

### Overview:

The overall objective of this application is to count the occurrences of words within Twitter tweets, and conduct some analysis on those tweets.

The primary steps break down as:

1. Receive tweets
2. Parse into words
  - a. Remove URLs and other non-words
    - i. Note 1: This section could be expanded to include articles, pronouns, etc., depending on the type of word/story you were trying to tell
    - ii. Notes, if we were looking for specific tweets, we would need to be much more specific in our search.
    - iii. In this case, we have kept it very high level.
  - b. Make lower case (so in a count ME = me = Me = mE)
3. Store words in a database (insert if they do not exist, otherwise increment count)
4. Run analysis based on word/count values stored in the database.

Breaking this into greater detail:

Tweets are received, parsed, and counted using Storm, and words/counts are saved to a postgres database.

The image to the right is the specific storm architecture used for this project.

To start, the spout receives streaming tweets, as delivered by the Twitter API. The Twitter API is accessed via python, using a package called Tweepy. Tweepy is dependent on using a specific Twitter application. As access rights to the application are required, a new twitter application (via my Twitter account) was created, as access rights were then able to be set.

For the spout, because the request is broad (twitter), parallelism in this step could not be used, because we don't have the level of control necessary in the spout process to ensure different spouts have different types of tweets. Introducing parallelism at this spout would likely mean double-counting values being offered by the API.

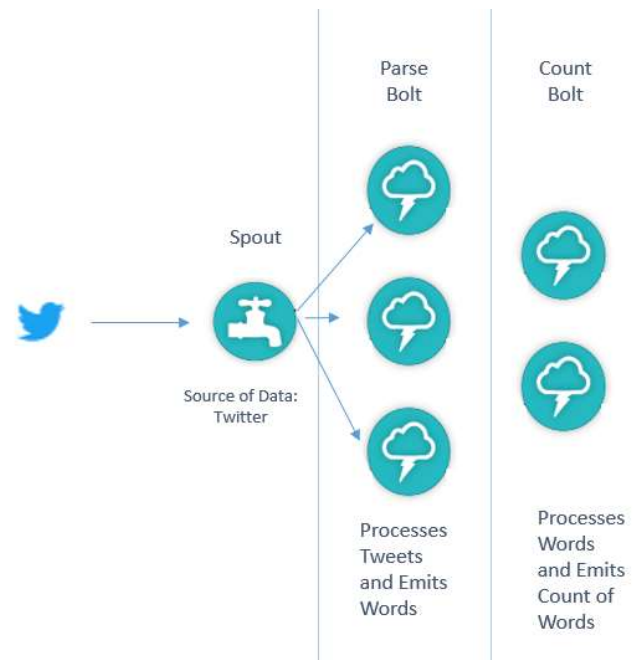
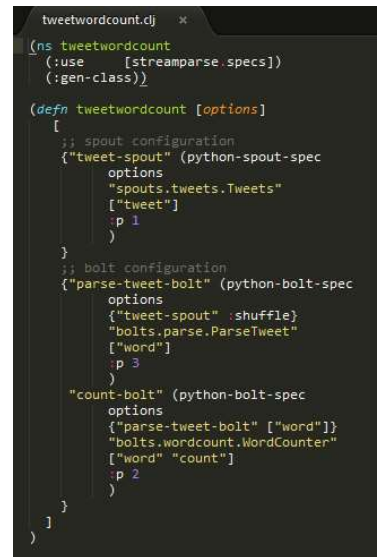


Figure 1: Storm Topology

For the parse bolt, we have 3 level parallelism. For the Count bolt, there is 2 level parallelism. The output of the spout is distributed to the parse bolts (as determined most efficient by storm). The words emitted from the parse bolt are grouped on word, and sent to the count bolt.

Within the count bolt, along with streaming the values to the screen, they are also being stored to a postgres database. For this to work, if the word already exists in the database, the count is incremented. Alternatively, the word is added with a count of 1. In this case, saving the word and the word count is more efficient than saving repeated values of the word (and counting at analysis). Although we are making many updates to the database, the number of words could grow very quickly.

Image to the right shows the storm topology. Below shows streaming tweets.



