SensitiveMood Project Report

Spring 2017: COMS 6998 Sec 5 Cloud Computing and Big Data

## **Team Apollo Group 19** Ming Zhou [mz2591@columbia.edu](mailto:mz2591@columbia.edu) Backend + Swift (IOS) Frontend Pulkit Jain [pj2313@columbia.edu](mailto:pj2313@columbia.edu) Ionic (Android) Frontend Emily Hua [yh2901@columbia.edu](mailto:yh2901@columbia.edu) Modelling + DataBase

## **Project Description**

SensitiveMood is a mobile app that uses Machine Learning combined with traditional statistics to keep track of your friends’ posts on Twitter, analyze their sentiment status and report their daily mood score directly to you. It will also send email alert when your friends of interest is going through a major mood swing. SensitiveMood also alert you when your phone is encountering abnormal movement, which can be further developed into functionalities including user authentication and theft alert.

SensitiveMood aims to increase the humane interaction among friends facilitated with online Deep Learning APIs like IBM Natural Language Understanding. We collect users’ public twitter information as well as accelerometer data associated with each device. The algorithm will take users’ daily activities into account and generate the sentiment score from their posts, profile user behaviors using unsupervised clustering and report mood swing and abnormal behavior when the aggregated score reaches the threshold.

## **Architecture Solution**

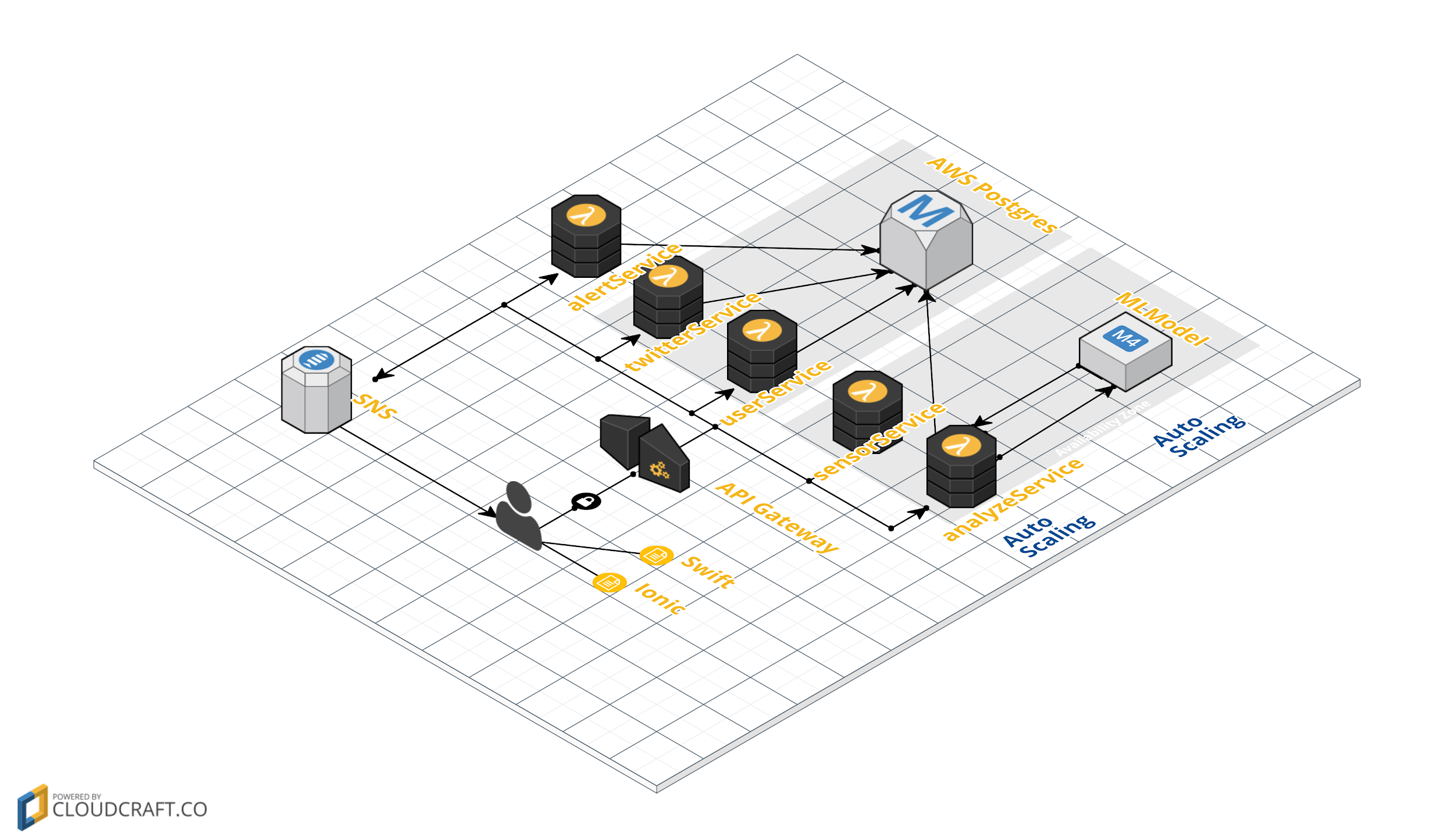
In order to approach a much lightweight and robust solution, we choose the modern microservices architecture. We are greatly inspired by the guest lecturer and impressed by the flexibility it will bring to the table. Being inherently independent from actual machines, it scales very well, and more importantly, it scales automatically. Microservices architecture can adjust to a large amount of requests and will only initialize just enough services to finish the job as soon as possible, but still cost effectively. We divide the entire backend system into 5 different microservices, UserService, TwitterService, SensorService, AlertService, and a powerful, machine learning based AnalyzeService. With extensibility in mind, we choose the modern serverless framework to host our microservices. The framework of our choice, AWS Lambda, is provided by Amazon with extensive configuration and guaranteed always online. To expose public APIs to the end clients, we find API Gateway is a most suitable service. Its advanced design and close integration with other AWS services makes it the best choice among all possible technologies, including Flask, ExpressJS or SpringMVC. For database, we use AWS RDS-Postgres, for its scalability and easy replication. The relational database is chosen over Dynamo as our application requires extensive aggregation over user data for the machine learning model to consume, which operation is lacking in noSQL databases.

We have configured the AWS Simple Notification Service (SNS) to be the one that pushing real time notification to the clients. Once we have detected any abnormal behavior from our clients, we will immediately analyze the situation, make a record and notify their followers in no time. It is extremely important for us to provide such functionality as our goal-to build an entire community for people to help and support each other-is very much time sensitive. For further development, we will introduce multi-layered notification system, so the information will always reach our clients as early as possible.

To maximize user experience, we have provided a native, swift written iOS application, an well designed Android application and a web application which can be accessed through any platforms. Of course, for a time sensitive task, the mobile application should receive most attention. So we chose native and lightening fast Swift for our implementation of iOS application as we want to eliminate any possible latency, and provide with our user the best experience.

