# Big Data Analysis of the 2016 Presidential Candidates

Berkeley MIDS W251 Final Project

***Dhaval Bhatt, James Gray, Tuhin Mahmud***

## 1. Introduction

This project uses various Big Data techniques to analyze characteristics of the 2016 Presidential candidates using batch and real-time data processing capabilities. We chose four candidates for our analysis: Donald Trump, Hillary Clinton, Ted Cruz and Bernie Sanders.

The entire Reddit corpus from October 2007 through August 2015 was used to evaluate various characteristics of the candidates including post volume over time, the most popular keywords, parts of speech that describe the candidates and sentiment. The intent was to evaluate the rise or decline of these candidates and to tease out insights. We also chose to process Twitter data to evaluate sentiment of Donald Trump in particular given the rise in interest as a candidate and competing views.

We selected the politics domain given the high volume of data related to the 2016 election and curiosity to explain how these candidates may differ.

Our goal for the project was to answer a few primary questions of four presidential candidates:

- Which Presidential candidates have gained interest over the last few years and within the last few weeks? This may give us insight into candidates where support and interest is increasing or decreasing.
- What is the sentiment for the Presidential candidates on social media (Twitter)? Is support improving or not?
- What are the most popular keywords and part of speech tags that characterize each candidate? This may give us insight into key themes or characteristics.

## 2. Architecture

The overall architecture is composed of batch and speed layer to enable the data analysis scenarios (lambda architecture). All code can be found on Github.

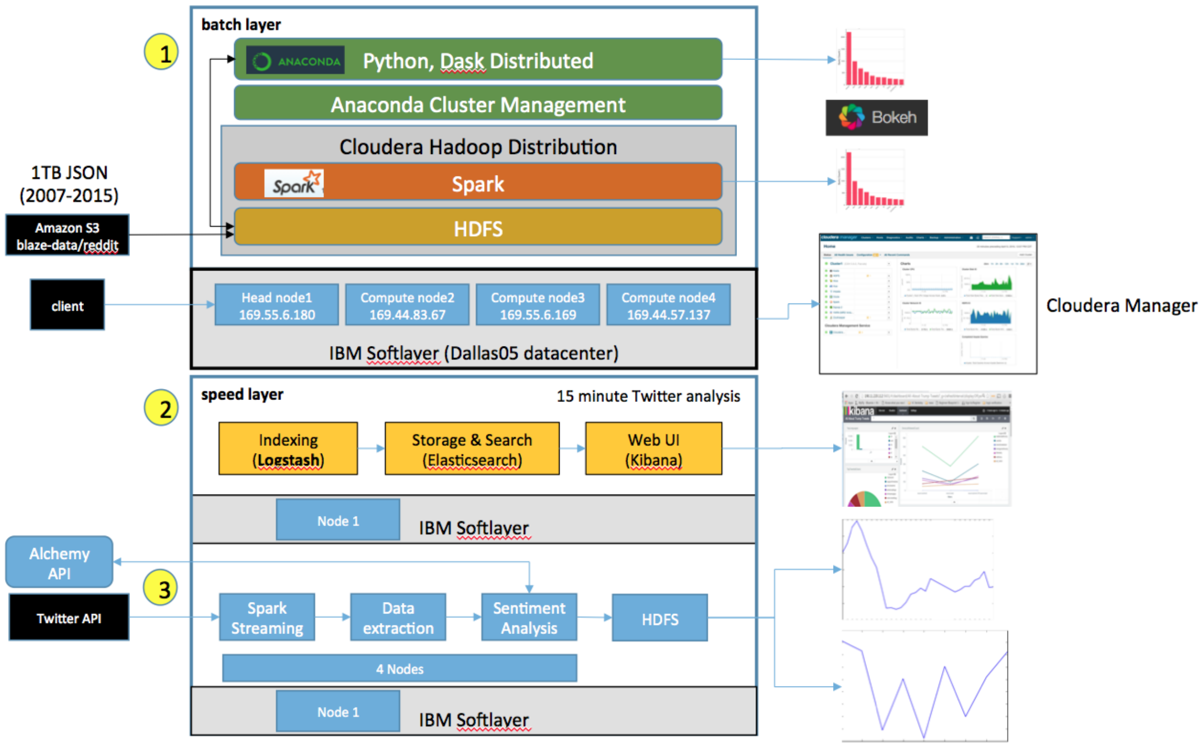


*Figure 1 - Architecture*

## Batch Layer

The batch layer is powered by a four-node IBM Softlayer cluster in the Dallas05 data center. It includes the following components:

- **IBM Softlayer -** 4 node cluster with one headnode and 3 workers. All nodes are configured the same: Ubuntu 14.04, 32GB RAM, 16 cores, 5.7 TiB HDFS
- **Cloudera Hadoop Distribution** - Hadoop components including HDFS, Spark, YARN were installed using the Cloudera Hadoop Distribution (Enterprise). CDH was installed on the headnode using the Cloudera Quickstart instructions. Cloudera Manager is the portal for managing the Hadoop services and is accessible at: http://169.55.6.180:7180/cmf/login using username: admin, password: midsw251.
- **Anaconda Cluster Management -** analyses were performed using pure Python and the Anaconda Cluster Management tool was used to install Python and Python packages on the distributed cluster. Packages were deployed from a MacBook. Additional configuration details are available in the Appendix.
- **Python libraries -** the following Python third-party libraries were used:
    - **hdfs3** - Python connector to HDFS
    - **dask, distributed** - a distributed processing framework on Hadoop using Python (see appendix for monitoring user interface)
    - **Bokeh** - interactive visualization
    - **Pyspark** - Spark programming interface

  - **NLTK** - natural language processing
  - **Matplotlib** - charting
- **Amazon S3** - the Reddit JSON files (1TB) were downloaded from the Amazon S3 object store onto the cluster. The data transfer completed in approximately two hours.
- **Dask Distributed Processing Framework** - Dask is a parallel processing framework using pure Python on top of the Hadoop stack. The Dask status interface is accessible at: http://169.55.6.180:8787/status
- **Jupyter Notebook** - a Jupyter notebook was used to manage the Python code and connect to the headnode for execution. The Jupyter server was installed and hosted on the headnode. The notebooks at accessible at: https://169.55.6.180:8889/tree (user = admin, password = midsw251)

## 3. Speed Layer

There are two data processing pipelines within the speed layer. The first is a pipeline to evaluate live Twitter data on dashboard for a day and the second is a 2 week sentiment analysis pipeline both analyze twitter related to different candidates to get current trends and sentiments.

The *dashboard speed laye*r includes following components

- **Elasticsearch -** creates the index of the tweeter feed and later used by kibana to display and analyze.
- **Logstash** - input configured for getting tweeter data continuously
- **Kibana -** generate display and analysis dashboard

The Live twitter data dashboard allows for analysis of any signal in user sentiment related to ongoing events and candidate popularity.

The ELK stack is installed in two different servers 50.97.205.234 and 198.11.220.112 , where each is dedicated to a candidate we are monitoring.

## 4. Batch Data Processing Scenarios

The objective of the batch data processing scenario was to generate insights about the presidential candidates by mining the 1TB Reddit JSON data. The insights were generated using Spark and a pure Python approach (Dask) to compare and contrast performance and code implementation. The Jupyter notebook can be found on Github and the Juypter notebook server dashboard is accessible at https://169.55.6.180:8889/ (username: admin, password = midsw251).

### Data Transfer from Amazon S3

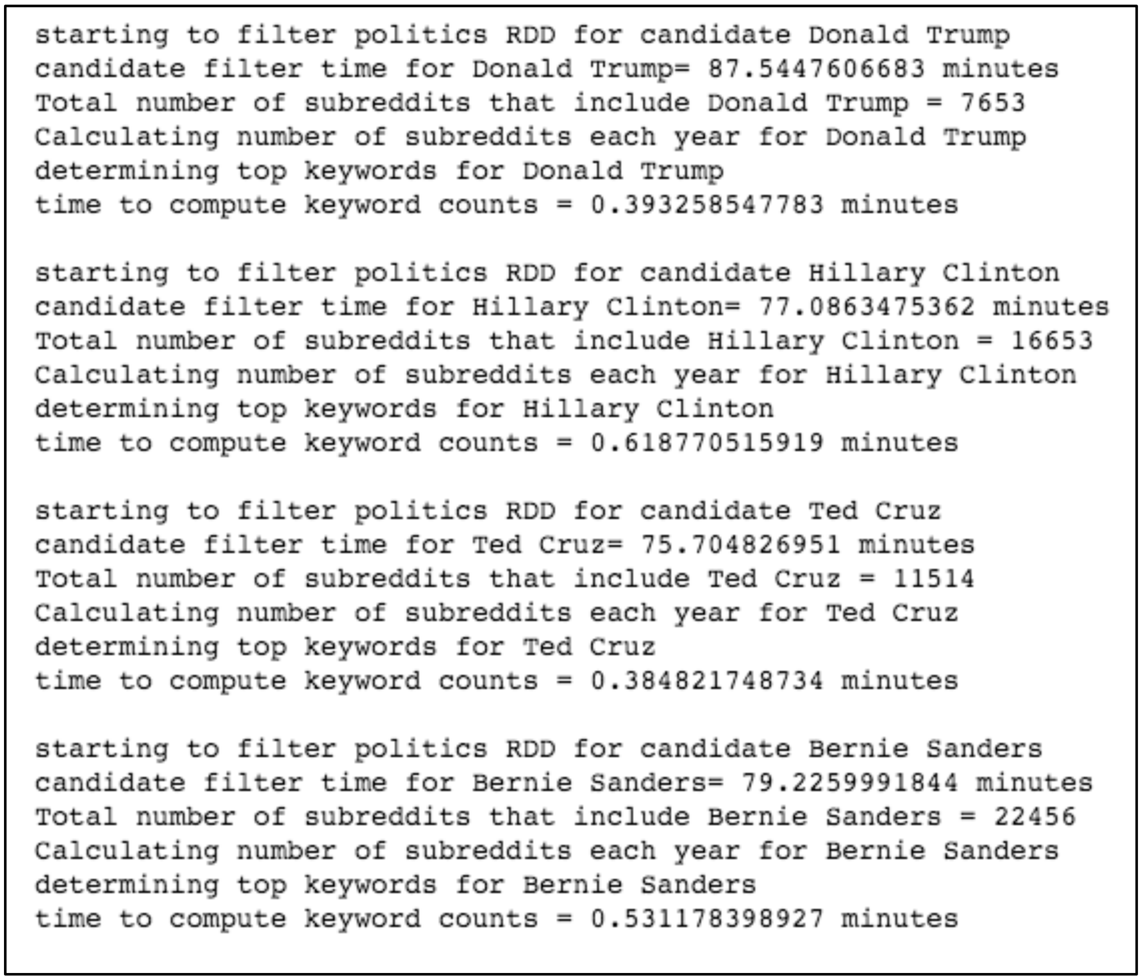
The 1TB of Reddit JSON data was downloaded from Amazon S3 to the Softlayer cluster using the following command:

```
hadoop distcp s3n://AWS_ACCESS:AWS_SECRET@blaze-data/reddit/json/*/*.json /user/root
```

