

Where's My Bus

Application:

The idea of the application is to ingest and process real time bus location data for SFO area in California.

Why tracking data for busses?

- Helps consumers better plan their commute and save time
- A better consumer experience can potentially result in increased revenue for the transit authority
- Real-time and aggregated data can be used for improving bus scheduling

Architecture:

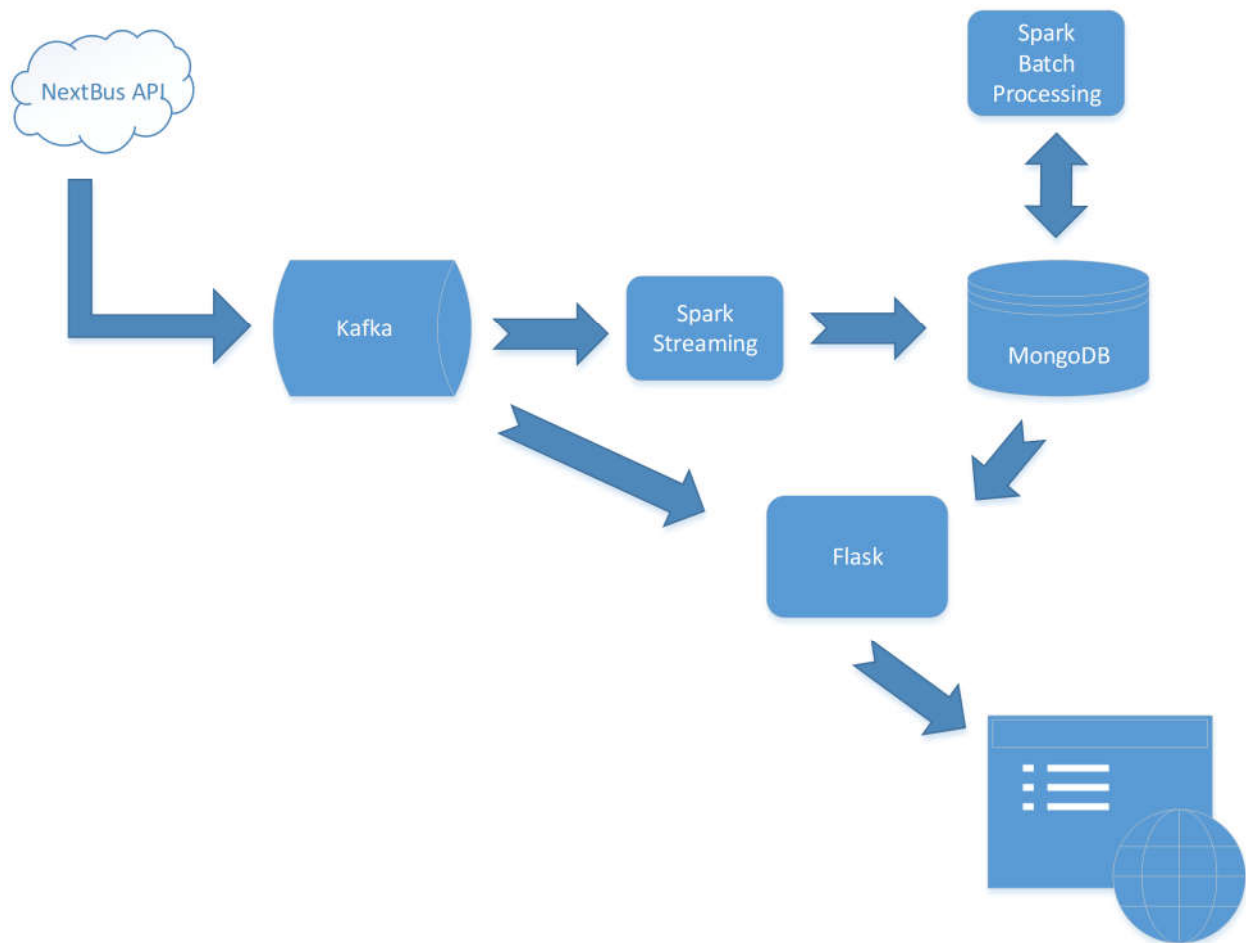
Figure 1 shows the high-level architecture of the application. It follows the Lambda Architecture.

Why Lambda Architecture?