However, serverless framework does not allow us to constantly gathering data and improve our machine learning model on the fly. So we open up a separate instance to allow model improvement after we acquire more user-centric data.

**Architecture Diagram:**

****

## **Modelling**

This project involves two machine learning model, both of which uses K-means unsupervised clustering to profile user/device behavior based on either twitter data or accelerometer data.

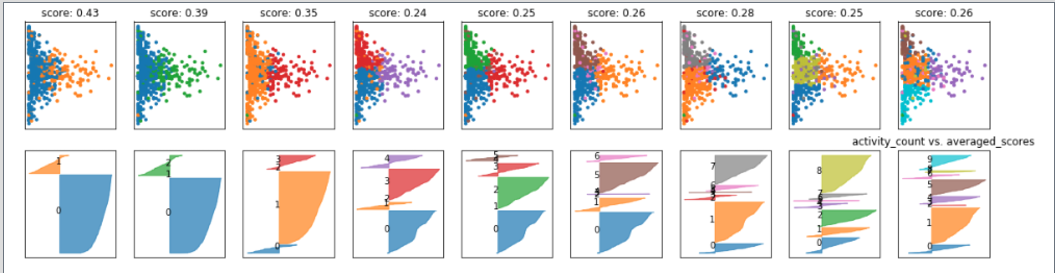
1. **Twitter-data based model**

In order to alert for mood swing, we need to define the threshold beyond which we identify the behavior as sentiment altering. This threshold is set based on our first machine learning model. The simplest threshold could be defined as the standard deviation of a user’s tweet sentiment score within a context window; however, we want to avoid setting threshold based on a single feature which may cause false positive decisions. Hence, we develop a unsupervised clustering model based on ~1000 twitter users’ various features, which includes:

*followerCount, favoritesCount，friendsCount，statusesCount, activityCount， averagedSentimentScores*

We use *tweepy* to get over 5000 tweets whose user once sent tweets in Manhattan and are captured by our 24 hours tweet firehose. Each individual Tweet sentiment score is obtained through IBM Natural Language Understanding API. And we process the data to get aggregated features values through a 7-day window.

We identify number of clusters based on silhouette score, which measures the compactness among each cluster. The highest silhouette is achieved with 2 clusters, which can be shown below, where we plot *activityCount* against *averagedSentimentScores:* (each column has plots plotted based on 2,3,4,5,6,7,8,9, 10 hyperparameter-number of clusters accordingly)

**

Upon examination of each cluster centroid, we find out that the smaller cluster can be defined as a group of user who has a large amount of followers, favorites count and huge friends count, status count and activities, but relatively negative tweet sentiment score. While the larger cluster’s center has fewer follower, favorites, friends, status and activity account but a more positive averaged sentiment score within a one week window.

Based on the assigned class label by our model, we calculate the standard deviation of tweet sentiment score and activities for each cluster, and used that as the alert threshold. In this way, the alert threshold is defined based on user profiles which is established by the aforementioned features.

Our alert logic thus becomes:

Step1. For each new user, wait till we have one-week (T1) worth of data, aggregate it and send feature values to the clustering model for profiling.

Step2. Use the model prediction (cluster number) to set its alert threshold.

Step3. At the end of second week (T2), we calculate the accumulated activities and averaged sentiment score in the past week, comparing against the threshold and send out alerts if apply.

Step4. Redo Step1-2-3 (use the current T2’s aggregated features to reprofile user, and get updated threshold, and apply T3’s sentiment and activity score to send alert)

1. **Accelerometer-data based model**

As suggested by Professor Sahu, we gather some phone-based data to expand our mood-swing model to a more general anomaly detection framework. In our case, as we are building a native IOS app, phone-data accessibility is much more restricted than Android counterpart. After some study, we build the second model with similar logic\* as the first model but now based on accelerometer data. \*(clustering, and set threshold as the standard deviation of each cluster)

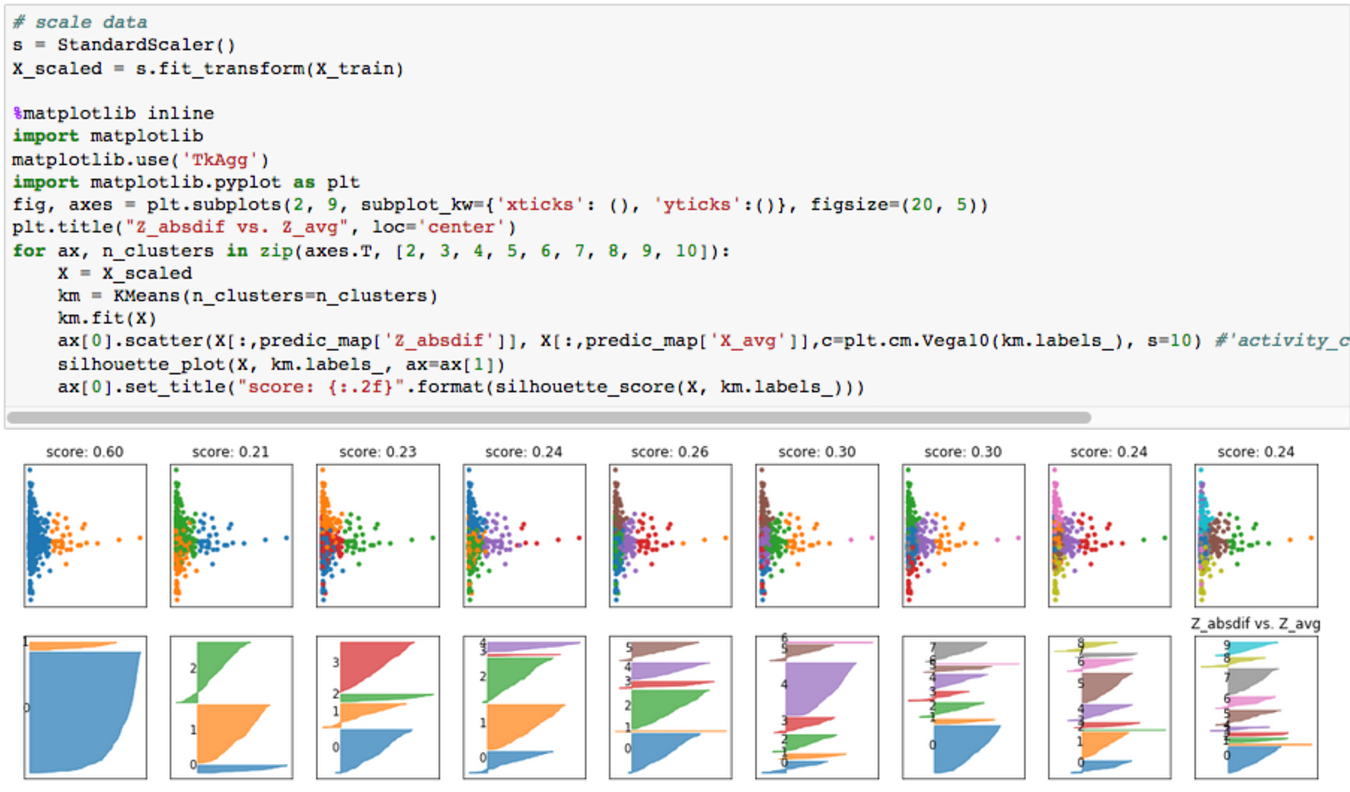
This clustering model is trained on accelerometer data we acquired from [a Kaggle competition](https://www.kaggle.com/c/accelerometer-biometric-competition/data), which contains the raw X,Y,Z coordinates of over 300 devices on 200 millisecond sampling rate.