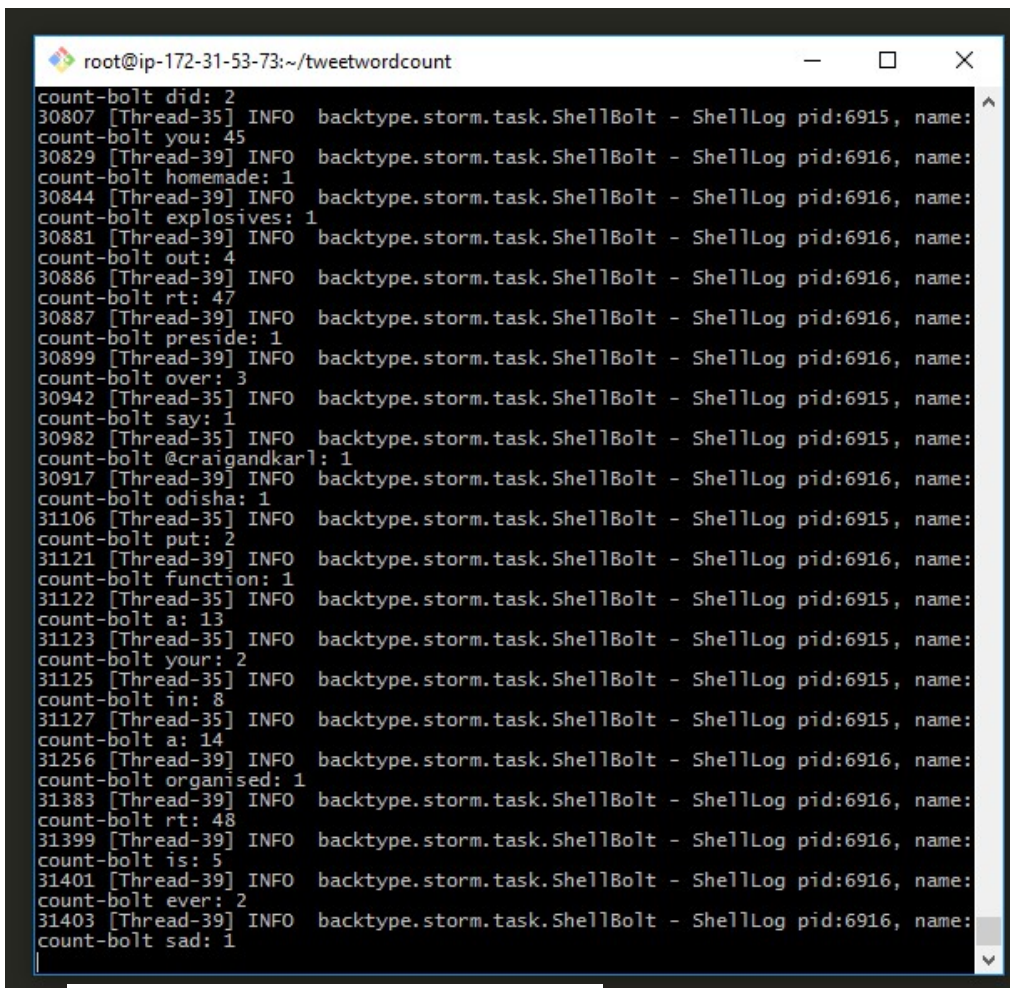
```

tweetwordcount.dj
(ns tweetwordcount
  (:use [streamparse specs])
  (:gen-class))

(defn tweetwordcount [options]
  [
    ;; spout configuration
    {"tweet-spout" (python-spout-spec
      options
      "spouts.tweets.Tweets"
      ["tweet"]
      p 1
    )}
    ;; bolt configuration
    {"parse-tweet-bolt" (python-bolt-spec
      options
      {"tweet-spout" :shuffle}
      "bolts.parse.ParseTweet"
      ["word"]
      p 3
    )}
    {"count-bolt" (python-bolt-spec
      options
      {"parse-tweet-bolt" ["word"]}
      "bolts.wordcount.WordCounter"
      ["word" "count"]
      p 2
    )}
  ]
)