The data set is composed of one JSON file for each month. They are stored in the /user/root folder on HDFS (See appendix)

## JSON Data Processing

The approach for data processing for both Spark and Dask were essentially identical. The entire 1TB data set was first loaded into a Spark RDD and Dask Bag respectively. The objects were then reduced to the subreddit "politics" so that the data analysis was focused on the political domain. These objects were persisted and then further reduced by each presidential candidate during the analysis sections described below. We found that RDD transformations (filtering) and actions (e.g. counts) consumed substantial memory. For example, the analyses below required more than 1-2 hours to process the entire 1TB data set. We experimented with persisting the RDD's but these created memory errors.



```
starting to filter politics RDD for candidate Donald Trump
candidate filter time for Donald Trump= 87.5447606683 minutes
Total number of subreddits that include Donald Trump = 7653
Calculating number of subreddits each year for Donald Trump
determining top keywords for Donald Trump
time to compute keyword counts = 0.393258547783 minutes

starting to filter politics RDD for candidate Hillary Clinton
candidate filter time for Hillary Clinton= 77.0863475362 minutes
Total number of subreddits that include Hillary Clinton = 16653
Calculating number of subreddits each year for Hillary Clinton
determining top keywords for Hillary Clinton
time to compute keyword counts = 0.618770515919 minutes

starting to filter politics RDD for candidate Ted Cruz
candidate filter time for Ted Cruz= 75.704826951 minutes
Total number of subreddits that include Ted Cruz = 11514
Calculating number of subreddits each year for Ted Cruz
determining top keywords for Ted Cruz
time to compute keyword counts = 0.384821748734 minutes

starting to filter politics RDD for candidate Bernie Sanders
candidate filter time for Bernie Sanders= 79.2259991844 minutes
Total number of subreddits that include Bernie Sanders = 22456
Calculating number of subreddits each year for Bernie Sanders
determining top keywords for Bernie Sanders
time to compute keyword counts = 0.531178398927 minutes
```

*Figure 2 - Data Processing Performance*

We concluded that increasing the cluster size, increasing RAM, pre-processing the JSON files and persisting the RDD's and Dask Bag objects in memory would have improved performance considerably. We used the Dask user interface to monitor performance and graph execution for gaining insight into optimizations.

## Volume Analysis

We were curious to explore the Reddit data to see how the volume of each presidential candidate trended over the last 7-8 years. Figure 3 displays the number of posts that included the candidate name in the post body for each year from 2007 - 2015. Note that 2007 (October-December) and 2015 (January - August) contained posts for only part of the year.

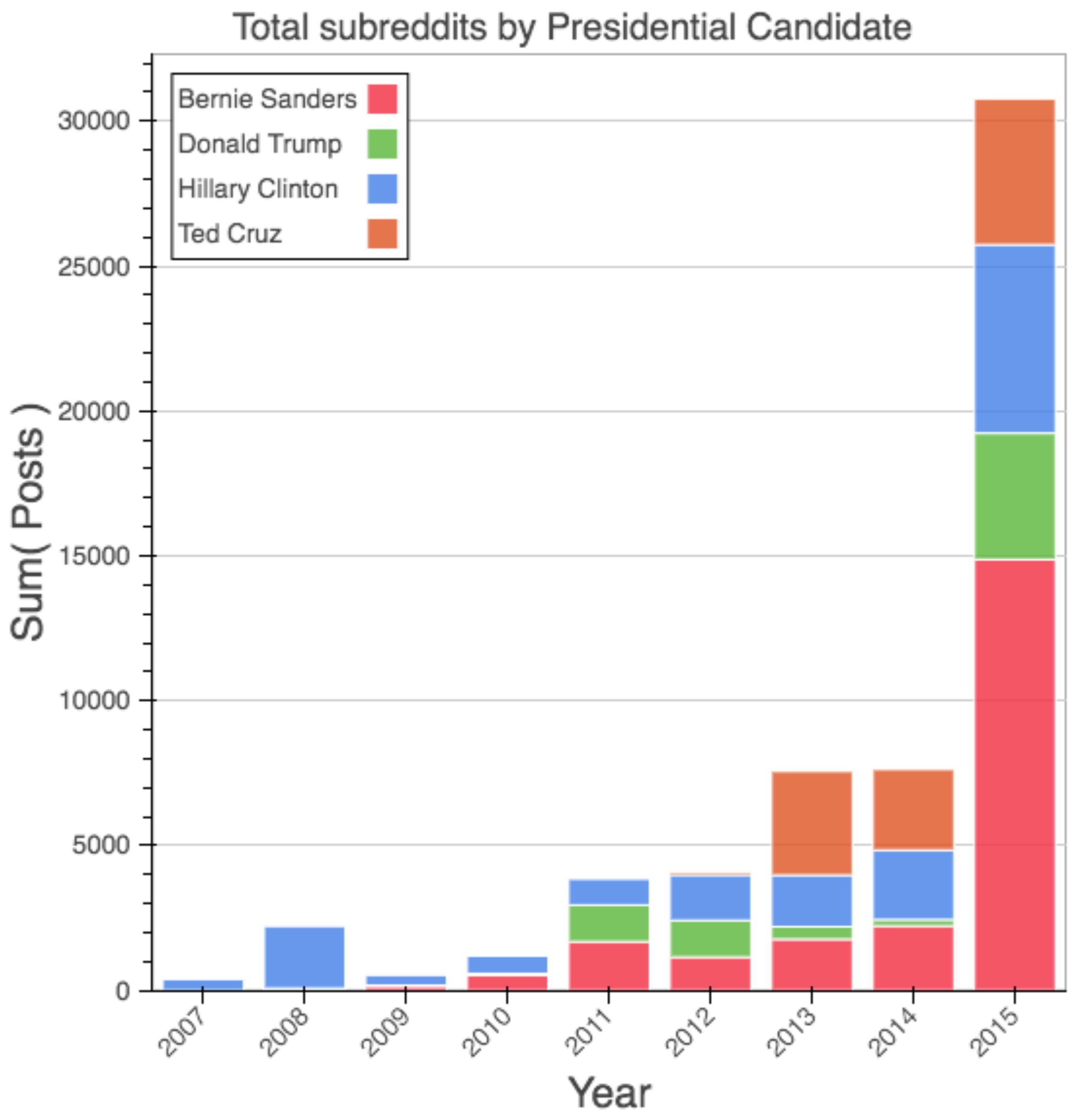
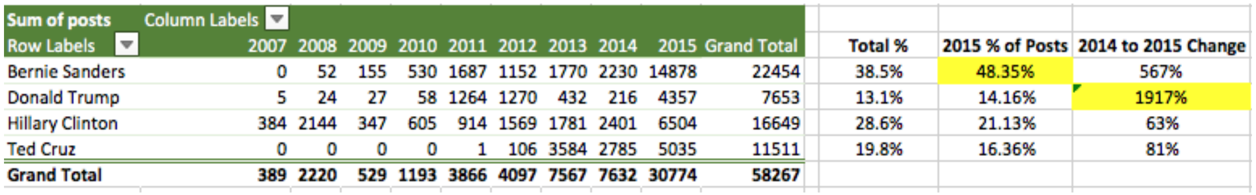


*Figure 3 - Subreddit volume for each candidate (Oct 2007 - Aug 2015)*



| Sum of posts | Column Labels | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row Labels | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 | Grand Total | Total % | 2015 % of Posts | 2014 to 2015 Change |
| Bernie Sanders | 0 | 52 | 155 | 530 | 1687 | 1152 | 1770 | 2230 | 14878 | 22454 | 38.5% | 48.35% | 567% |
| Donald Trump | 5 | 24 | 27 | 58 | 1264 | 1270 | 432 | 216 | 4357 | 7653 | 13.1% | 14.16% | 1917% |
| Hillary Clinton | 384 | 2144 | 347 | 605 | 914 | 1569 | 1781 | 2401 | 6504 | 16649 | 28.6% | 21.13% | 63% |
| Ted Cruz | 0 | 0 | 0 | 0 | 1 | 106 | 3584 | 2785 | 5035 | 11511 | 19.8% | 16.36% | 81% |
| **Grand Total** | **389** | **2220** | **529** | **1193** | **3866** | **4097** | **7567** | **7632** | **30774** | **58267** | | | |

*Figure 4 - Year over Year Post Volume*

Overall the volume of posts increased significantly leading into the 2016 election year. The number of posts including Hillary Clinton has remained fairly constant over the last few years while Donald Trump and Bernie Sanders surged 1917% and 567% respectively from 2014 to 2015. These data suggest a growing interest in both of the candidates when compared to Hillary Clinton and Ted Cruz. Somewhat surprising is that Bernie Sanders was referenced in 48% of the 2015 posts and over 38% of the posts from 2007-2015. The 2015 surge suggests a growing interest as a viable presidential candidate (blue line). Ted Cruz appeared on Reddit radar starting in 2012 but at a slower rise when compared to Sanders, Clinton and Trump.