- I needed to track the bus geo location in real time.
- I also want to aggregate hourly geolocation data to get insights into bus traffic patterns and identify hotspots

Since we needed both stream processing and batch processing, lambda architecture was the right fit. Following are the component that form the lambda architecture

1. Kafka: I decided to use kafka since we needed a messaging system through which we can have multiple consumers consuming at different rates, and ability to replay messages.
2. Spark: I used spark for both stream processing and batch processing
3. MongoDB: I used MongoDB for storage since it allows easy querying of geo location data. (A feature we ultimately did not use, but it makes it easier if in future we needed that functionality)
4. Flask: Flask was used for serving layer, and provides endpoint for streaming realtime data and heatmap data



Implementation:

Data Ingest:

For data ingestion I implemented a python script that runs in an infinite loop. It queries the NextBus API got current locations of all the busses in SFO area. The data is then cleaned , timestamped and pumped to kafka.

One of the primary disadvantage of the approach is that if the service fails, its doesn't auto recover and results in data loss.

Streaming:

Spark streaming job cleans the data and generates geohash of length 5,6,7,8 and stores to MongoDB. I decided on generating geohashes of varying length to allow experimentation with aggregation without have to reprocess the data.

To account for failure in streaming job or upgrades, I implemented checkpointing. With checkpointing spark saves the offsets of the last message processed. This way, the next time the job runs, it can continue from where it left off.

Bath Processing:

For batch processing I used spark jobs, that's kicked off every hour by a shell script. The spark job queries MongoDB for all data received in the last 1 hour. It then aggregates them on geohash and counts the number of rows. I decided not to use distinct bus ID, since I wanted a heatmap that shows busses at a given location during a 1 hr period. Which means if during traffic, the bus was moving so slow that the change in location was very small, I still wanted to count all of them.

Geohash:

For aggregating busses for a geolocation, i considered couple of approaches.

1. DBSCAN: This algorithm is similar to K-Means clustering, wherein it picks one geo location and then finds all busses close to it .The algorithm can be tuned using couple of parameters
 - a. **Epsilon** – This determines how far to search for points near a given point
 - b. **MinPoints** - This determines how many points should be present in the neighborhood of a given point in order to keep expanding a given cluster
2. Geohash:
 - a. Geohash encodes the latitude and longitude into a string of length n (n -> precision of hash)
 - b. The idea is that it generates same hash for coordinates in a given rectangle
 - c. The size of the rectangle is determined by the length of the hash
 - d. This allows easy aggregation based on location.
 - e. While visualizing, I decode the hash to get the geo coordinates.

It should be noted that decoding the geohash is lossy. I used geo hash of 7 since which has an accuracy of approximately 152m X 152m

Flask:

Flask provides the serving layer for both real time data and heat maps.

For realtime data , the app subscribes to kafka and reads real time geo location data and streams to the web UI via socketIO. The points are then plotted on google maps to show bus positions.

For heatmaps, it queries MongoDB to get aggregated data and plots it on Google Map using the HeatMap layer.

Technical Challenges:

Following are some of the technical challenges faced:

- Setting up flask for streaming real time data using socketIO
- Achieving exactly-once semantics in spark streaming
- Since the data ingest scripts queries NextBus API every 10ms, it creates bursts of data. Had to make sure streaming job is never overwhelmed by setting `spark.streaming.kafka.maxRatePerPartition`
- Setting up automation of spark aggregate job using Apache Airflow

Limitations And Extensions:

The project was implemented in a very short duration, and as a result has quiet a few limitations