```

Figure 2: Storm Topology



```

root@ip-172-31-53-73:~/tweetwordcount
count-bolt did: 2
30807 [Thread-35] INFO backtype.storm.task.ShellBolt - ShellLog pid:6915, name:
count-bolt you: 45
30829 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt homemade: 1
30844 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt explosives: 1
30881 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt out: 4
30886 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt rt: 47
30887 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt preside: 1
30899 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt over: 3
30942 [Thread-35] INFO backtype.storm.task.ShellBolt - ShellLog pid:6915, name:
count-bolt say: 1
30982 [Thread-35] INFO backtype.storm.task.ShellBolt - ShellLog pid:6915, name:
count-bolt @craigandkarl: 1
30917 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt odisha: 1
31106 [Thread-35] INFO backtype.storm.task.ShellBolt - ShellLog pid:6915, name:
count-bolt put: 2
31121 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt function: 1
31122 [Thread-35] INFO backtype.storm.task.ShellBolt - ShellLog pid:6915, name:
count-bolt a: 13
31123 [Thread-35] INFO backtype.storm.task.ShellBolt - ShellLog pid:6915, name:
count-bolt your: 2
31125 [Thread-35] INFO backtype.storm.task.ShellBolt - ShellLog pid:6915, name:
count-bolt in: 8
31127 [Thread-35] INFO backtype.storm.task.ShellBolt - ShellLog pid:6915, name:
count-bolt a: 14
31256 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt organised: 1
31383 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt rt: 48
31399 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt is: 5
31401 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt ever: 2
31403 [Thread-39] INFO backtype.storm.task.ShellBolt - ShellLog pid:6916, name:
count-bolt sad: 1

```

Figure 3: Streaming Tweets

## File Directory Description

The image below illustrates the file directory used for this project. The main directory holds the folder with the stream parse project (tweetwordcount), screenshots as requested for the project, and various python files, and the primary script (which calls the python file).

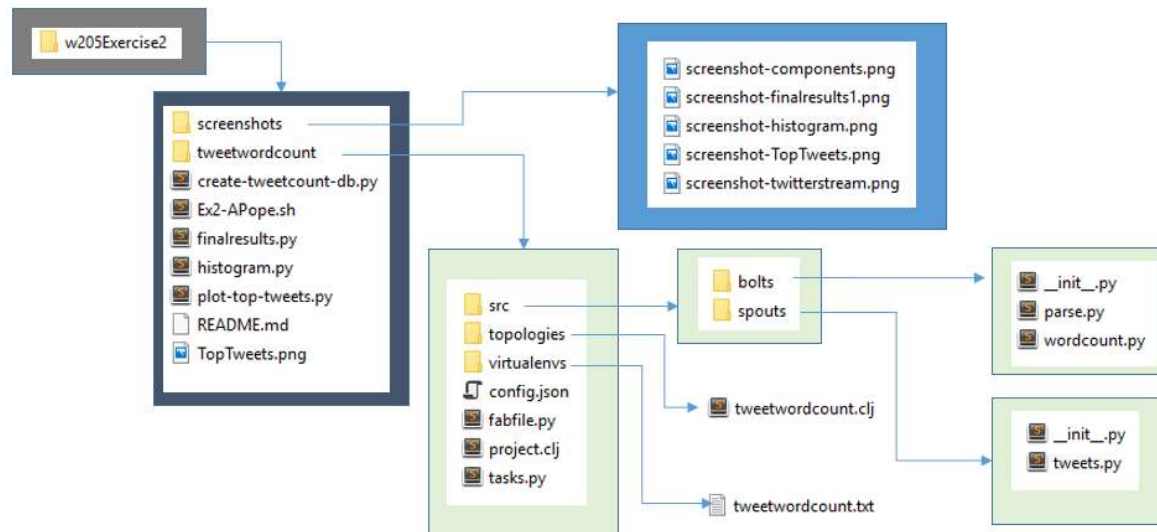


Figure 4: File Directory

## To Run Application:

1. Start instance, and ensure postgres is running
2. Clone project directory (from root directory)  
`git clone https://github.com/APope-UCB/w205Exercise2.git`
3. Navigate to project folder:  
`cd /root/w205Exercise2/`
4. Ensure permissions on the script  
`chmod +x Ex2-APope.sh`
5. Run script  
`./Ex2-APope.sh`
  - a. NOTE: follow prompts if required
  - b. Requires specific packages are installed
    - i. If not already installed on local instance, uncomment lines from script file
    - ii. Includes:
      1. psycpg2
      2. tweepy
      3. pandas
      4. matplotlib
  - c. Script Executes the Following:
    - i. Creates Steamparse Project