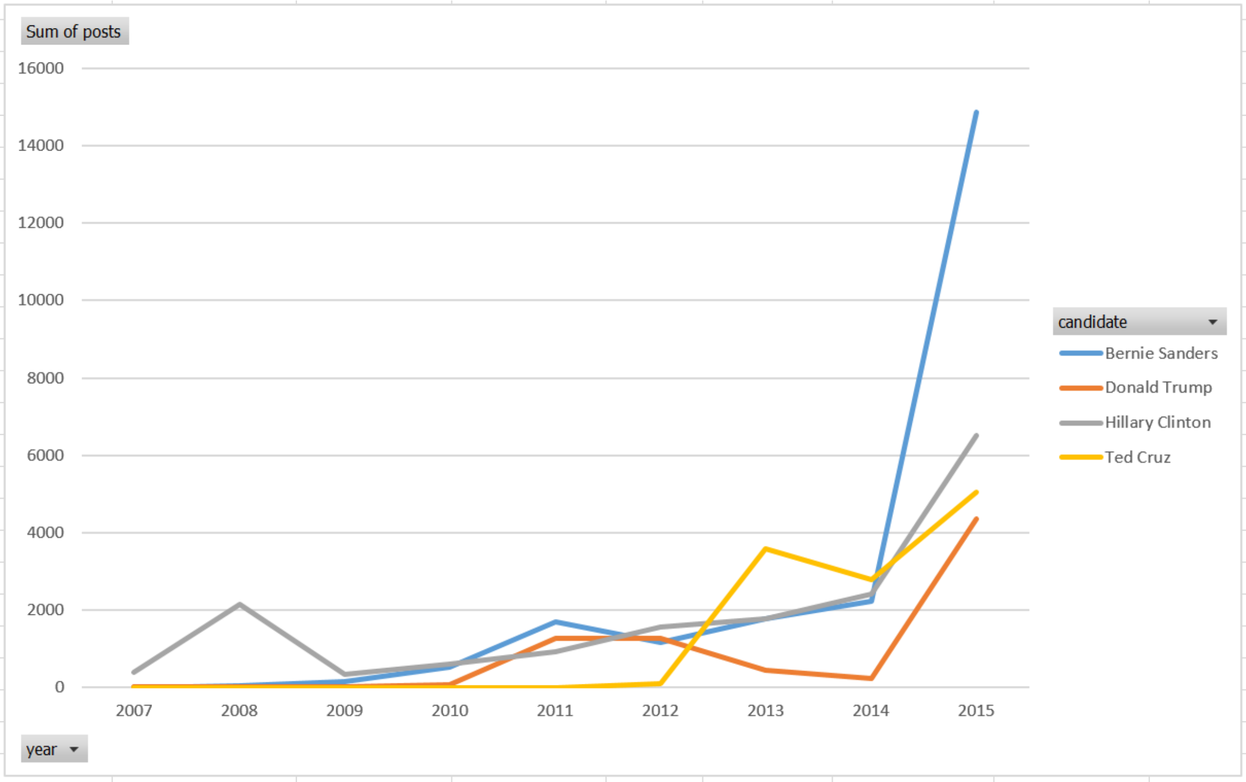
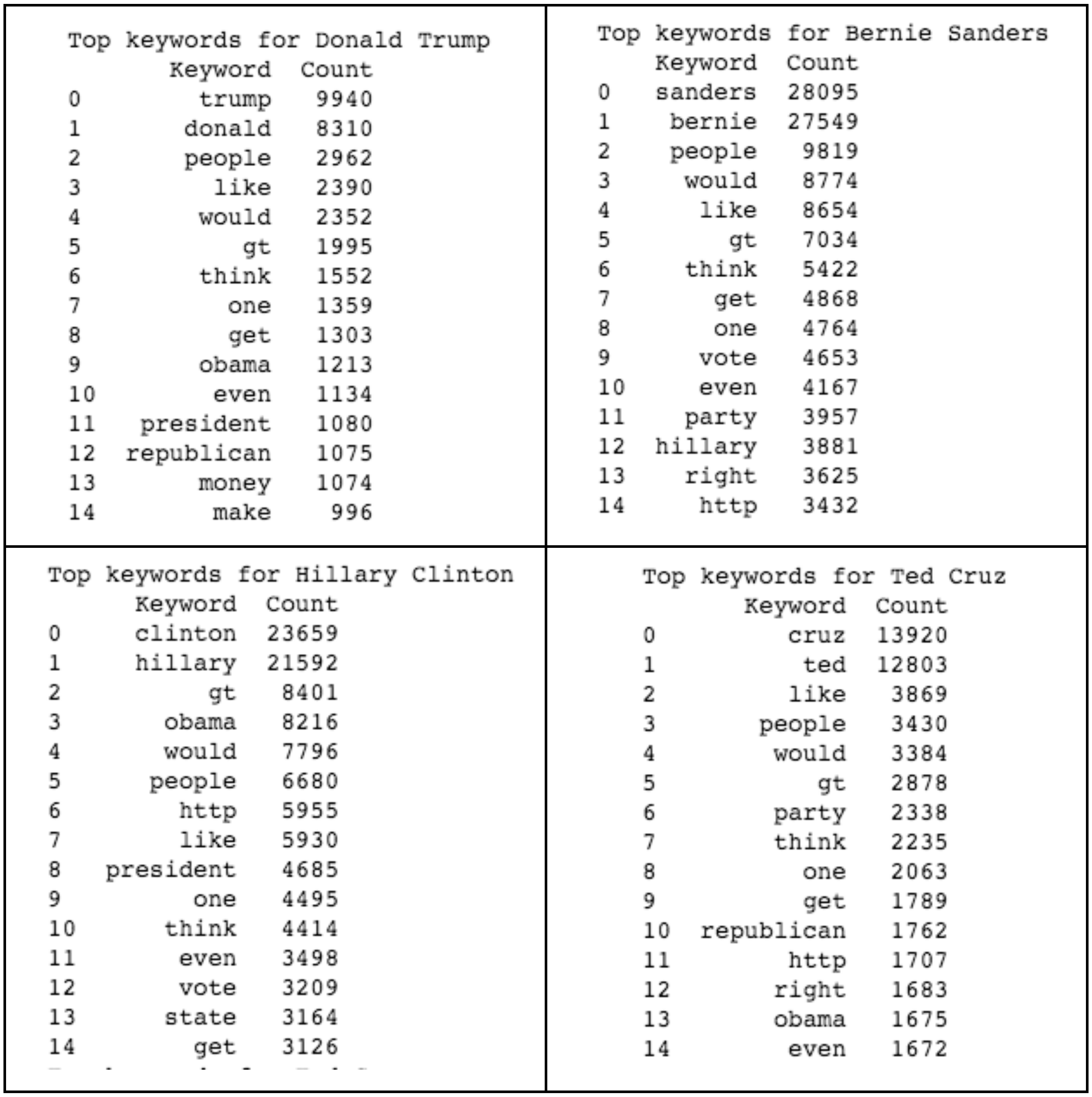


*Figure 5 - Reddit Post Volume*

## Top Keyword Analysis

Analyzing the top keywords within the posts that referenced each presidential candidate may provide some insights into themes or characteristics of the candidates. The "body" attribute of the JSON was processed using the Natural Language Toolkit (NLTK) Python library to tokenize the words and calculate the frequencies. The processing produced the top 15 keywords for each candidate:



```
Top keywords for Donald Trump
       Keyword  Count
0        trump   9940
1       donald   8310
2       people   2962
3         like   2390
4        would   2352
5           gt   1995
6        think   1552
7          one   1359
8          get   1303
9        obama   1213
10        even   1134
11   president   1080
12  republican   1075
13       money   1074
14        make    996
```

```
Top keywords for Bernie Sanders
    Keyword  Count
0   sanders  28095
1    bernie  27549
2    people   9819
3     would   8774
4      like   8654
5        gt   7034
6     think   5422
7       get   4868
8       one   4764
9      vote   4653
10     even   4167
11    party   3957
12  hillary   3881
13    right   3625
14     http   3432
```

```
Top keywords for Hillary Clinton
      Keyword  Count
0     clinton  23659
1     hillary  21592
2          gt   8401
3       obama   8216
4       would   7796
5      people   6680
6        http   5955
7        like   5930
8   president   4685
9         one   4495
10      think   4414
11       even   3498
12       vote   3209
13      state   3164
14        get   3126
```

```
Top keywords for Ted Cruz
       Keyword  Count
0         cruz  13920
1          ted  12803
2         like   3869
3       people   3430
4        would   3384
5           gt   2878
6        party   2338
7        think   2235
8          one   2063
9          get   1789
10  republican   1762
11        http   1707
12       right   1683
13       obama   1675
14        even   1672
```

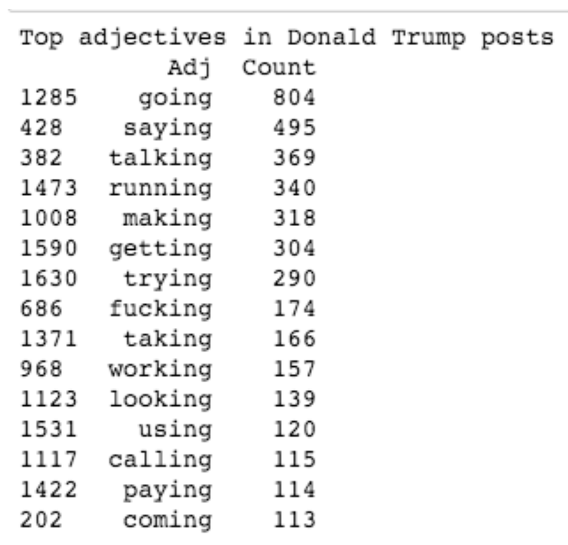
Figure 6 – Keyword Analysis

Overall the analysis did not reveal deep insights given that many common words showed up in the top 15 other than the candidates. It was not surprising that "money" was a top keyword for

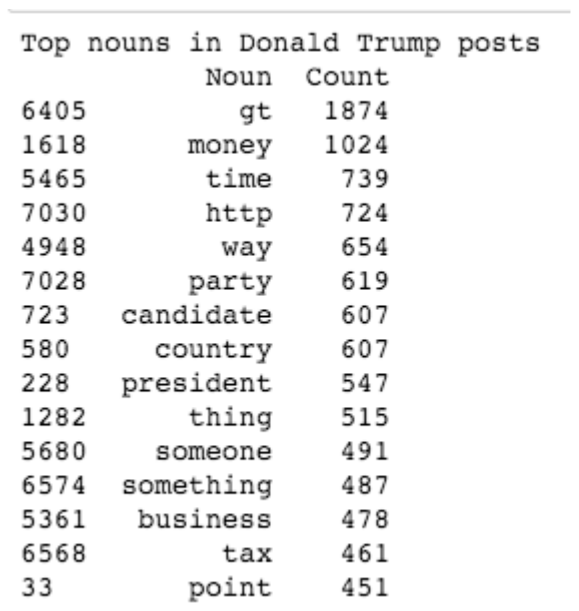
Trump while we found it interesting that “Obama” was a keyword for all candidates except Sanders. Extending the analysis for more keywords may have provided additional insights.

