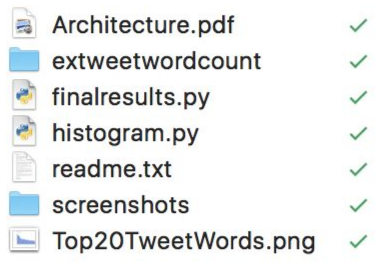
Exercise 2

W205, Spring 2018
Author: Jason Salter

This document will outline the setup, architecture, and running of an application designed to parse tweets into a postgres SQL database. This project will be run in Amazon EC2 with the UC BERKELEY AMI designed for Exercise 2. The github repository 'exercise2' contains:



```
Architecture.pdf        ✓
extweetwordcount        ✓
finalresults.py         ✓
histogram.py            ✓
readme.txt              ✓
screenshots             ✓
Top20TweetWords.png     ✓
```

The primary Spark architecture is contained in extweetwordcount. In the topologies sub-directory, tweetwordcount.clj contains the topology setup:

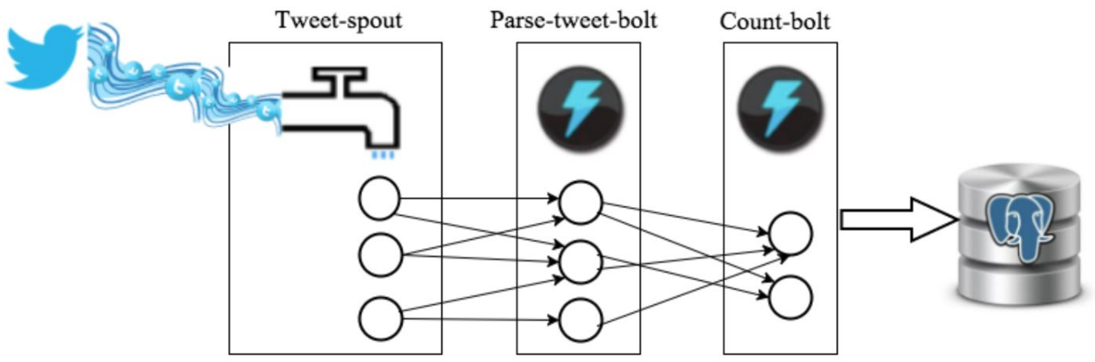


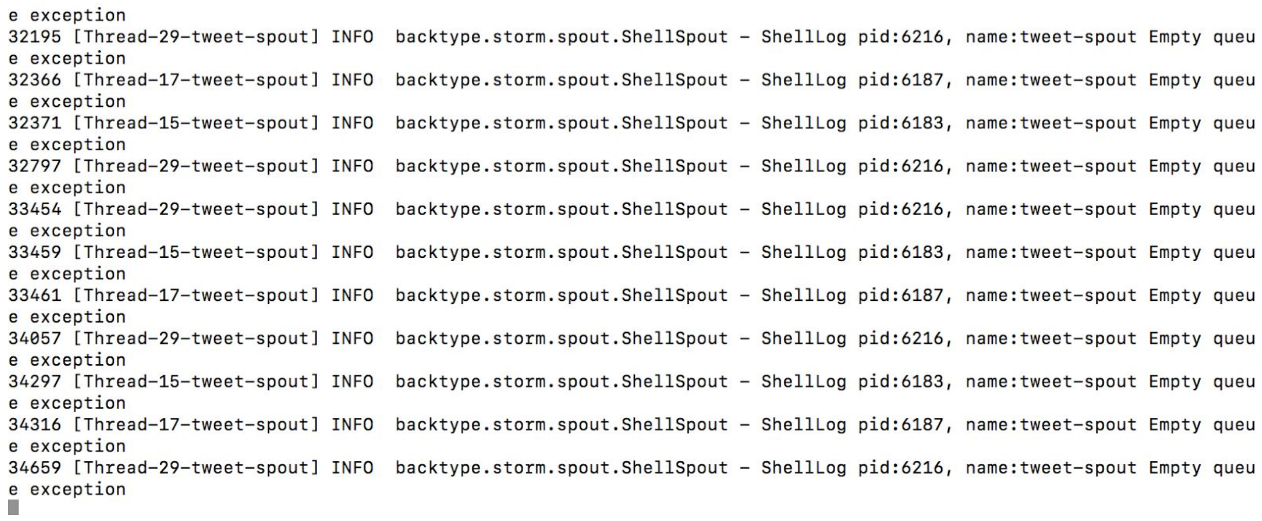
Figure 1: Application Topology

There are three instances of spouts, three bolts for parsing the tweets, and two bolts to update word counts. In order to successfully run the program, extweetwordcount/src/spouts contains tweets.py that will require twitter credentials to access twitter via tweepy. Once the spout is updated with credentials, the program will be ready to run.

