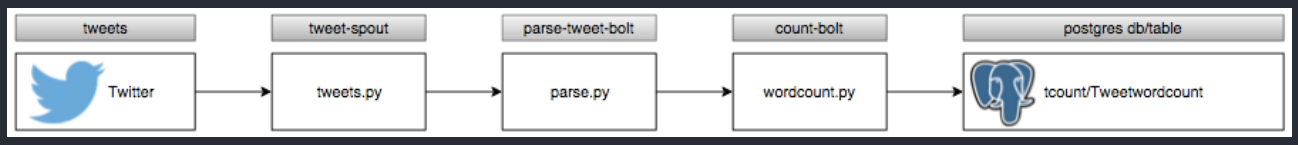
Architecture

The following is the architecture of the program:

Diagram



Description

Topologies

tweetwordcount.clj :

- tweet-spout --> spouts.tweets.Tweets ("Tweet")
- parse-tweet-bolt --> bolts.parse.ParseTweet ("word")
- count-bolt --> bolts.wordcount.WordCounter

Bolts

tweets.py :

- Tweet listener for English tweets
- Emits tweets to parse-tweet-bolt (*parse.py*)

Spouts

parse.py (parse-tweet-bolt):

- Extract tweet, split tweet into words, filter hash tags/RT/@/URLs, strip leading punctuation, and limit characters to ascii
- Emit words that have been cleaned to the next bolt (count-bolt)

wordcount.py (count-bolt):

- In addition to what has been provided, I removed all punctuation from the word that is being extracted, made it lowercase, and removed common words (using a list that I provided)
- Once the pre-processing is complete, I connect to Postgres, select the row in the table where the word is equal to the word being extracted
    - If no rows are returned, the word does not exist in the table yet. If this is the case, then I insert a row into the database with the given word and the count of 1
    - If rows are returned, I update the table with the given word and increase the count by 1

File structure

```
.
|-exercise_2
   |---EX2tweetwordcount
        |---_build
        |---config.json
        |---fabfile.py
        |---logs
        |---project.clj
        |---README.md
        |---_resources
        |---src
             |---bolts
                  |---__init__.py
                  |---parse.py
                  |---wordcount.py
             |---spouts
                  |---__init__.py
                  |---tweets.py
        |---tasks.py
        |---topologies
             |---tweetwordcount.clj
        |---virtualenvs
             |---wordcount.txt
   |---Exercise-2-Subject-205-Real Time Data Processing Using Apache Storm.pdf
   |---finalresults_limit20.py
   |---finalresults.py
   |---hello-stream-twitter.py
   |---histogram.py
   |---psycopg-sample.py
   |---README.md
   |---tweetwordcount
   |---Twittercredentials.py
   |---Twittercredentials.pyc
   |---wordcount

|-w205_Exercise2_TimDavid
   |---screenshots
        |---01_sparse_quickstart_EX2tweetwordcount.png
        |---02_sparse_run_t_300.png
        |---03_streamparse_mid_run.png
        |---04_finalresults_python_script_input_hello.png
        |---05_finalresults_python_script_part_1.png
        |---06_histogram_python_script.png
        |---AMI_selection_2.png
        |---AMI_selection.png
        |---architecture_diagram.png
   |---scripts
        |---create_table_tcount.py
        |---wordcount.py
        |---tweets.py
   |---twitterApplicationCodes
        |---finalresults.py
        |---histogram.py
   |---architexture.html
   |---architecture.md
   |---architecture.pdf
   |---automation.sh
   |---Plot.png
   |---Readme.html
   |---Readme.md
   |---Readme.pdf
   |---Readme.txt
```

Twitter Application Codes

finalresults.py :

- If user gives more than one argument (*i.e. 'hello'*), connect to pmostgres and select the count column from the row where word is equal to the argument provided
- If user does not provide an argument, connect to postgres and query all rows, in alphabetical order, printing each on its own line

histogram.py :

- If user gives exactly two arguments, separated by two numbers, check to make sure that the first number is less than or equal to the second number
- If first condition is satisfied, then connect to postgres and supply the counts of each number in the range supplied
    - In order to make this work, I needed to only keep words that consisted of all numbers (*where word ~ '^[0-9]*$'*), and cast those words as integers (*word::bigint*). This resulted in multiple rows of each number, so I took the aggregate sum of each number
- If the first condition is not satisfied, print error, instructing users to provide the correct form of argument