We use *Hadoop Hive* to process the raw data (29,563,984 rows), and construct features including:

*X\_avg, Y\_avg, Z\_avg, X\_absdif, Y\_absdif, Z\_absdif*

, where X\_avg is averaged X coordinates within a second interval  
Y\_avg is averaged Y coordinates within a second interval  
Z\_avg is averaged Z coordinates within a second interval  
X\_absdif is the accumulated absolute difference in X value within a second interval  
Y\_absdif is the accumulated absolute difference in Y value within a second interval  
Z\_absdif is the accumulated absolute difference in Z value within a second interval  
The resulting processed data contains 10,362 rows.

Again, we choose 2 clusters based on it having the highest silhouette score. A silhouette plot with scores between different number of clusters (2-10 clusters) and scatter plot of  *Z\_avg vs. Z\_absdif* is demonstrated below:

Based on each cluster centroid, we define the smaller cluster (labeled as 1) as those device whose *Z\_absdif, Y\_absdif, X\_absdif* values are large, which indicate large up-down movement (lift: picking up or dropping the device), large tilt (lean back or forward), and large twist(like turning a doorknob). The larger cluster’s center value indicates it is relatively more stable. 

With the assigned cluster label, we calculate the standard deviation of each cluster and use them as the threshold for alerting abnormal phone movement based on accelerometer data.

Our alert logic is developed as:

Step1: get aggregated second level accelerometer data for each device, send them through the model to profile the behavior. Get the alert threshold based on predicted cluster number.

Step2: schedule a job to get new accelerometer data of a later context window, if upon re-clustering, the new cluster number changed from cluster 0 to cluster 1, then we alert the user that the device of interest is undergoing drastic movement; if upon sampling, the newly aggregated accelerometer data breaches the stored threshold, we alert the user of the device’s abnormal behavior as well.

With SQL, we can change the window size to any interval. For the current dataset, we train the model on the second level, as the training dataset from Kaggle has the longest consecutive sequences last up to a second.

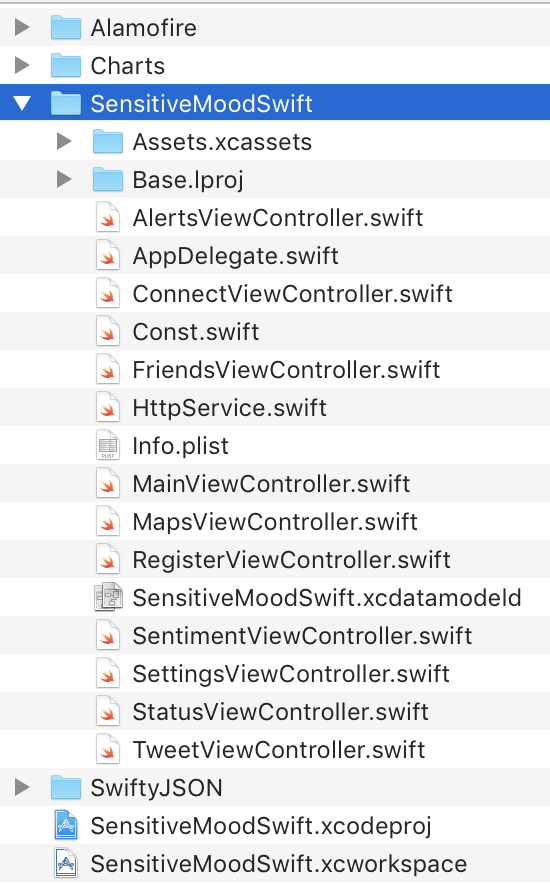
After we accumulate enough this kind of accelerometer data, we believe that this model can be expand to user authentication.

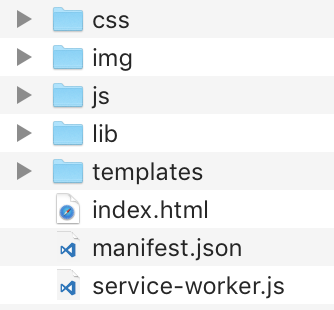
To summarize, our clustering model profiles behaviors and we generate standard deviation from each cluster and use it as basis for alert threshold. This methodology is developed by us and it is not a commonly seen outlier detection model, which often uses EllipticEnvelope or Isolation Forest. We refrained from those outlier detection models as we do not have enough user data nor features to approximate a Gaussian distribution (Gaussian distribution could be a wild assumption anyway), which is strictly required by generative model like EllipticEnvelope. Even though Isolation Forest might work, it requires us to build an independent model for each user, which creates maintenance issue in the long run. For this project, we combines machine learning and traditional statistical approach to build this alert pipeline that is highly maintainable but robust enough as it incorporates various aspect of a user’s daily activity.

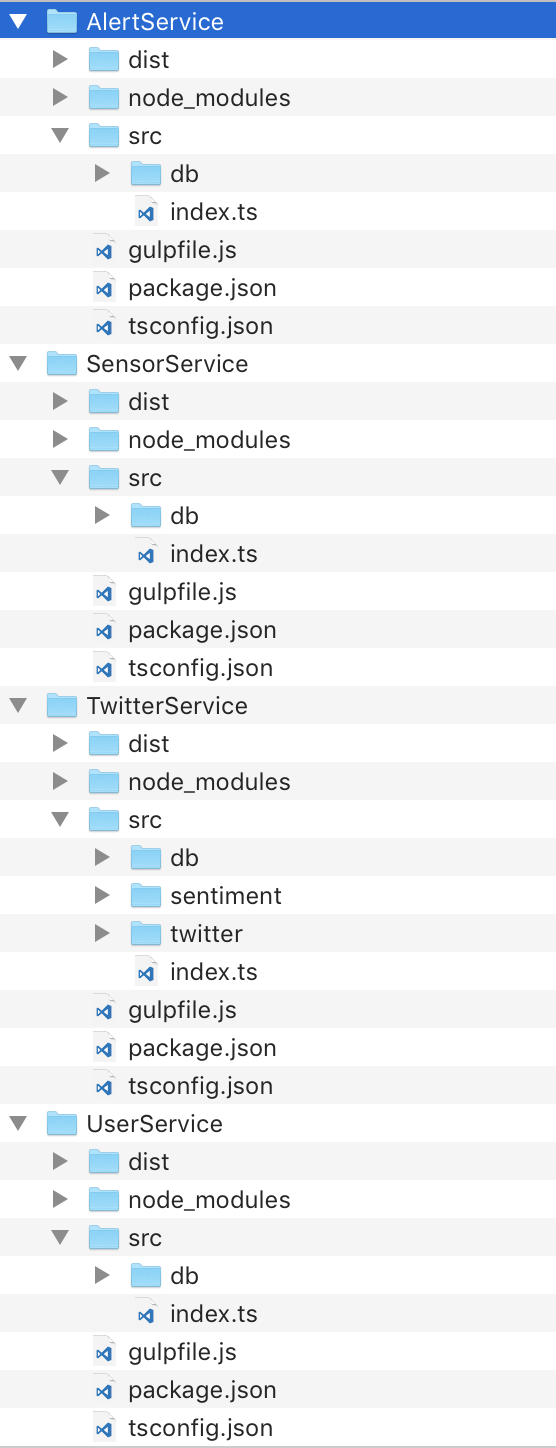
These two models are wrapped with Python Flask and accessible to Lambda query through RESTful APIs.