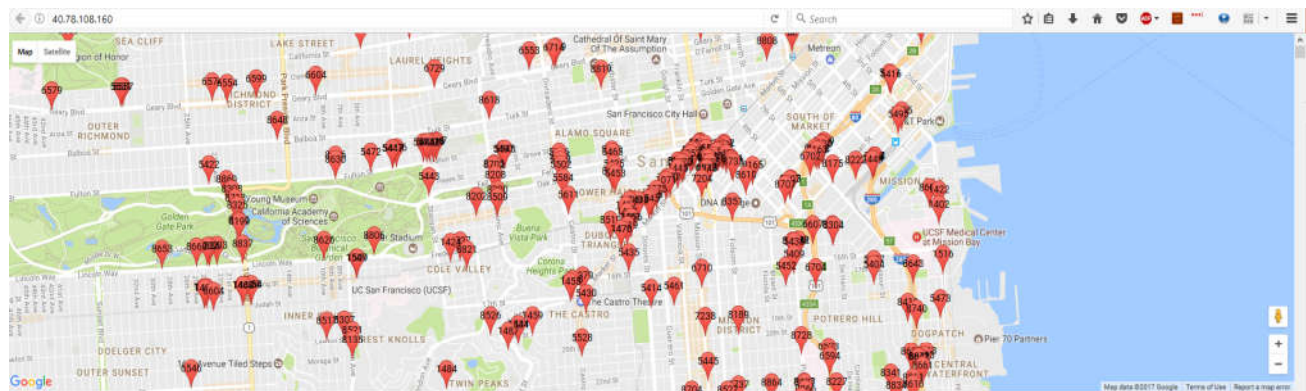
1. The data ingest has a single point of failure.
2. Aggregate job doesn't backfill i.e If the job script is down for few hours and then restarted, it wont backfill the missed hours
3. The infrastructure setup needs a little bit more configuration to run in cluster mode.
4. If the WebUI for real time tracking is kept open for too long, it can get sluggish due to the number of markers

Extensions:

1. Add more data sources (different cities, modes of transport etc)
2. Implement Data ingest as a more fault tolerant service
3. Aggregate job can be automated using Apache Airflow, which allows backfilling and also fault tolerance.
4. Making SocketIO serve multiple clients for streaming data

Results:

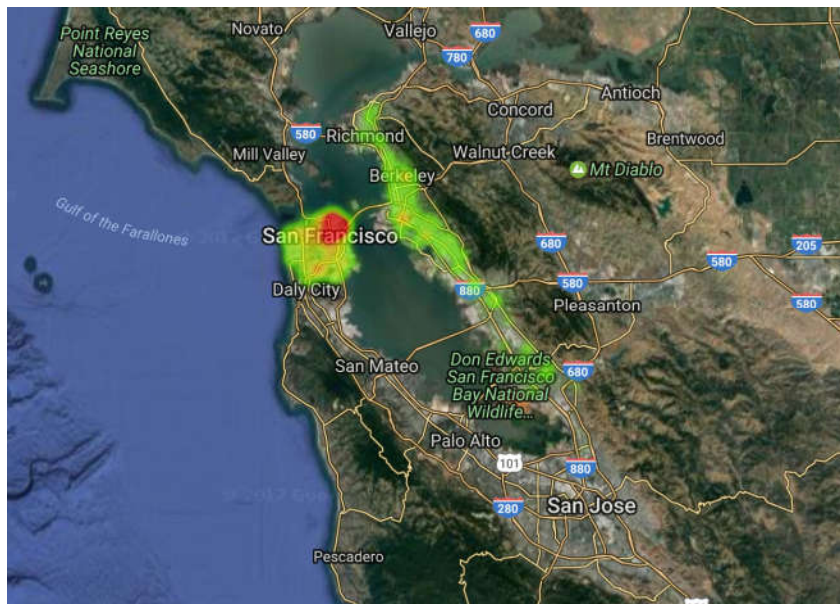
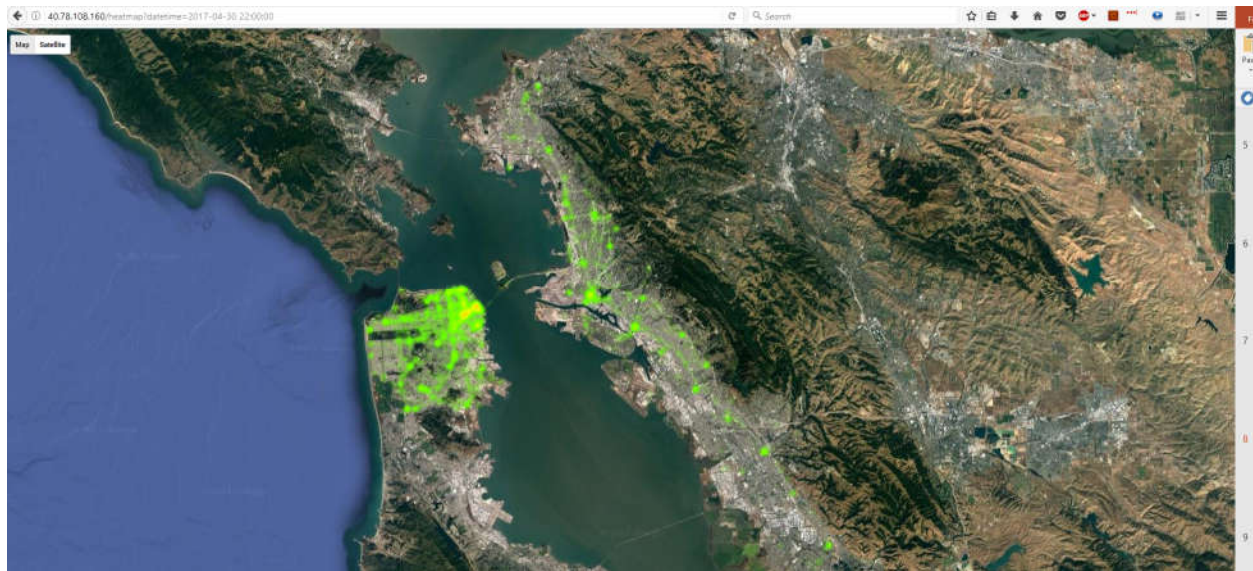
Realtime Streaming data