## Part of Speech Analysis

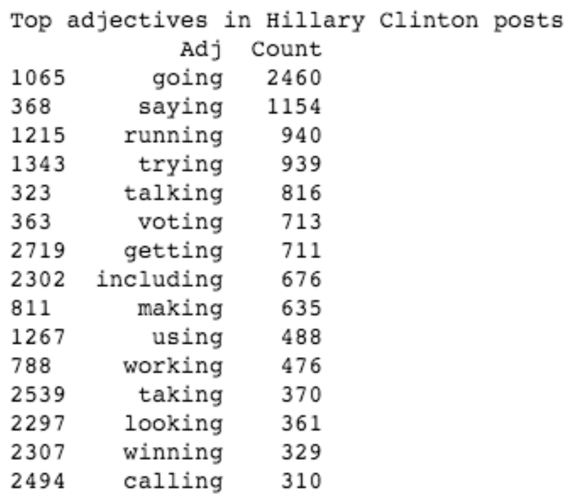
We also used the NLTK to mine the parts of speech including nouns and adjectives to give us further insight into themes or characteristics for each candidate. The part of speech processing produced the the top 15 nouns and adjectives for each candidate:



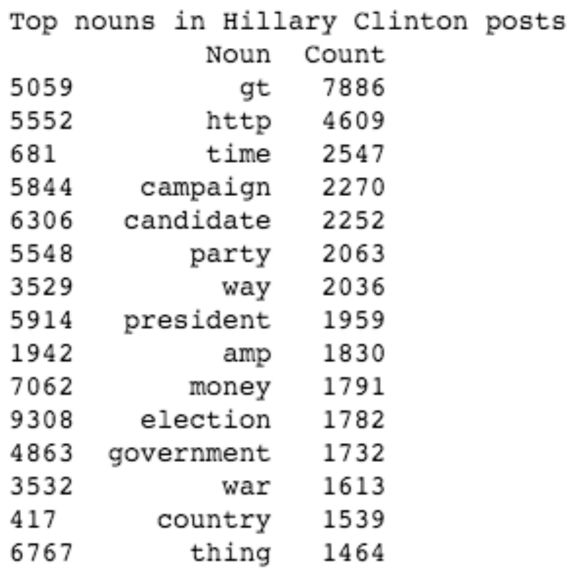
```
Top adjectives in Donald Trump posts
           Adj  Count
1285     going    804
428     saying    495
382    talking    369
1473   running    340
1008    making    318
1590   getting    304
1630    trying    290
686    fucking    174
1371    taking    166
968    working    157
1123   looking    139
1531     using    120
1117   calling    115
1422    paying    114
202     coming    113
```



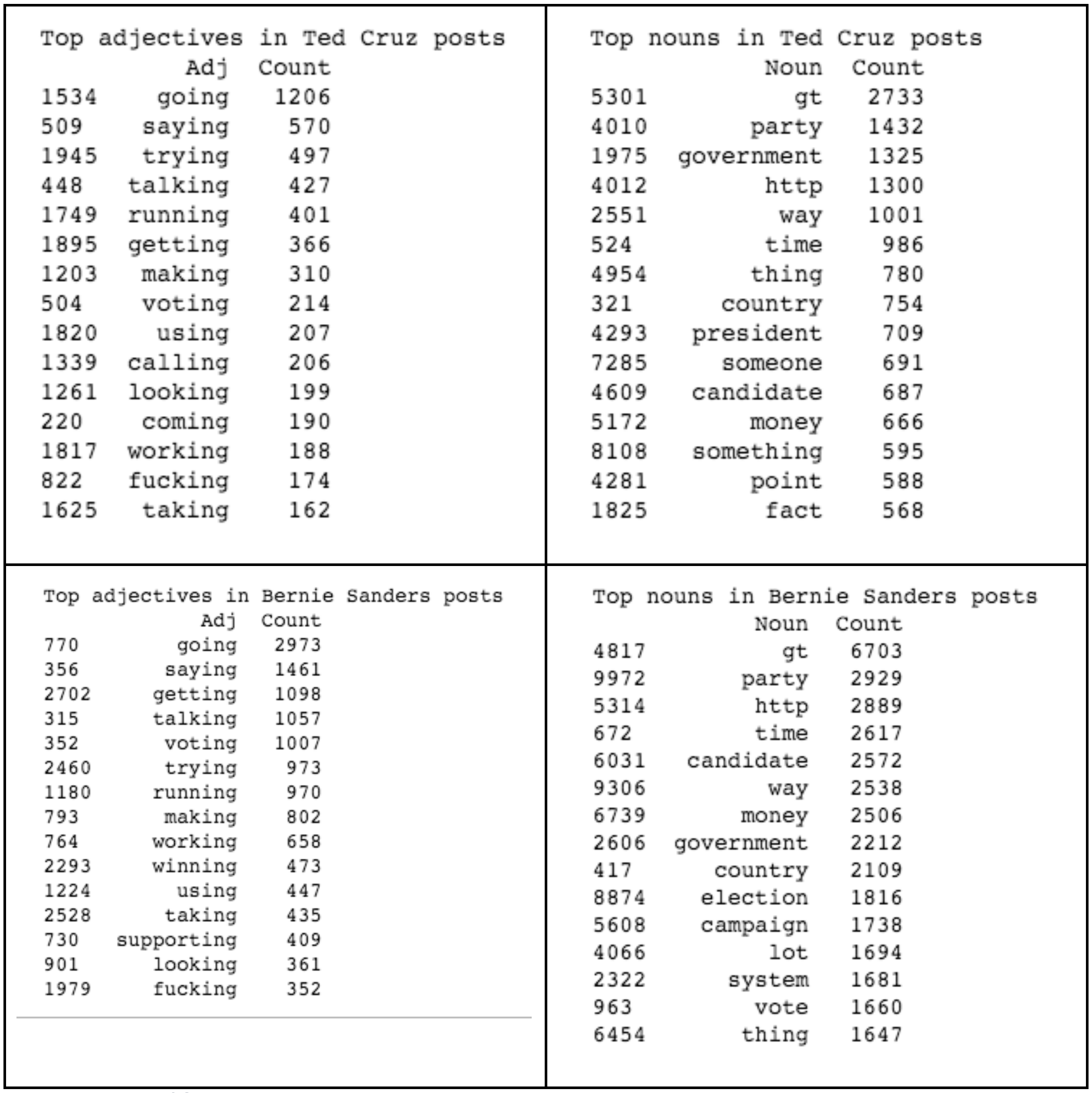
```
Top nouns in Donald Trump posts
             Noun  Count
6405           gt   1874
1618        money   1024
5465         time    739
7030         http    724
4948          way    654
7028        party    619
723     candidate    607
580       country    607
228     president    547
1282        thing    515
5680      someone    491
6574    something    487
5361     business    478
6568          tax    461
33          point    451
```



```
Top adjectives in Hillary Clinton posts
             Adj  Count
1065       going   2460
368       saying   1154
1215     running    940
1343      trying    939
323      talking    816
363       voting    713
2719     getting    711
2302   including    676
811       making    635
1267       using    488
788      working    476
2539      taking    370
2297     looking    361
2307     winning    329
2494     calling    310
```



```
Top nouns in Hillary Clinton posts
              Noun  Count
5059            gt   7886
5552          http   4609
681           time   2547
5844      campaign   2270
6306     candidate   2252
5548         party   2063
3529           way   2036
5914     president   1959
1942           amp   1830
7062         money   1791
9308      election   1782
4863    government   1732
3532           war   1613
417        country   1539
6767         thing   1464
```



```
Top adjectives in Ted Cruz posts
          Adj  Count
1534    going   1206
509    saying    570
1945   trying    497
448   talking    427
1749  running    401
1895  getting    366
1203   making    310
504    voting    214
1820    using    207
1339  calling    206
1261  looking    199
220    coming    190
1817  working    188
822   fucking    174
1625   taking    162
```

```
Top nouns in Ted Cruz posts
             Noun  Count
5301           gt   2733
4010        party   1432
1975   government   1325
4012         http   1300
2551          way   1001
524          time    986
4954        thing    780
321       country    754
4293    president    709
7285      someone    691
4609    candidate    687
5172        money    666
8108    something    595
4281        point    588
1825         fact    568
```

```
Top adjectives in Bernie Sanders posts
             Adj  Count
770        going   2973
356       saying   1461
2702     getting   1098
315      talking   1057
352       voting   1007
2460      trying    973
1180     running    970
793       making    802
764      working    658
2293     winning    473
1224       using    447
2528      taking    435
730   supporting    409
901      looking    361
1979     fucking    352
```

```
Top nouns in Bernie Sanders posts
            Noun  Count
4817          gt   6703
9972       party   2929
5314        http   2889
672         time   2617
6031   candidate   2572
9306         way   2538
6739       money   2506
2606  government   2212
417      country   2109
8874    election   1816
5608    campaign   1738
4066         lot   1694
2322      system   1681
963         vote   1660
6454       thing   1647
```

*Figure 7 - Part of Speech Analysis*

- The nouns for Donald Trump suggest a focus on business, money and taxes. Profanity may suggest a strong opposition from the general public.
- The nouns for Hillary Clinton suggest that war is a theme given her involvement in recent wars including Libya, Iraq and the Taliban. The analysis of adjectives did not generate any conclusive insights other than a “running” theme of whether she was going to run for the presidency.
- Profanity appeared in posts for all candidates except Hillary Clinton.