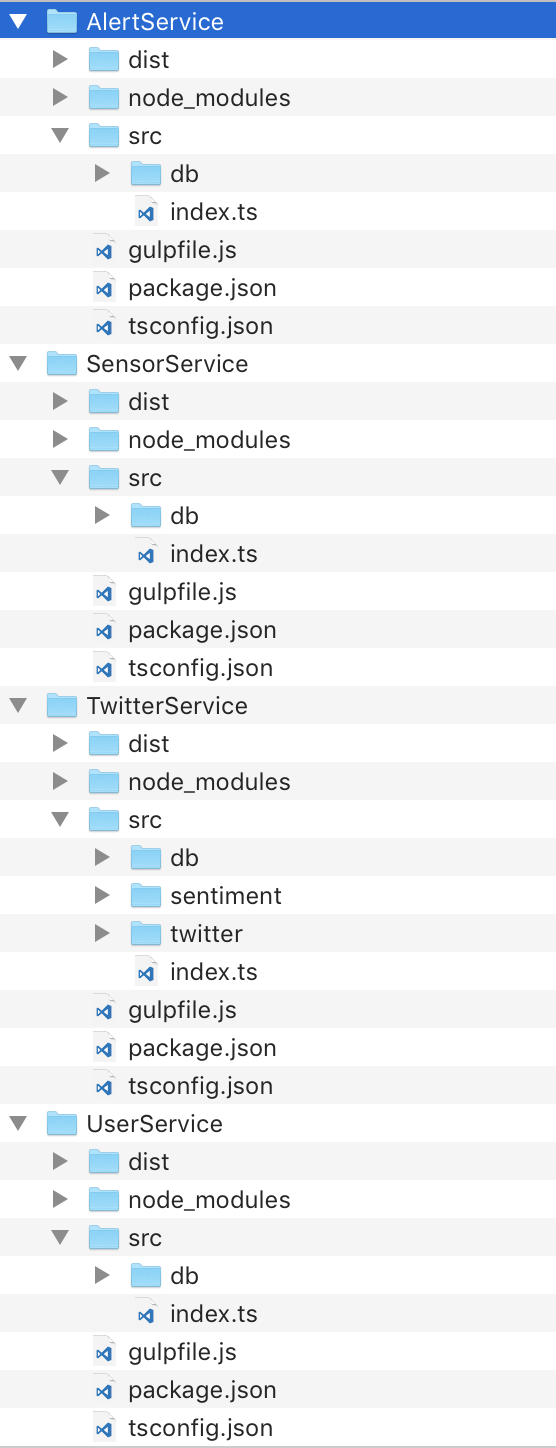
**Code Design**

For the purpose of decoupling the components in our system. We have separated out the frontend system, backend services, mobile apps (iOS and Android) and the machine learning. All of the components are connecting using RESTful APIs and will only transfer data.



****

The most important part in the actual code (besides machine learning models), are the microservices. We have chosen microservice architecture not only because it is an advanced architecture and will bring so much to the table, but also it enables us to focus on the actual code logic, instead of worrying about the environment and integration with other components. That is why we can break down the backend code to its very skeleton, and produce beautifully designed, and well implemented code. 

For the mobile app, we chose iOS platform and Swift 3 as the language. We would like to give our clients the best user experience, and have a chance to well integrate with the mobile platform, in order to get the most of it. We have been using accelerometer data as it gives us the most important motion once a user is behaving abnormal. 