Receive:

```
Received #undefined: Connected
Received #223: 223-38-456659,-122.717248
Received #224: 224-38-363208,-122.720594
Received #210: 210-38-525579,-122.779243
Received #219: 219-38-28067,-122.668277
Received #206: 206-38-44807,-122.712694
Received #221: 221-38-456672,-122.732506
Received #205: 205-38-764642,-122.986894
Received #1344: 1344-37-799427,-122.2737808
Received #1538: 1538-37-7435188,-122.1475753
Received #1551: 1551-37-8160667,-122.2748794
Received #1207: 1207-37-7220153,-122.1605606
Received #1455: 1455-37-8039932,-122.2879714
Received #1556: 1556-37-9252967,-122.316864
Received #2110: 2110-37-7588653,-122.1863555
```

Heat Maps



Dependencies

The following tools and libraries are needed to run the application

Tools:

1. Kafka
2. Spark
3. MongoDB
4. Python 2.7
5. Flask

Libraries:

1. Flask==0.10.1
2. Flask_SocketIO
3. itsdangerous==0.24
4. Jinja2==2.7.3
5. MarkupSafe==0.23
6. python-engineio
7. python-socketio
8. six==1.9.0
9. Werkzeug==0.10.4
10. eventlet
11. confluent_kafka
12. pymongo
13. kafka
14. pykafka psycpg2
15. confluent_kafka
16. urllib2
17. pyspark
18. geohash

File Structure:

File Name	Location	Description
Runproducer.sh	\2017_spring_205_project	Starts kafka and the ingest script
Index.html	\2017_spring_205_project\portal\templates	HTML page for real time tracking
App.py	\2017_spring_205_project\portal	Flask app
Geohash.py	\2017_spring_205_project\portal	Implementation of geohash algorithm
Requirements.txt	\2017_spring_205_project\portal	Dependencies for running flask app
Architecture.docx	\2017_spring_205_project	Architecture doc
Geohash.py	\2017_spring_205_project	Implementation of geohash algorithm
Nextbus.py	\2017_spring_205_project	Python script to query NetBus data and push to kafka

Purgedb.py	\2017_spring_205_project	Python script to purge a specific hour of data to avoid duplication
README.md	\2017_spring_205_project	Deployment instruction
Runsparkaggregate.sh	\2017_spring_205_project	Script to run spark aggregate script every hour
Runsparkstreaming.sh	\2017_spring_205_project	Script to run spark streaming job
Spark_aggregate.py	\2017_spring_205_project	Python script for aggregation using spark
Spark_transform.py	\2017_spring_205_project	Python script for data transformation using spark

Steps to run application:

I could not figure out how to share the VM image with public on Azure. To get around, I have instantiated a VM from the image. None of the following steps are run on it.