The wordcount.py file in extweetwordcount/src/bolts contains the code to create postgres database 'tcount' and table 'tweetwordcount'. The parsed words are the primary key in postgres. If the count of the word is 1, the word and count will be inserted to postgres. If the count is greater than 1, the count will be updated in postgres.

Running the Program

For this part, it is recommended a second command line secure shell be setup. The user must start postgres via /data/start_postgres.sh. After updating the twitter credentials in extweetwordcount/src/tweets.py, change directories to extweetwordcount and execute the 'sparse run' command. This will set up the database and table, and then it will begin parsing the tweets. After initial setup, it will continuously run parsing tweets as available. The three tweet spouts will each show their own process ID and thread:

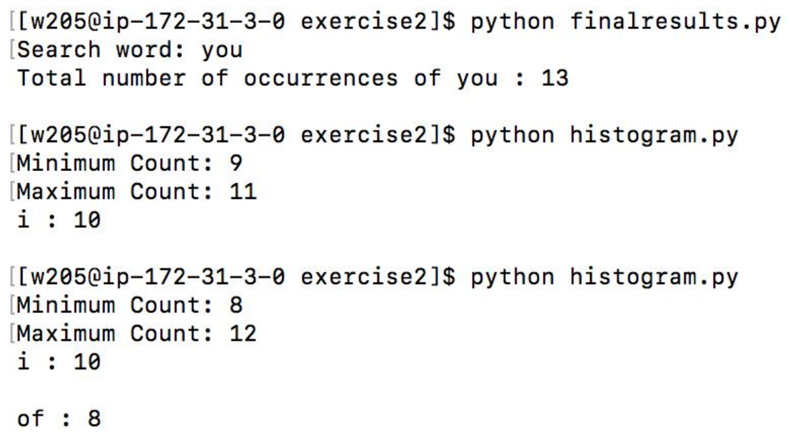


```
e exception
32195 [Thread-29-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6216, name:tweet-spout Empty queu
e exception
32366 [Thread-17-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6187, name:tweet-spout Empty queu
e exception
32371 [Thread-15-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6183, name:tweet-spout Empty queu
e exception
32797 [Thread-29-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6216, name:tweet-spout Empty queu
e exception
33454 [Thread-29-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6216, name:tweet-spout Empty queu
e exception
33459 [Thread-15-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6183, name:tweet-spout Empty queu
e exception
33461 [Thread-17-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6187, name:tweet-spout Empty queu
e exception
34057 [Thread-29-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6216, name:tweet-spout Empty queu
e exception
34297 [Thread-15-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6183, name:tweet-spout Empty queu
e exception
34316 [Thread-17-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6187, name:tweet-spout Empty queu
e exception
34659 [Thread-29-tweet-spout] INFO  backtype.storm.spout.ShellSpout - ShellLog pid:6216, name:tweet-spout Empty queu
e exception
```

In the other command line window, change directories to the exercise2 repository. Run 'python finalresults.py', after which the user will be prompted to input a search word. Press Enter to continue without a search word, in which case an alphabetically sorted list of every word in the database will be output.

If a search word is desired, the user can input the word when prompted, and the count of that word will be returned. The user input is converted to lower case, as is the tweet words parsed into the postgres database. To query a word containing a single quote, the user must escape the single quote by adding another. i.e. "you're" should be searched as "you''re".

The other option is to run histogram.py, which will allow the user to input two integers for processing. Run 'python histogram.py', after which the user will be asked for the minimum count they are searching for. It will then ask for a maximum count to bound the query by. The program will output the words with counts that meet the criteria. Here is an example output:



```
[[w205@ip-172-31-3-0 exercise2]$ python finalresults.py
[Search word: you
 Total number of occurrences of you : 13

[[w205@ip-172-31-3-0 exercise2]$ python histogram.py
[Minimum Count: 9
[Maximum Count: 11
 i : 10

[[w205@ip-172-31-3-0 exercise2]$ python histogram.py
[Minimum Count: 8
[Maximum Count: 12
 i : 10

 of : 8
```

Finalresults.py and histogram.py can continuously be run so long as the sparse run window continues. When satisfied, the user should CTRL-C in the window running sparse run. From there, they can safely shutdown the postgres server with /data/stop_postgres.sh and stop the AMI.