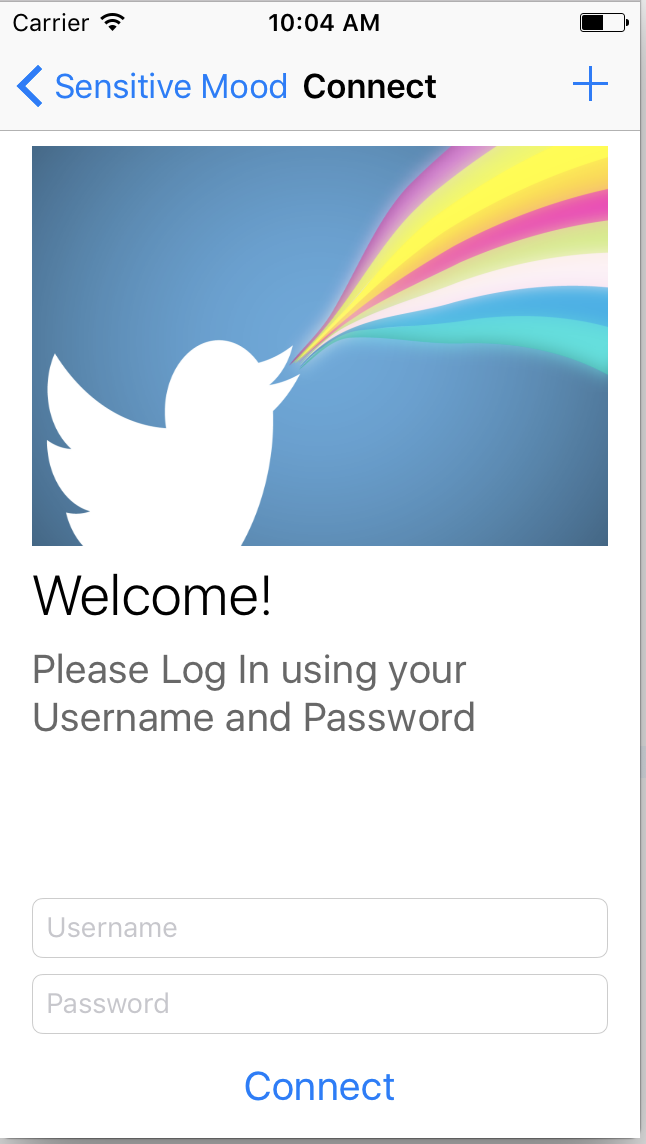
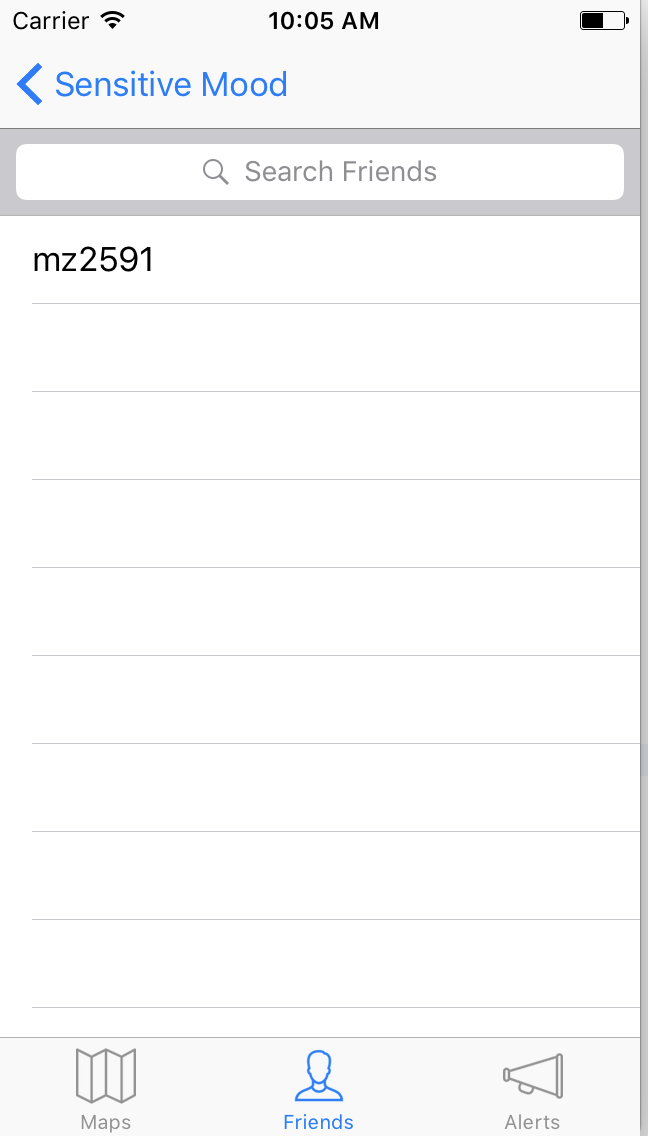
To build a community and an ecosystem, we have also released our Android app and web app, which are all based on modern Ionic hybrid application framework. We would like to target as many markets as we can so it will give our clients a great opportunity to connect with friends all over the world.

For the backend services, we have chosen AWS Lambda for hosting our microservices. AWS Lambda platform is so beautifully integrated with the traditional EC2 (used for machine learning modeling), API Gateway (used for exposing RESTful APIs), SNS (used for sending email, sms as well as regular notifications and alerts). The language of choice is TypeScript, which is a superset of JavaScript, that can run on NodeJS platform. It compiles to JavaScript so it can then be used with the powerful NodeJS ecosystem. We have also introduced many development tools, like GulpJS, in order to automate all the trivial but necessary work.

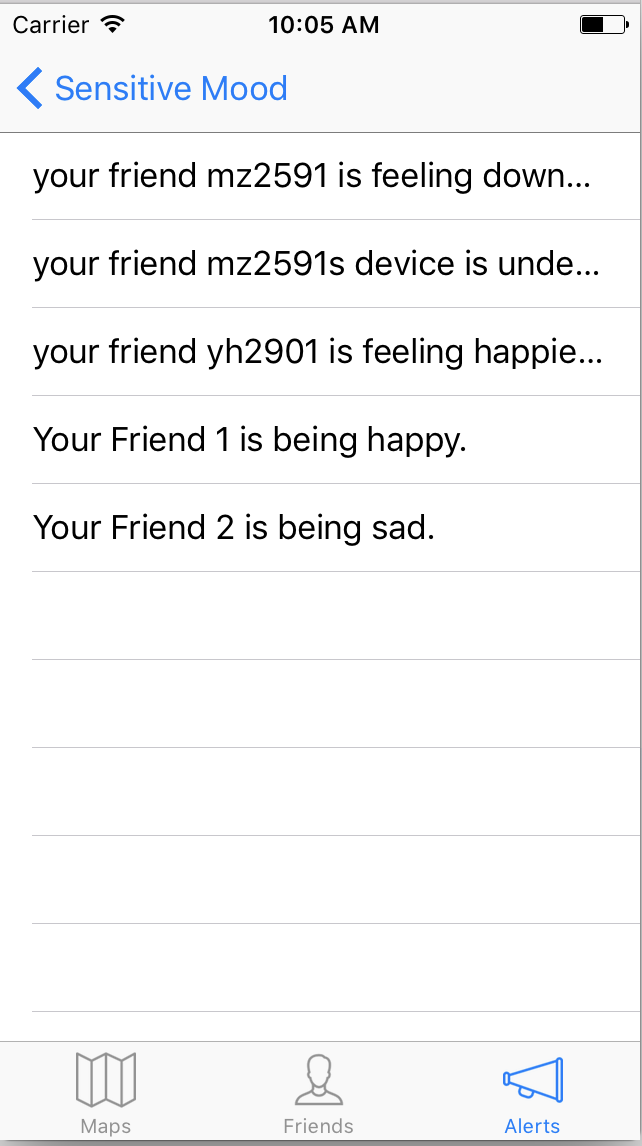
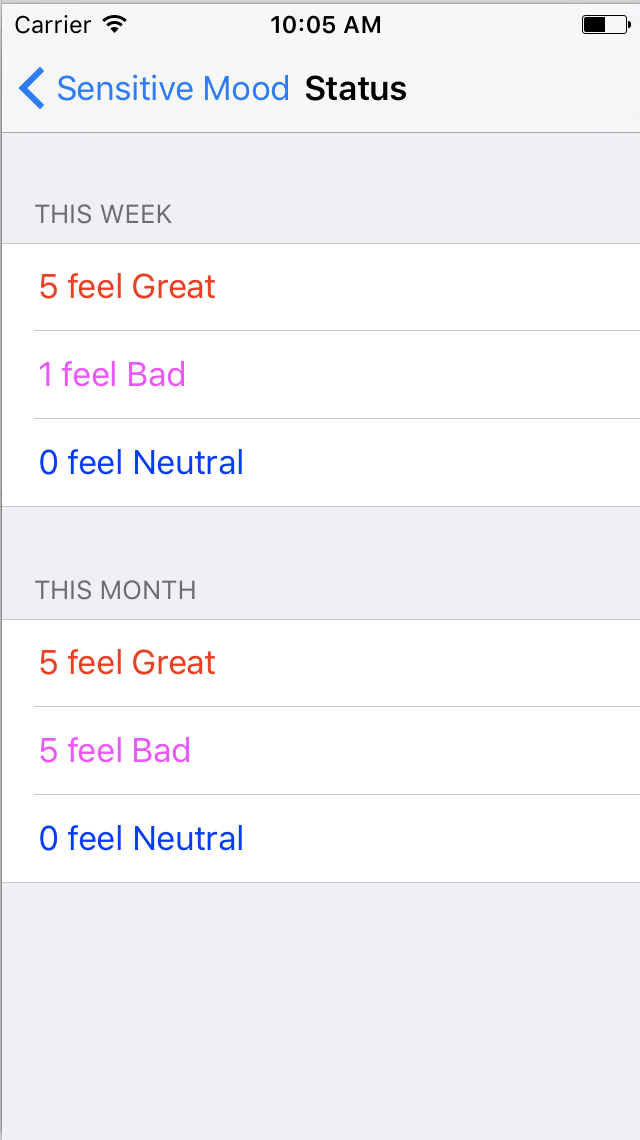
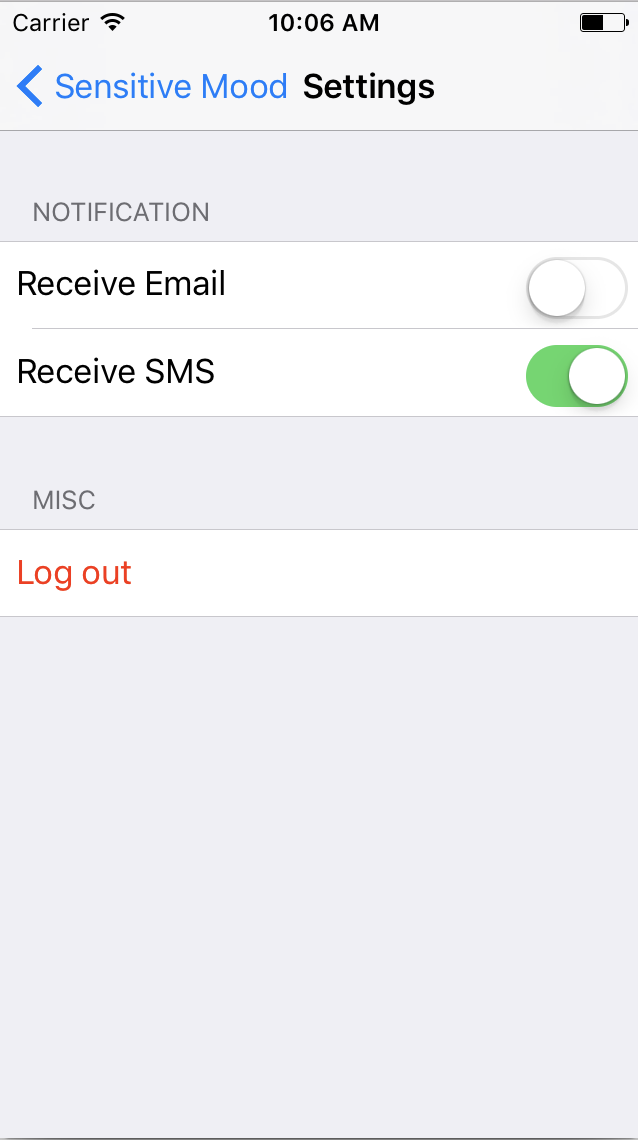
**Screenshots**