The part of speech analysis delivered limited insights and we concluded that analysis of more than 15 keywords is likely required as well as filtering common words that appeared across the candidates (e.g., candidate, government, election).

# 5. Real-Time Analytics

One of the speed layer implementation we did was to capture the snapshot of the current data on tweeters for two candidates (Hillary and Donald Trump) . We implemented the real time visualization of tweeter data using ELK stack consisting of the following

1. ElasticSearch (elasticsearch-2.3.1)
   Elasticsearch is a distributed open source search engine based on Apache Lucene, and released under an Apache 2.0 license. It provides horizontal scalability, reliability, and multitenant capability for real-time search.

2. Logstash (logstash-1.5.4 )
   Logstash is a data pipeline that helps collect, parse, and analyze a large variety of structured and unstructured data and events generated across various systems. It provides plugins to connect to various types of input sources and platforms, and is designed to efficiently process logs, events, and unstructured data sources for distribution into a variety of outputs with the use of its output plugins.

3. Kibana (kibana-4.1.1)
   Kibana is an open source Apache 2.0 licensed data visualization platform that helps in visualizing any kind of structured and unstructured data stored in Elasticsearch indexes.

## Setup

We put together two Softlayer servers,. each with 16G memory and 100G disk space and installed the ELK stack on both. This allowed us to monitor the two candidates separately and analyze the data independently.

### Logstash configuration

Logstash was configured to capture and filter tweets for a candidate. Logstash is than run using the configuration file to get the input data from tweeter and the output is directed to the elasticsearch.

```
input {
  twitter {
    consumer_key =>  "XXXXXX"
    consumer_secret => "XXXXX"
    oauth_token =>  "XXXXXXX"
    oauth_token_secret =>  "XXXXXXX"
    keywords => ["#hillary"]
    full_tweet => "true"
  }
}
output {
  elasticsearch {
    protocol => "http"
    host => "localhost"
    port => "9200"
    index => "twitter"
    document_type => "realtime"
  }
}
```

## Elasticsearch Configuration

Elasticsearch is configured to run on port 9200 and accept the logstash file for indexing.

## Kibana Setup and Dashboard creation

We created following visualization for each candidate on each server

1. Popular Hashtags (Top 5)
2. Top Tweeted User (Top 5)
3. Metrics
    a. Tweeter Count
    b. Unique count of entities.hashtags
    c. Unique user.screen_name
    d. Unique count of user.time zone

Visualization on the dashboard captures the most popular hastags related to a candidate on that day. For example, tweets during New York Primary day on April 19th,2016 had following top 5 hashtags *#trump2016, #nyvalues, #nyprimary, #primaryday, #ny* and for hillary clinton it included
#transcript, #vote, #nyprimary #bern,#imwithher

## ELK Data Pipeline

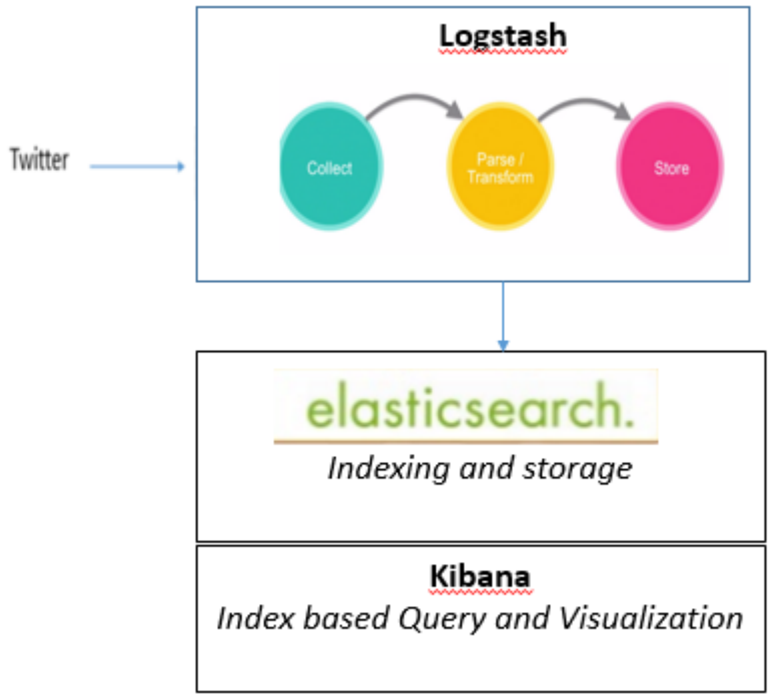
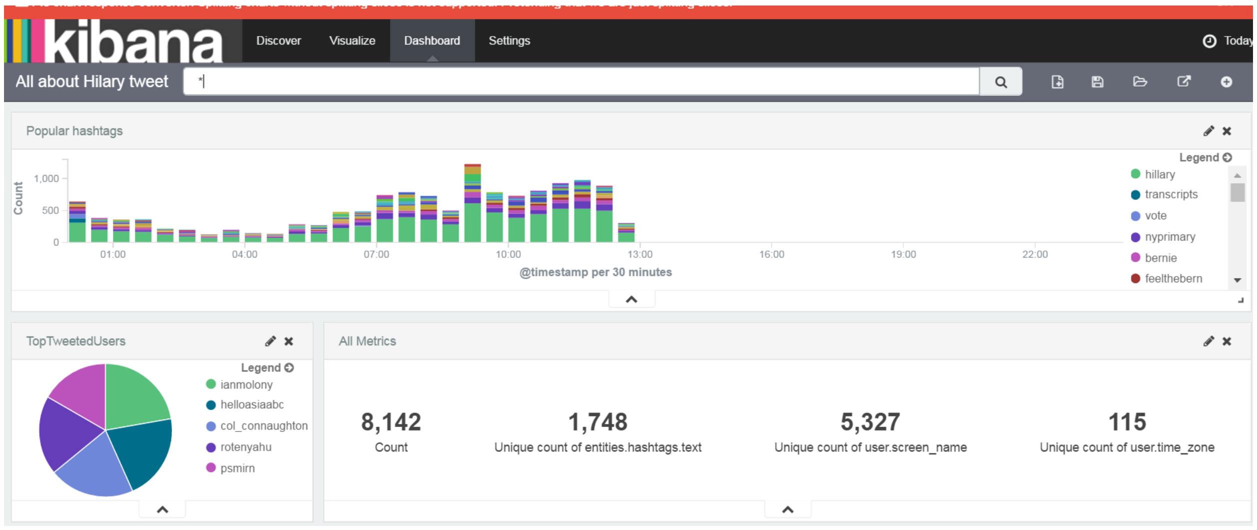


*Figure 8 – ELK Data pipeline*

In a typical ELK stack data pipeline, multiple application server outputs are shipped through logstash shipper to a centralized logstash indexer. In our example we used twitter as input to the logstash and set up the logstash and indexer on the same server. Logstash indexer outputs the data to elasticsearch cluster on the same server machine and this is queried by the Kibana to display visualizations and build dashboards.

### Visualization Dashboards

We implemented the two dashboards, one for Donald Trump and one for Hillary Clinton which depicts the current trends about the candidate on tweeterverse. The time line can be configured for 1 day, 1 hour or 15 minutes depending on where we want to narrow our focus for analysis.



8,142 Count
1,748 Unique count of entities.hashtags.text
5,327 Unique count of user.screen_name
115 Unique count of user.time_zone

*Figure 9 – Kibana Dashboard*

Dashboard for Hillary Clinton

1. http://50.97.205.234:5601/#/dashboard/All-about-Hilary-tweet

Dashboard of Donald Trump

2. http://198.11.220.112:5601/#/dashboard/All-About-Trump-Tweets

The same concept can be extended to monitor any number of candidates

# 6. Twitter Sentiment Analysis

## Overview

In this part of the project, we use Spark, Hadoop, S3 to find out various aggregations about the tweets posted about 'donald trump'.

We use spark to collect and process tweets and store the output to locally attached HDFS disks. We periodically move the output files to S3. Finally, we use this data from S3 to create various visualization and for further aggregation using python.

Here, we use the above technologies to search twitter for tweets about 'donald trump'. Using these tweets, we find out the top 10 keywords used in these tweets every hour. We also find out the place each tweet corresponds and we aggregate by the places. We also create a plot of the number of tweets gathered every hour.

## Cluster setup

The cluster was setup using 4 Softlayer virtual servers, each with 16GB of RAM, 4 vCPUs and two disks of 100GB each. One disk each of the 4 nodes was formatted to HDFS. Then spark cluster was created on top of this Hadoop installation. Stock sources were used and spark and Hadoop were installed separately.

## SPARK Implementation

Spark was installed on top of Hadoop and Hadoop environment variables were created and supplied to spark. This way the default output location of the spark program was set to the HDFS disk. The spark program was created using Scala.

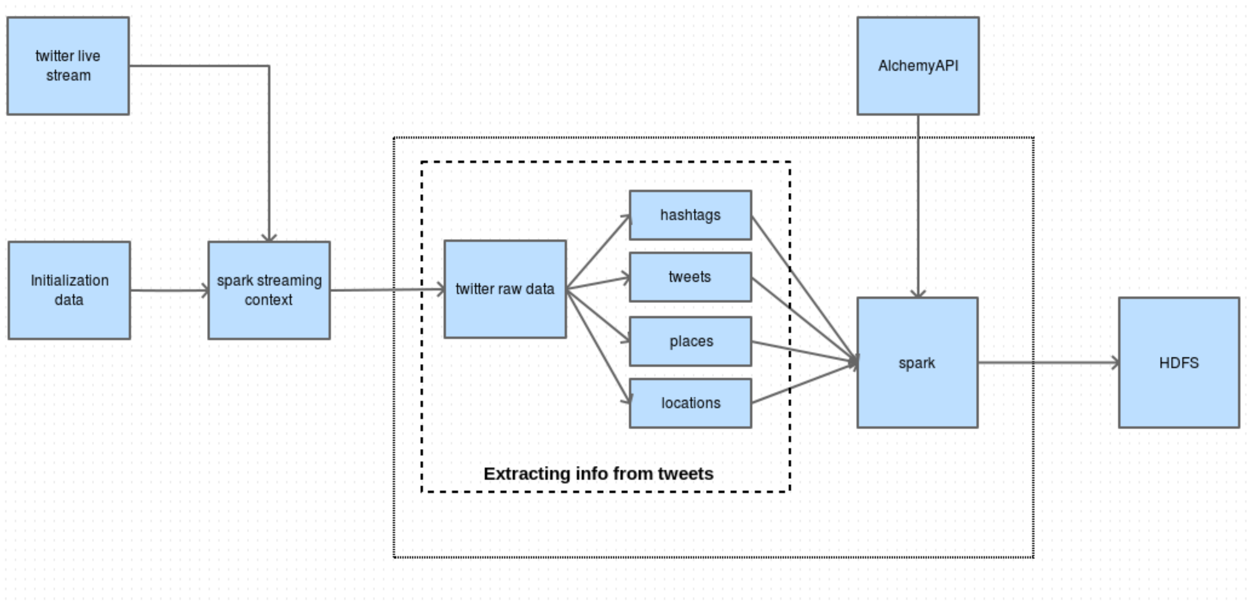


*Figure 10 – Sentiment analysis pipeline*

For removing stopwords, we use a custom english stopword list taken from http://www.ranks.nl/stopwords/. We add this list to HDFS so that it's visible by all the nodes.