Steps:

1. Get key file to connect to VM

wget https://raw.githubusercontent.com/stp8954/2017_spring_205_project/master/pem.key

2. Open 4 SSH connections to the VM using

ssh -i pem.key w205@13.88.20.241

3. Clone git repository

git clone https://github.com/stp8954/2017_spring_205_project.git

4. CD to the project dir in all 4 terminals

```
cd 2017_spring_205_project/
```

5. Make the shell script file executable

```
chmod 755 ./runsparkaggregate.sh
```

```
chmod 755 ./runsparkstreaming.sh
```

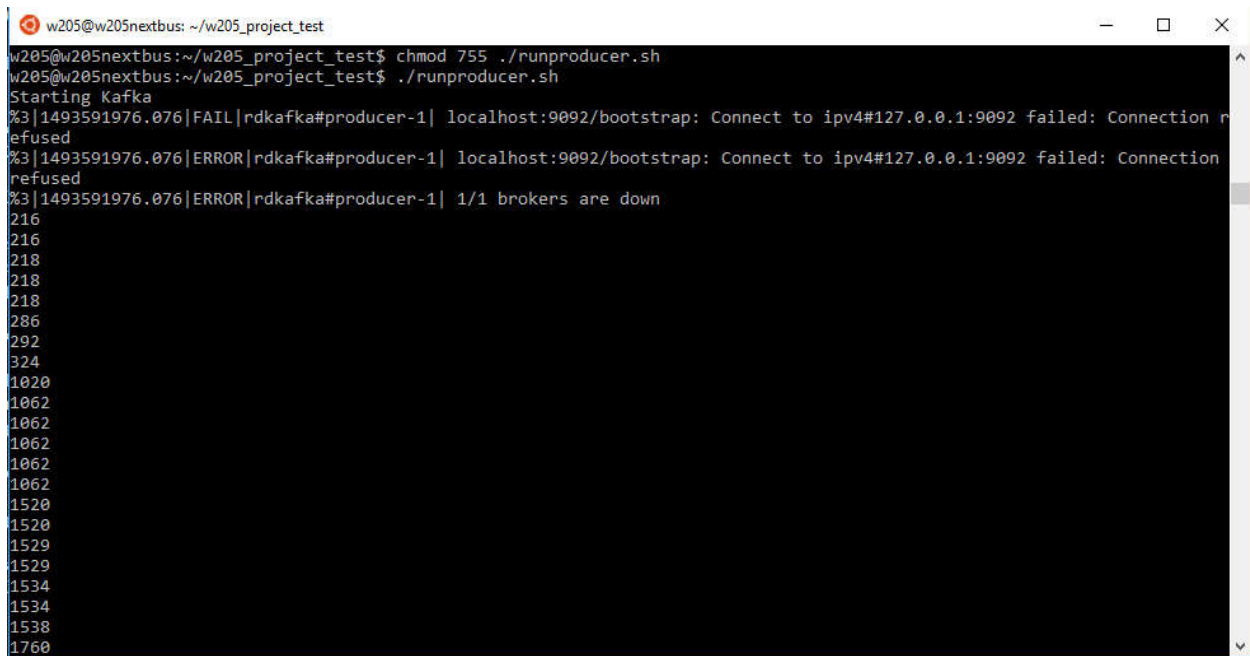
```
chmod 755 ./runproducer.sh
```

6. Run the following scripts in each terminal

```
./runproducer.sh
```

```
./runsparkstreaming.sh
```

```
./runsparkaggregate.sh
```