The following part is the screenshots of our applications, iOS and Android.

For iOS (native application written in Swift 3):

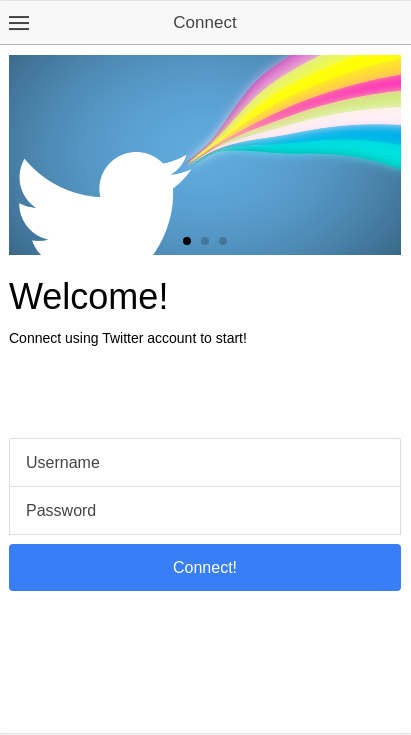
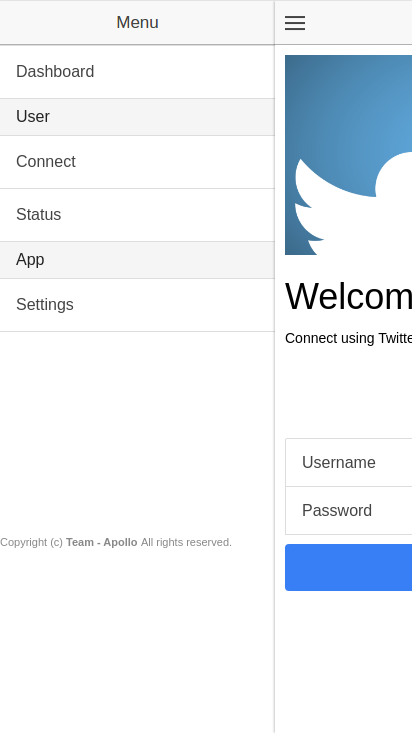
**  **

**Log in Latest Tweet Displayed on the Map Show Friend List**

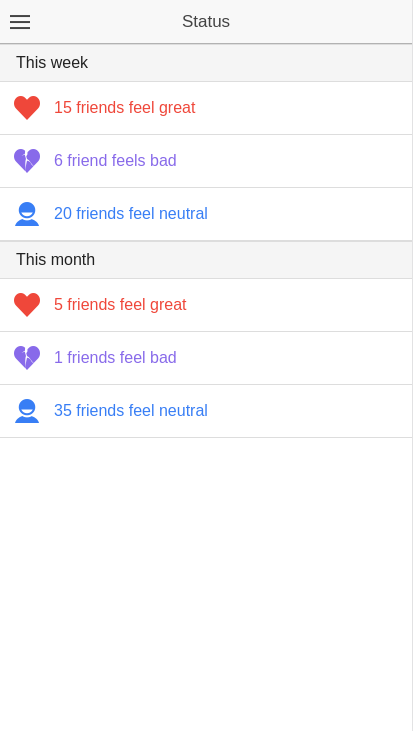
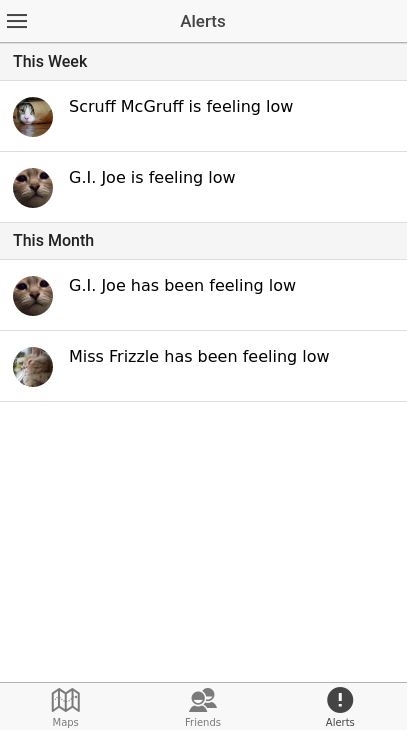
**  **