Using batch size of 1 seconds, that is, tweets are gathered every second, and sliding window duration of 1 hour and analysis window duration of 1 hour, we perform our analysis.

Since the analysis window duration and sliding window duration are same, spark does not use any data from the previous analysis window.

Since we are dumping data every hour and not using it in memory, this saves RAM and we can use the aggregated data in a different program for further analysis.

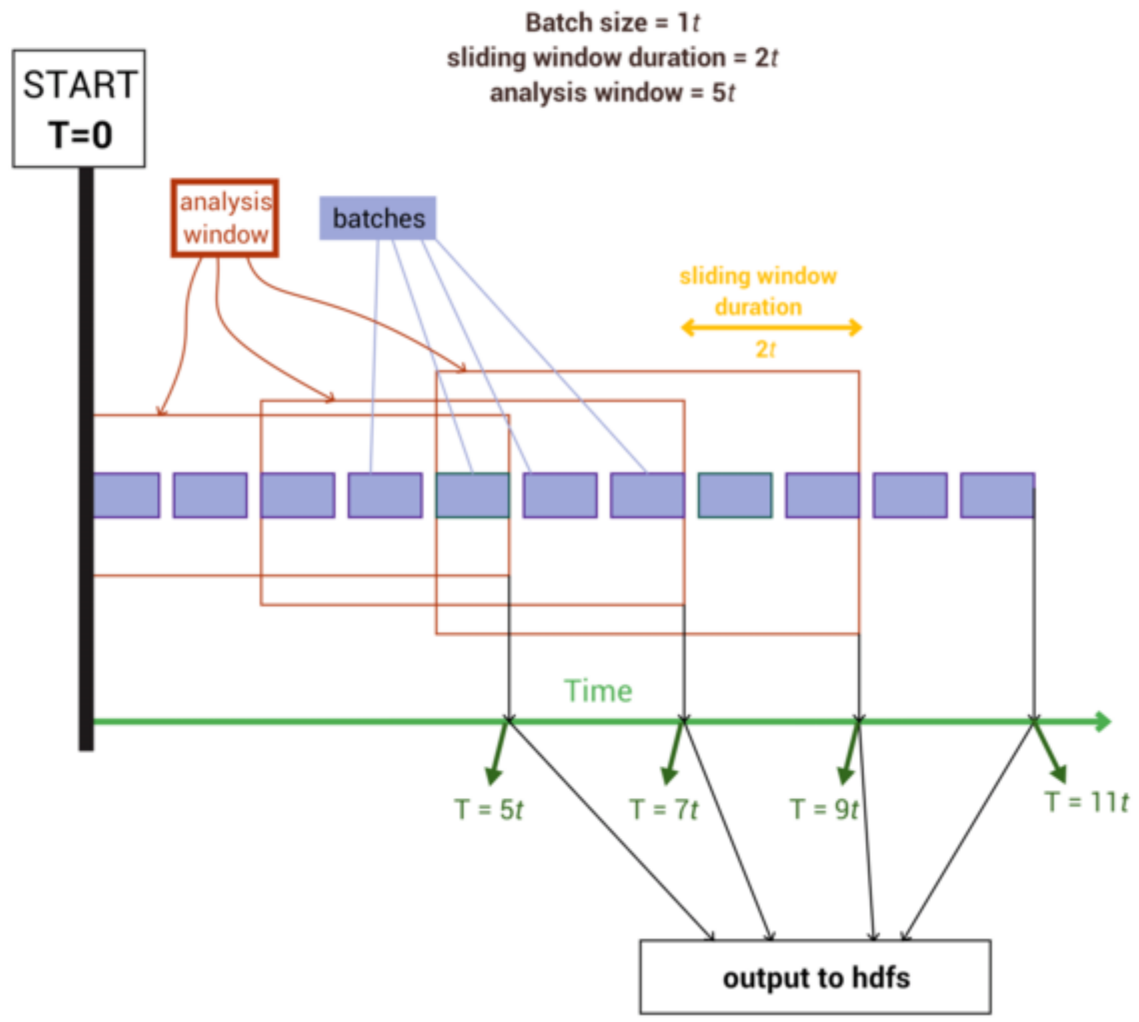


*Figure 11 – Spark streaming*

For collecting tweets, we use the built-in twitter4j library. The library provides various functions to extract fields from the JSON returned by the twitter streaming API. Here we extract tweet text, hashtags, keywords, and the places the tweets correspond to. Keywords are the words in the tweet text which are not in the stopwords list.

After the fields are extracted, we perform aggregation on each field and dump the results to HDFS. In our case, we have performed simple counting aggregation.

After the data was dumped, we use alchemyAPI to perform sentiment analysis on top 40 tweets obtained in the current analysis window. We also find the number of unique tweets, hashtags etc. We write all this output to stdout.

A separate bash script is used to run the program and provide various parameters to the program. IO redirection is used to append out the output from stdout to local file 'count_and_sentiments.txt' and write the error and info from stderr to another local file 'error.log' We use the 'error.log' file to debug any errors encountered during the execution of the script.

The HDFS files are uploaded to s3 using distcp command in Hadoop. And the local file is uploaded is using the AWS CLI.

## Python Implementation

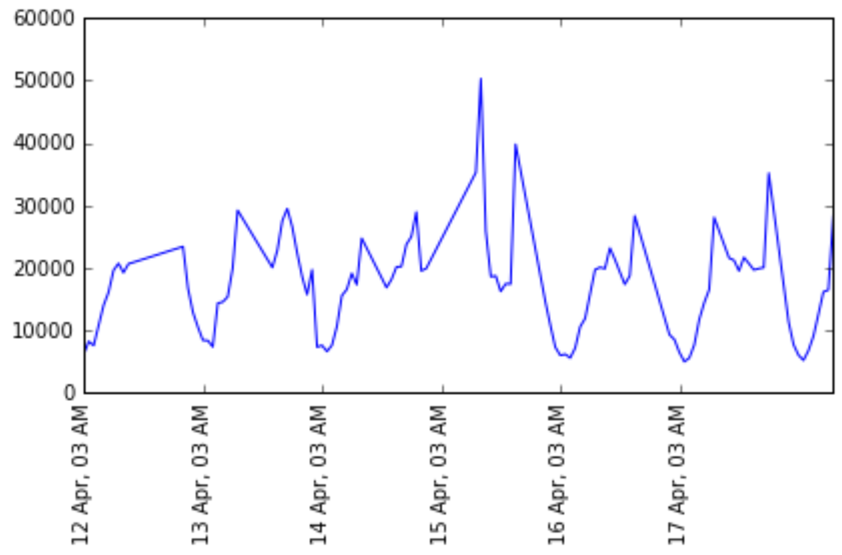
IPython notebook is used for creating time-series plots and to find out top keywords from the output of spark program.

For this python program, we used the uploaded S3 data as input.

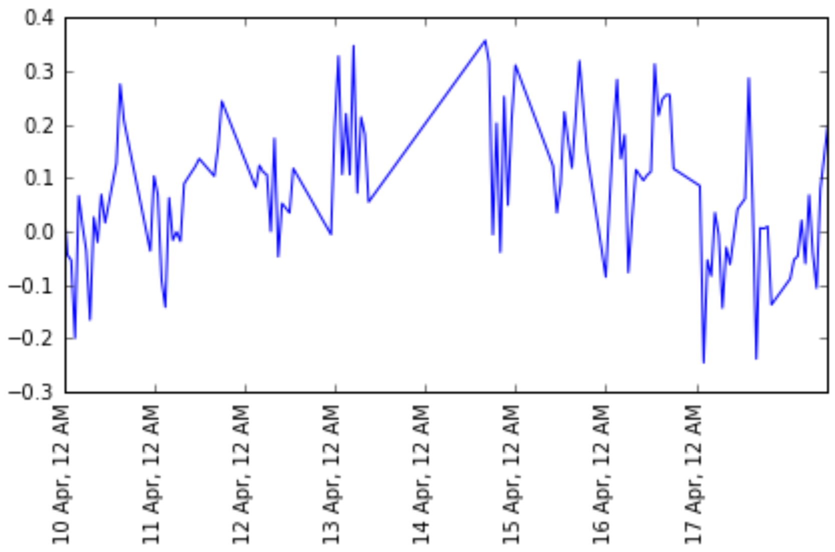
Using boto, we gather all keyword files from S3 and group them by the dates and print out the top 10 keywords in the tweets.