- ii. Moves files from git directory into steamparse project
- iii. Creates dB for tweets to be stored
- iv. Streams tweets for a specific period of time
- v. Runs analysis

d. To run specific analysis, the following can also be run

- i. Ensure in project directory  
`cd /root/w205Exercise2/`
- ii. Analysis files  
`python finalresults.py happy` # returns count for 1 word (happy)  
`python finalresults.py` # returns count for all words  
`python histogram.py 13 15`  
`python plot-top-tweets.py`

```
[root@ip-172-31-53-73 w205Exercise2]# python histogram.py 13 15
all: 15
artist: 15
did: 15
life: 15
no: 15
now: 15
happy: 14
or: 14
too: 14
year: 14
been: 13
how: 13
know: 13
look: 13
over: 13
[root@ip-172-31-53-73 w205Exercise2]#
```

Figure 5: histogram

```
Total number of occurrences of 'dog': 3.
[root@ip-172-31-53-73 w205Exercise2]# python finalresults.py happy
Total number of occurrences of 'happy': 14.
[root@ip-172-31-53-73 w205Exercise2]# python finalresults.py goodluck
There are no occurrences of 'goodluck'.
[root@ip-172-31-53-73 w205Exercise2]# python finalresults.py dog
Total number of occurrences of 'dog': 3.
[root@ip-172-31-53-73 w205Exercise2]#
```

Figure 6: Various Results from finalresults.py

### High Level Development Process:

1. Created new m3.medium instance, running UCB MIDS W205 EX2-Full AMI
2. Installed Postgres (directly to instance, vs on volume)
3. Installed various required packages (as noted above)
  - a. psycpg2
  - b. tweepy
  - c. pandas
  - d. matplotlib
    - i. Note 1: Not all of these were installed at the beginning of the development process. However, in future builds (on a new instance), they were added at the beginning.
  - e. Note, to install any of these before running, uncomment install scripts in file:  
`Ex2-APope.sh`
4. Created Twitter Application (following Twitter guidelines)
5. Created dB to hold tweets
6. Created steamparse project
7. Conducted statistics

### Major Obstacle

Getting matplotlib to work correctly and generate a .png was challenging. Matplotlib was throwing an error that Tkinter was not working properly. Via slack, Prajakta shared some insight to changes that could be made in one of the matplotlib libraries, to remove the Tkinter related error, allowing the local

saving of the .png file. To see specific tasks, see the [Ex2-APope.sh](#) script. The matplotlib lines are currently commented out, but the process to follow is there.

### Additional Notes on the Development Process

Testing on the hello-stream-twitter.py file provided, some small updates needed to be made. Specifically, as written, it would not work, because it was looking for “Hello”, with a capital. This needed to be lowercase, as the step before was changing the twitter feed to lowercase.

I followed this example with the tweets the application received, switching everything to lowercase.

Below is a screen shot of the top 20 tweets (also available in screenshots folder).

The application will also create a version of this, using the plot-top-tweets.py python program. This program takes user input for the top N to show (below is 20), and queries the database for the top N words. Then using pandas and matplotlib, it displays a bar chart.

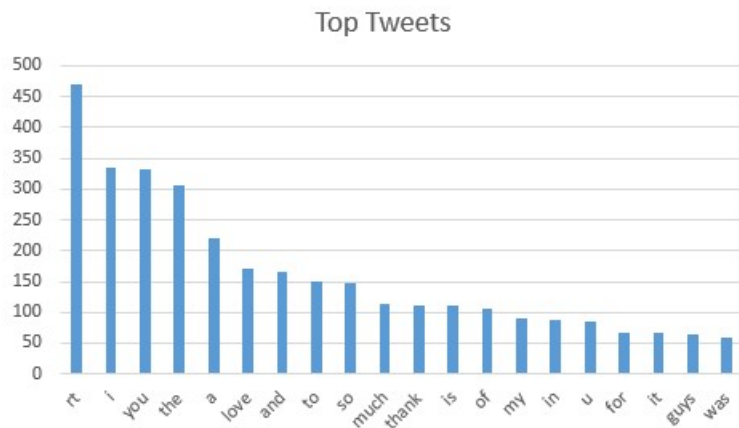


Figure 7: Top Tweets