**Alert History Friend Statuses Summary Change Alert**

For Android and Web (hybrid app written with Ionic):

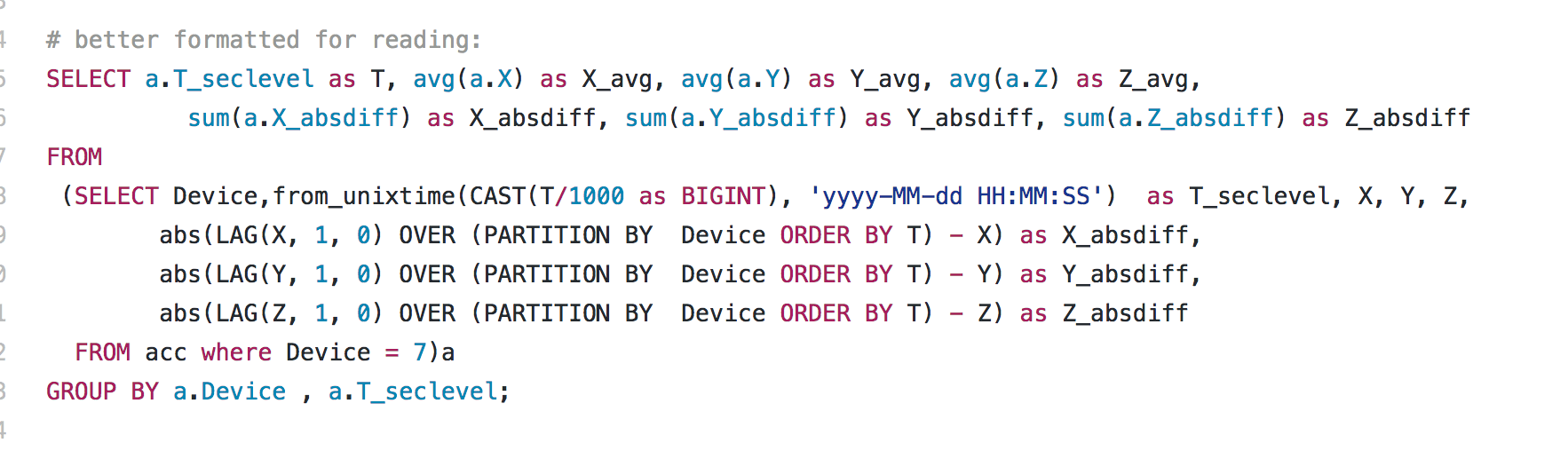
**Log in App Drawer Tweet Map**



**Friends Alerts Status**

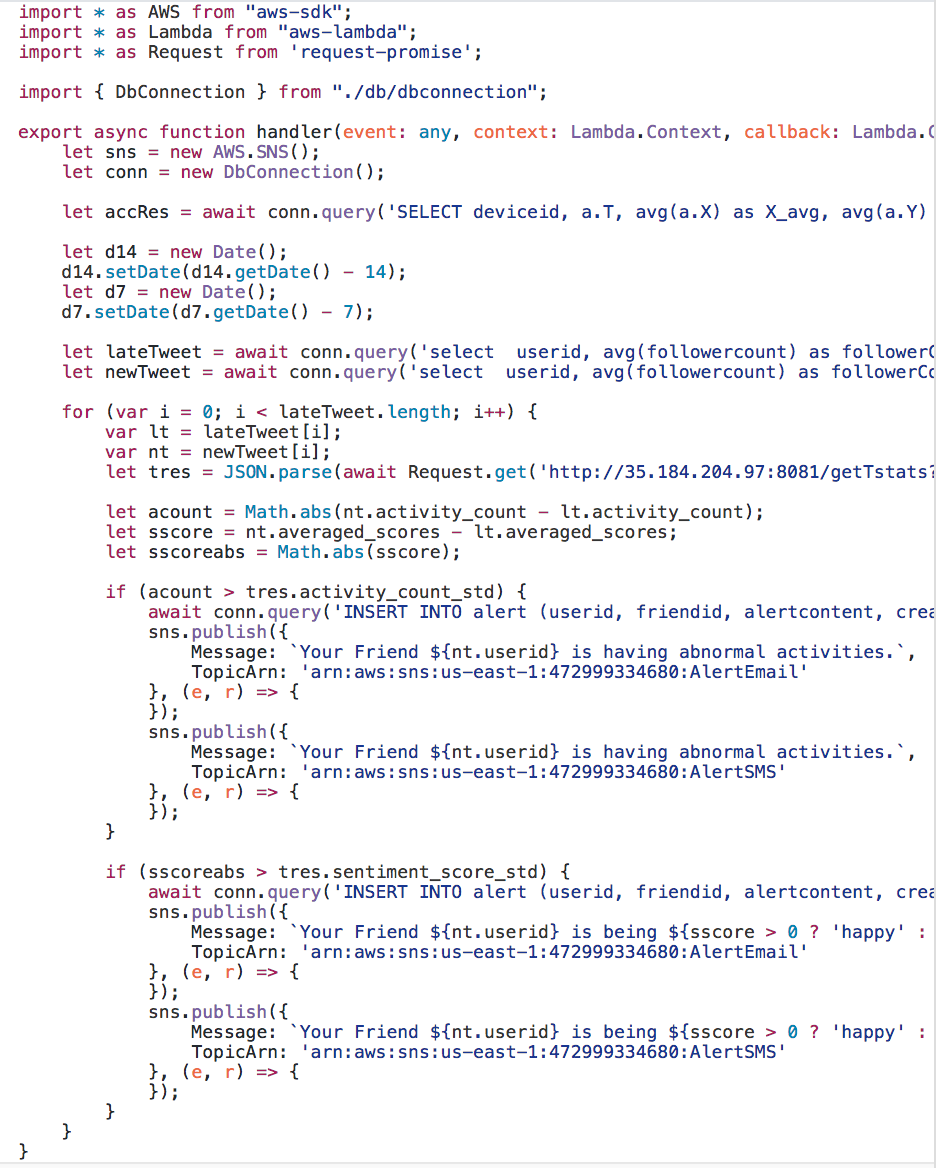
**Code Flow: (Language Summary and illustration of the Alert Pipeline)**