By using the tweets and their counts we also get the total count of collected tweets every hour. Finally, by parsing the 'count_and_sentiments.txt' output file, we plot average sentiment of top 40 tweets every hour.
Results

### References to Donald Trump



## Sentiment Analysis Time Series



## Most Positive Hour For Trump's Twitter Sentiments

Apr 14 3pm-4pm
Overall average sentiment: 0.357062

| **Top keywords** | **Top hashtags** |
|---|---|
| vote 1228<br>cruz 878<br>campaign 621<br>people 530<br>watch 504 | #Trump 1875<br>#Trump2016 787<br>#TrumpTrain 438<br>#NYPrimary 431<br>#trump 336 |

## Most Negative Hour For Trump's Twitter Sentiments

Apr 17 1am-2am
Overall average sentiment: -0.245588

| **Top keywords and Counts** | **Top hashtags and Counts** |
|---|---|
| infowars 656<br>delegates 464<br>vote 461<br>cruz 405<br>video 396 | #Trump 1192<br>#Trump2016 268<br>#trump 218<br>#ColoradoProtest 182<br>#TRUMP 159 |

# 7. Conclusions and Future Work

Reflecting our project work we concluded that we expended significant time on the infrastructure setup and operationalizing the data processing pipelines when compared to generating deep insights. The learning was extensive and with a few more weeks we believe we could have delivered additional insights. We also concluded that the project scope was broad from batch big data processing to real-time analytics and sentiment of Twitter streams. Each team member delivered was focused on one of the three capabilities and with more time we would have likely integrated these together on one infrastructure and user experience. Additional conclusions:

## Batch Layer

- 1 TB Reddit data processing consumed significant memory for both Spark and Dask. Recommendations include increasing cluster RAM, adding nodes to distribute processing, modularizing code, persisting objects in memory, and pre-processing the JSON files offline.
- Cloudera was a very efficient method for installing/modifying Hadoop and monitoring performance, health, services. We found it useful to monitor loads when executing intensive data processing code.
- Dask provided a comparable approach to Spark for distributed processing using pure Python. Anaconda Cluster Management significantly simplified Python library deployment across all nodes of cluster from a client machine (MacBook)
- Keyword and part of speech natural language processing provided limited insights about each candidate. This is an area that would require more investment to improve results including filtering common words and increasing the number of keywords for analysis. It may be also interesting to performance sentiment analysis on the top keywords.

## Speed Layer

### Data Insights

1. Noticed a major spike in sentiments when the program was running on April 14th. This was a day after the Town Hall Debate was conducted.
2. Trump's campaign experienced a major dip in sentiment on April 17. This may have to do with negative news stories from InfoWars (hypothesizing based on corresponding hashtags)

### Architectural Insights

3. Spark cluster failed twice because of memory issues resulting from poorly designed garbage collection in Scala code. In future, Scala code written for this could be improved for better garbage collection.

# Appendix

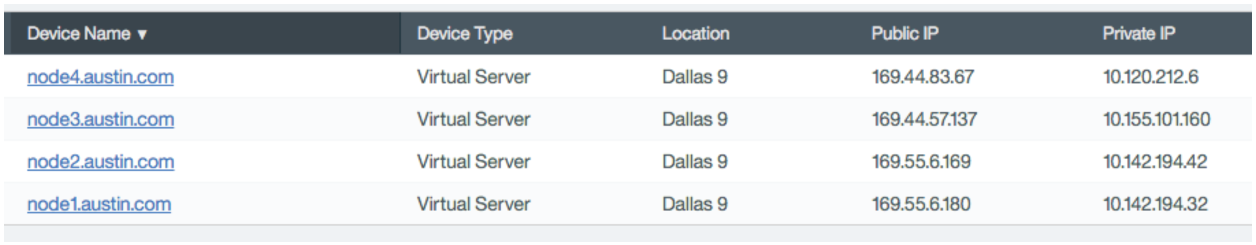
## Anaconda Cluster Management Configuration

The “Bare Metal” installation was used to install Anaconda Cluster Management given that Cloudera and Softlayer were used to create the 4-node cluster. The following configuration files were used to deploy Anaconda to the cluster.

```
$ cat ~/.acluster/profiles.d/bare-metal.yaml
name: bare-metal
provider: bare_metal
node_id: bare_metal
node_type: bare_metal
user: root
num_nodes: 4
machines:
  head:
    - 169.55.6.180
  compute:
    - 169.55.6.169
    - 169.44.57.137
    - 169.44.83.67
```

```
$ cat ~/.acluster/providers.yaml
bare_metal:
  cloud_provider: none
  private_key: ~/.ssh/mids251project
```

## IBM Softlayer Virtual Servers for Batch Data Processing



| Device Name ▾ | Device Type | Location | Public IP | Private IP |
|---|---|---|---|---|
| node4.austin.com | Virtual Server | Dallas 9 | 169.44.83.67 | 10.120.212.6 |
| node3.austin.com | Virtual Server | Dallas 9 | 169.44.57.137 | 10.155.101.160 |
| node2.austin.com | Virtual Server | Dallas 9 | 169.55.6.169 | 10.142.194.42 |
| node1.austin.com | Virtual Server | Dallas 9 | 169.55.6.180 | 10.142.194.32 |

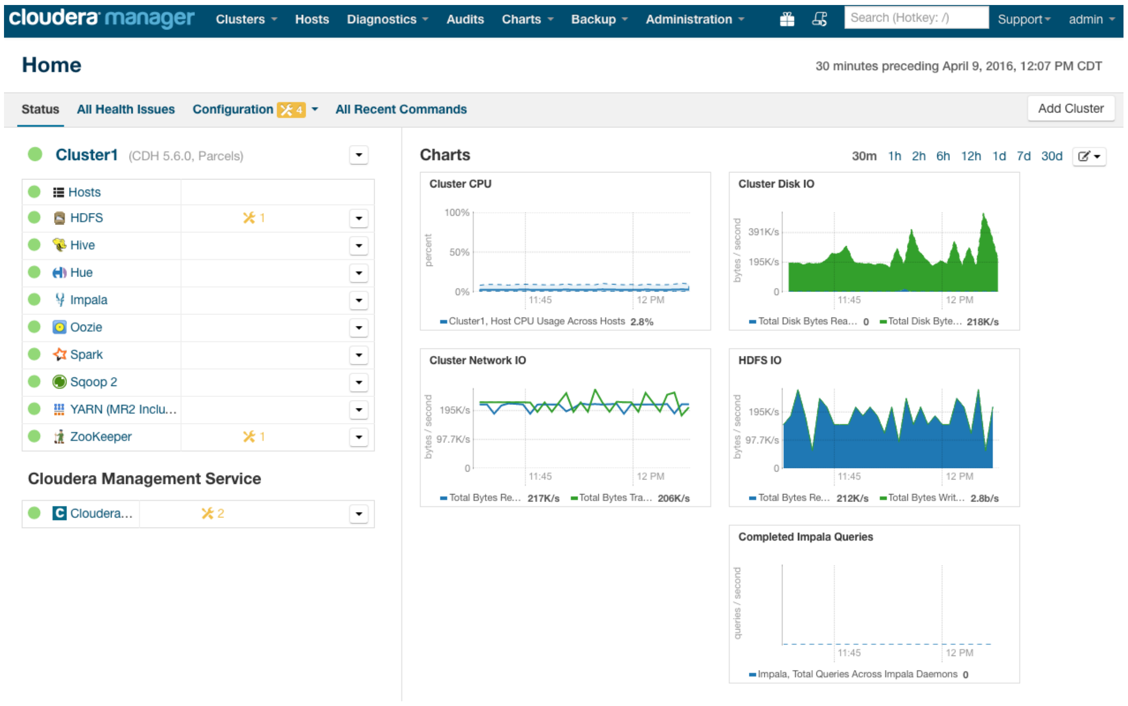
Note that the etc/hosts file must contain the private IP addresses for the Cloudera Hadoop installation.

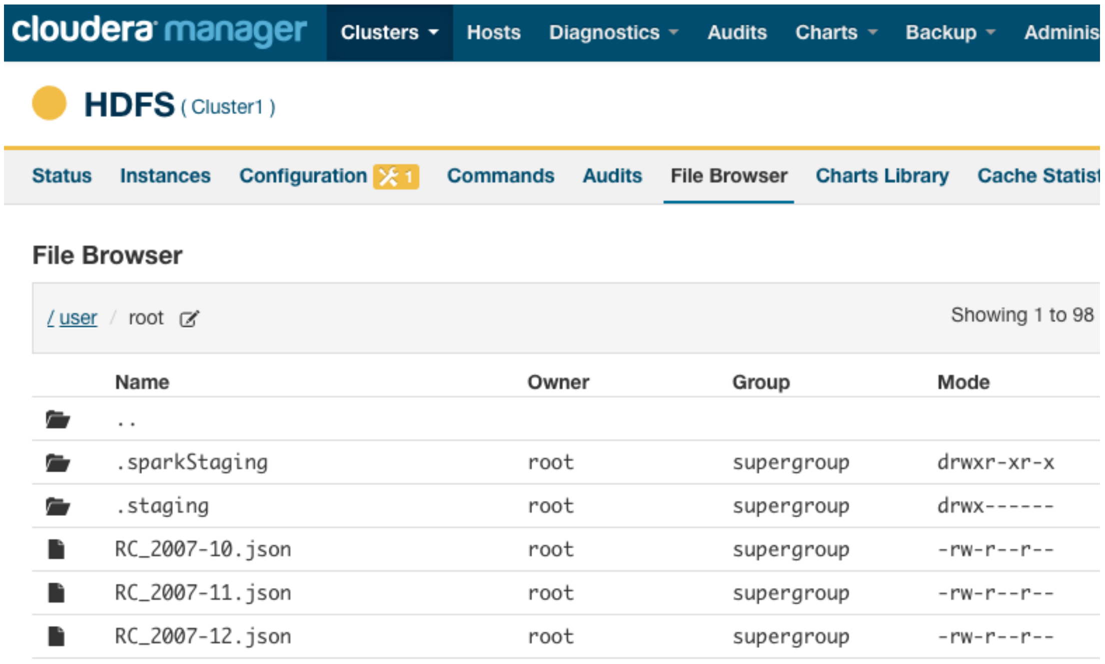
```
10.142.194.32     node1.austin.com node1
10.142.194.42     node2.austin.com node2
```

```
10.155.101.160    node3.austin.com node3
10.120.212.6      node4.austin.com node4
```

The Ubuntu firewall should also be turned off temporarily to enable all ports and protocols for the Cloudera installation.