A terminal window titled 'w205@w205nextbus: ~/w205_project_test' with standard window controls. The terminal shows the execution of 'chmod 755 ./runproducer.sh' followed by './runproducer.sh'. The script attempts to start a Kafka producer, but fails with a 'Connection refused' error for the bootstrap server at localhost:9092. The error message is repeated three times, and the final status is '1/1 brokers are down'. The terminal output is as follows:

```
w205@w205nextbus:~/w205_project_test$ chmod 755 ./runproducer.sh
w205@w205nextbus:~/w205_project_test$ ./runproducer.sh
Starting Kafka
%3|1493591976.076|FAIL|rdkafka#producer-1| localhost:9092/bootstrap: Connect to ipv4#127.0.0.1:9092 failed: Connection r
efused
%3|1493591976.076|ERROR|rdkafka#producer-1| localhost:9092/bootstrap: Connect to ipv4#127.0.0.1:9092 failed: Connection
refused
%3|1493591976.076|ERROR|rdkafka#producer-1| 1/1 brokers are down
216
216
218
218
218
286
292
324
1020
1062
1062
1062
1062
1062
1520
1520
1529
1529
1534
1534
1538
1760
```



```
w205@w205nextbus: ~/w205_project_test
w205@w205nextbus:~/w205_project_test$ ./runsparkstreaming.sh
Ivy Default Cache set to: /home/w205/.ivy2/cache
The jars for the packages stored in: /home/w205/.ivy2/jars
:: loading settings :: url = jar:file:/usr/local/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.apache.spark#spark-streaming-kafka-0-8_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
  confs: [default]
    found org.apache.spark#spark-streaming-kafka-0-8_2.11;2.0.1 in central
    found org.apache.kafka#kafka_2.11;0.8.2.1 in central
    found org.scala-lang.modules#scala-xml_2.11;1.0.2 in central
    found com.yammer.metrics#metrics-core;2.2.0 in central
    found org.slf4j#slf4j-api;1.7.16 in central
    found org.scala-lang.modules#scala-parser-combinators_2.11;1.0.2 in central
    found com.101tec#zkclient;0.3 in central
    found log4j#log4j;1.2.17 in central
    found org.apache.kafka#kafka-clients;0.8.2.1 in central
    found net.jpountz.lz4#lz4;1.3.0 in central
    found org.xerial.snappy#snappy-java;1.1.2.6 in central
    found org.apache.spark#spark-tags_2.11;2.0.1 in central
    found org.scalatest#scalatest_2.11;2.2.6 in central
    found org.scala-lang#scala-reflect;2.11.8 in central
    found org.spark-project.spark#unused;1.0.0 in central
:: resolution report :: resolve 2375ms :: artifacts dl 18ms
:: modules in use:
  com.101tec#zkclient;0.3 from central in [default]
  com.yammer.metrics#metrics-core;2.2.0 from central in [default]
  log4j#log4j;1.2.17 from central in [default]
  net.jpountz.lz4#lz4;1.3.0 from central in [default]
  org.apache.kafka#kafka-clients;0.8.2.1 from central in [default]
  org.apache.kafka#kafka_2.11;0.8.2.1 from central in [default]
```

```
w205@deploytest2: ~/2017_spring_205_project
stp8954@deploymenttest2:~/2017_spring_205_project$ ./runsparkaggregate.sh
2017-05-01 02:00:00
2017-05-01 03:00:00
Ivy Default Cache set to: /home/stp8954/.ivy2/cache
The jars for the packages stored in: /home/stp8954/.ivy2/jars
:: loading settings :: url = jar:file:/usr/local/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.mongodb.spark#mongo-spark-connector_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
  confs: [default]
    found org.mongodb.spark#mongo-spark-connector_2.11;2.0.0 in central
    found org.mongodb#mongo-java-driver;3.2.2 in central
downloading https://repo1.maven.org/maven2/org/mongodb/spark/mongo-spark-connector_2.11/2.0.0/mongo-spark-connector_2.11-2.0.0.jar ...
  [SUCCESSFUL ] org.mongodb.spark#mongo-spark-connector_2.11;2.0.0!mongo-spark-connector_2.11.jar (51ms)
downloading https://repo1.maven.org/maven2/org/mongodb/mongo-java-driver/3.2.2/mongo-java-driver-3.2.2.jar ...
  [SUCCESSFUL ] org.mongodb#mongo-java-driver;3.2.2!mongo-java-driver.jar (36ms)
:: resolution report :: resolve 832ms :: artifacts dl 91ms
:: modules in use:
  org.mongodb#mongo-java-driver;3.2.2 from central in [default]
  org.mongodb.spark#mongo-spark-connector_2.11;2.0.0 from central in [default]
-----
|      conf      | number | search | dwnlded | evicted | artifacts |
|-----|-----|-----|-----|-----|-----|
|      default   |      2 |      2 |      2   |      0   |      2     |
|-----|-----|-----|-----|-----|-----|
:: retrieving :: org.apache.spark#spark-submit-parent
  confs: [default]
  2 artifacts copied, 0 already retrieved (2101kB/9ms)
python: can't open file '/home/w205/w205_project_test/spark aggregate.py': [Errno 2] No such file or directory
```

7. In the 4th terminal , run following commands

```
cd ~/2017_spring_205_project/portal
```

```
virtualenv api
```

```
source api/bin/activate
```

```
sudo pip install -r requirements.txt
```

```
sudo python app.py
```

8. Open browser on any machine and go to

For real time data

<http://13.88.20.241>

For heat maps . The streaming data needs to run for at least 1 hour to have any heat map data

If the streaming scripts are started at 5/02/2017 1:25PM PST , then at 2:30PM PST run open the following link

<http://13.88.20.241/heatmap?datetime=2017-05-02%2021:00:00>

Logic for the URL datetime={YYYY-MM-DD HH:00:00} -> Time is UTC