lack of external tool support by NoSQL databases, we must resort to using modular security methods.

AWS user authentication policies must be implemented by the AWS administrator. When I worked with the data analyst for permissions to EMR, we found that each service within AWS had to be restricted individually. Ultimately, I had less permissions than she realized and it takes a lot of separate permissions to give someone access to a service.

Previously, we had thought that encrypting sensitive data fields would also fall to the AWS administrator. After speaking with the data analyst, theyd like to add data encryption for the future usage, not simply for us as developers as we had previously thought. We have implemented this already, as mentioned before in this presentation, by hashing the sensitive fields, putting the real fields in separate tables, and using the hashes as identifiers to link the data tables together without unauthorized users seeing this sensitive data.