## Cloudera Hadoop Distribution and Cloudera Manager





HDFS ( Cluster1 )

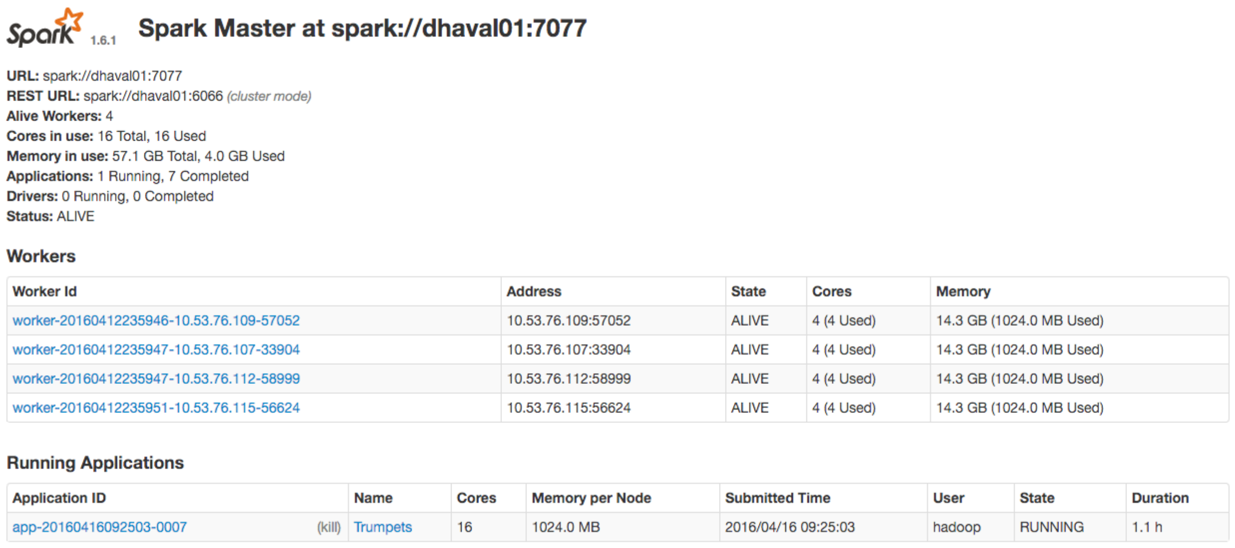
File Browser

/ user / root

Showing 1 to 98

| | Name | Owner | Group | Mode |
|---|---|---|---|---|
| | .. | | | |
| | .sparkStaging | root | supergroup | drwxr-xr-x |
| | .staging | root | supergroup | drwx------ |
| | RC_2007-10.json | root | supergroup | -rw-r--r-- |
| | RC_2007-11.json | root | supergroup | -rw-r--r-- |
| | RC_2007-12.json | root | supergroup | -rw-r--r-- |

## SPARK Configuration for Streaming Data Processing



Spark Master at spark://dhaval01:7077

**URL:** spark://dhaval01:7077
**REST URL:** spark://dhaval01:6066 *(cluster mode)*
**Alive Workers:** 4
**Cores in use:** 16 Total, 16 Used
**Memory in use:** 57.1 GB Total, 4.0 GB Used
**Applications:** 1 Running, 7 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

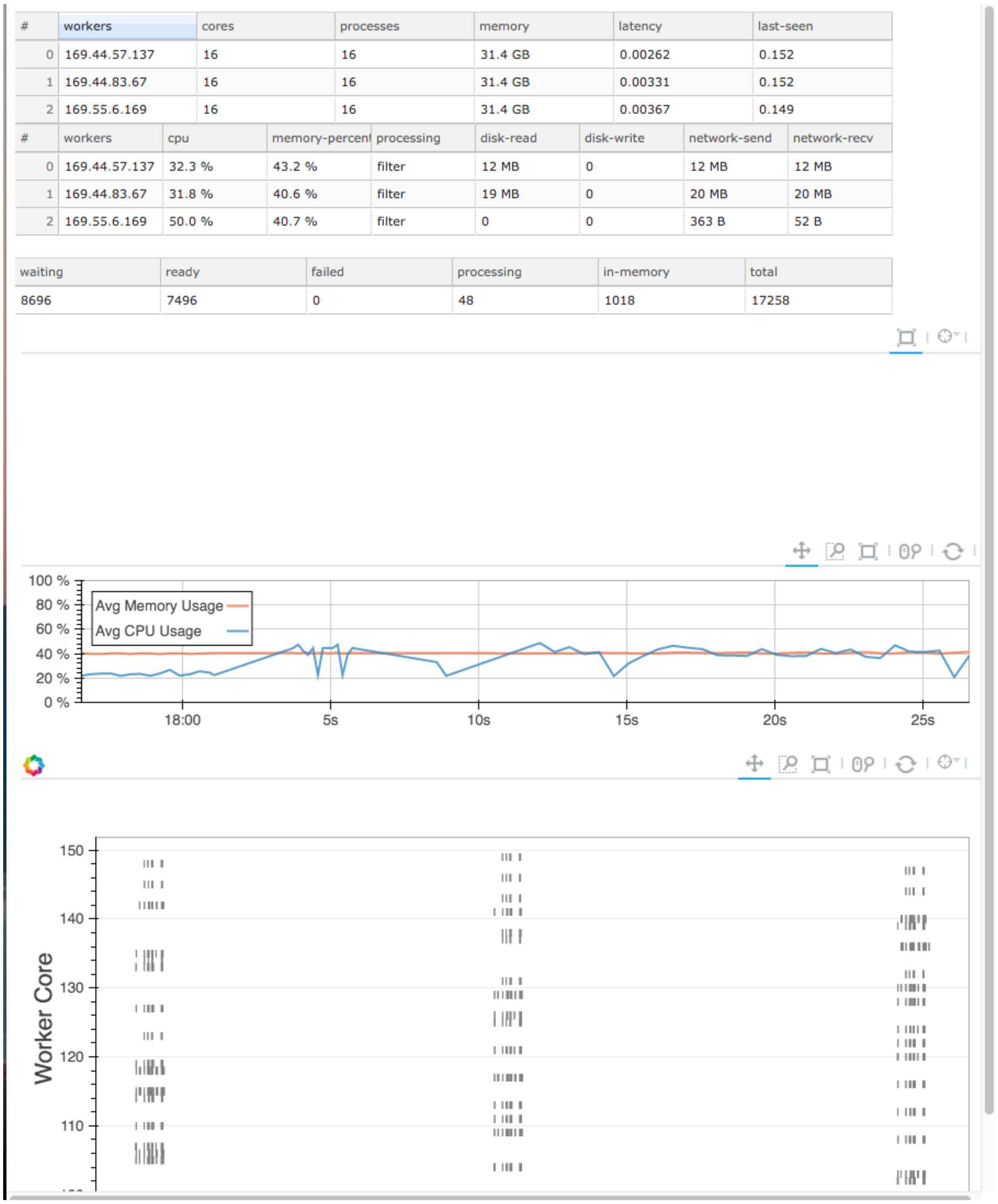
**Workers**

| Worker Id | Address | State | Cores | Memory |
|---|---|---|---|---|
| worker-20160412235946-10.53.76.109-57052 | 10.53.76.109:57052 | ALIVE | 4 (4 Used) | 14.3 GB (1024.0 MB Used) |
| worker-20160412235947-10.53.76.107-33904 | 10.53.76.107:33904 | ALIVE | 4 (4 Used) | 14.3 GB (1024.0 MB Used) |
| worker-20160412235947-10.53.76.112-58999 | 10.53.76.112:58999 | ALIVE | 4 (4 Used) | 14.3 GB (1024.0 MB Used) |
| worker-20160412235951-10.53.76.115-56624 | 10.53.76.115:56624 | ALIVE | 4 (4 Used) | 14.3 GB (1024.0 MB Used) |

**Running Applications**

| Application ID | Name | Cores | Memory per Node | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|
| app-20160416092503-0007 (kill) | Trumpets | 16 | 1024.0 MB | 2016/04/16 09:25:03 | hadoop | RUNNING | 1.1 h |

## Dask status user interface (http://169.55.6.180:8787/status)



| # | workers | cores | processes | memory | latency | last-seen |
|---|---|---|---|---|---|---|
| 0 | 169.44.57.137 | 16 | 16 | 31.4 GB | 0.00262 | 0.152 |
| 1 | 169.44.83.67 | 16 | 16 | 31.4 GB | 0.00331 | 0.152 |
| 2 | 169.55.6.169 | 16 | 16 | 31.4 GB | 0.00367 | 0.149 |

| # | workers | cpu | memory-percent | processing | disk-read | disk-write | network-send | network-recv |
|---|---|---|---|---|---|---|---|---|
| 0 | 169.44.57.137 | 32.3 % | 43.2 % | filter | 12 MB | 0 | 12 MB | 12 MB |
| 1 | 169.44.83.67 | 31.8 % | 40.6 % | filter | 19 MB | 0 | 20 MB | 20 MB |
| 2 | 169.55.6.169 | 50.0 % | 40.7 % | filter | 0 | 0 | 363 B | 52 B |

| waiting | ready | failed | processing | in-memory | total |
|---|---|---|---|---|---|
| 8696 | 7496 | 0 | 48 | 1018 | 17258 |

## References

Reddit JSON data structure: https://github.com/reddit/reddit/wiki/JSON

Anaconda for Cluster Management: http://docs.continuum.io/anaconda-cluster/installation

Dask: http://dask.pydata.org/en/latest/

Distributed: http://distributed.readthedocs.org/en/latest/

hdfs3 (HDFS): http://hdfs3.readthedocs.org/en/latest/

Cloudera quickstart:
http://www.cloudera.com/documentation/enterprise/latest/topics/cm_qs_quick_start.html

GitHub project repository: https://github.com/jamesgray007/Berkeley-W251Project

http://danielfrg.com/blog/2015/07/21/reproduceit-reddit-word-count-dask/]

Jupyter Notebook server installation: http://jupyter-notebook.readthedocs.org/en/latest/public_server.html

Logstash deep dive:
https://www.elastic.co/elasticon/conf/2016/sf/dive-deep-with-logstash-from-pipelines-to-persistent-queues?baymax=rtp&elektra=products&iesrc=ctr

Elasticsearch guide
https://www.elastic.co/guide/index.html