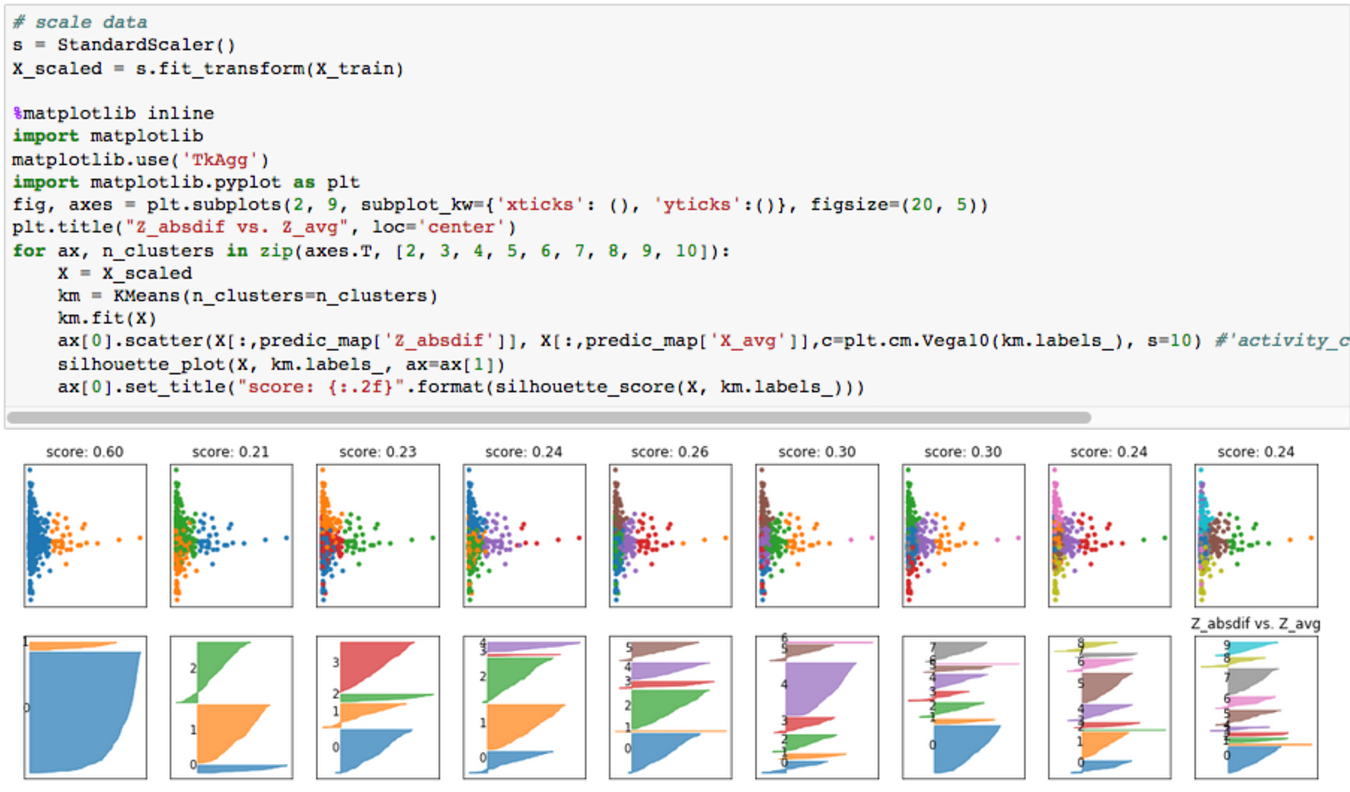
| **Services** | **Programming Language** |
| --- | --- |
| Lambda | TypeScript |
| Android Ionic | JavaScript |
| iOS | Swift |
| Machine Learning Models | Python (SKlearn) |
| Regular Query | Postgres SQL |
| Data Processing | Hive QL; Python (Pandas) |



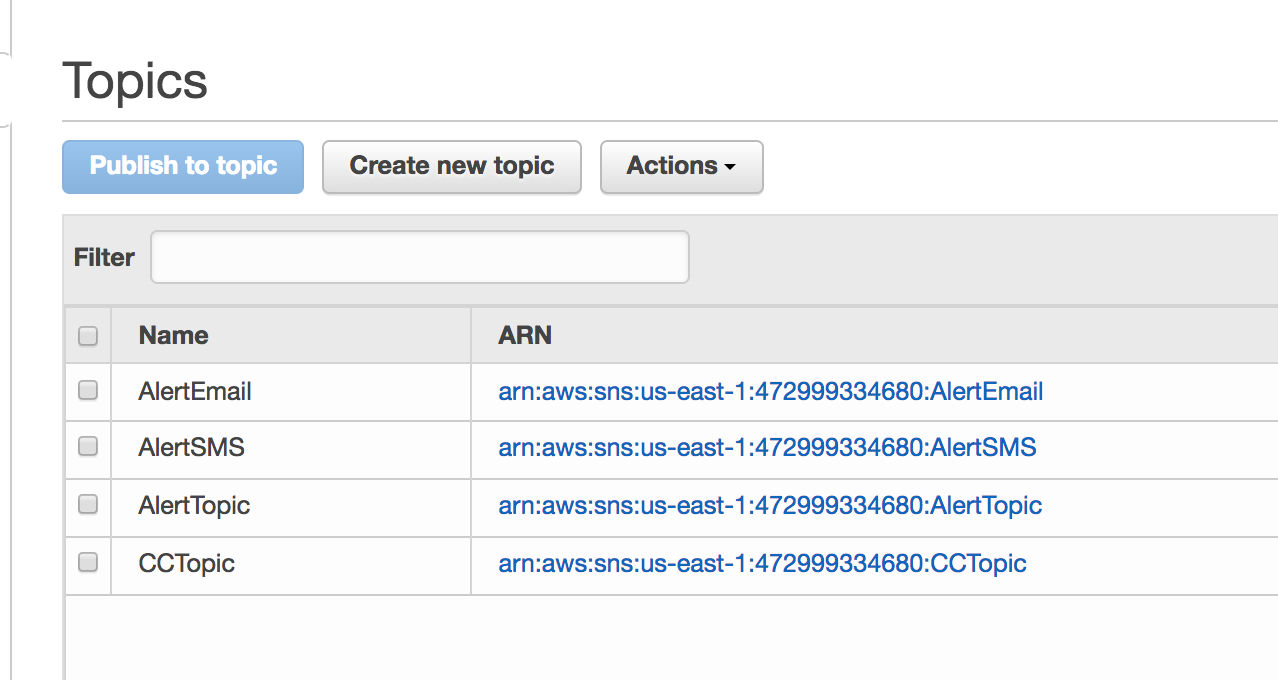
SQL to generate input for ML model

Alert service works on threshold and current user behavior and decide whether there should be an alert



Model returns threshold



SNS topic subscribed

Alerts