Application Idea

The application reads from the Twitter stream, pulling out tweets that contain the words "a", "the", "i", "you" or "u". It parses the tweet word, strips out some extraneous information, then writes a count of the frequency of each word to a postgres database. From there, a number of serving options are provided, including python scripts to pull out the words and their frequencies sorted alphabetically, to get the count of a given word, and to easily generate histograms based on a range of counts.

Folder Hierarchy

```
Architecture.pdf
tweetwordcount/
--    config.json
--    fabfile.py
--    finalresults.py
--    histogram.py
--    make_histogram.py
--  make_database_and_table.py
--    Plot.png
--    project.clj
--    README.md
--    results.txt
--    screenshots/
--    src/
            -- bolts/
                  -- __init__.py
                  -- parse.py
                  -- wordcount.py
            -- spouts/
                  -- __init__.py
                  -- tweets.py
--    tasks.py
--    topologies/
            -- tweetwordcount.clj
--    virtualenvs/
            -- tweetwordcount.txt
results.txt
screenshots/
-- screenshot-finalresults-all.png
-- screenshot-histogram.png
-- screenshot-stormtopo.png
-- screenshot-finalresults-singleword.png
-- screenshot-runningtopo.png
```

Architecture

Storm

Spout
The spout creates a tweepy stream, then listens for the words "a", "the", "i", "you", or “u”. This attempts to pull down English language tweets. The spout emits a unary tuple containing the text of the tweet. The output of the spout goes in shuffle grouping to the parse-tweet-bolt

Bolts
Parse tweet bolt: The parse tweet bolt splits the incoming tweet by spaces, then skips words that start with hashtags, “@”s, “RT”s, “http”, and removes everything but valid ascii. From there, it emits a unary tuple of words. The output is fields grouped

Count Bolt: The count bolt receives the set of words. For each word, it attempts to add a new row to the database with that word and a count of one. The bolt writes to the *tweetwordcount* table in the *tcount* database If this fails, we assume it’s because the word is already there and increment the word instead. It also emits a binary tuple of (word, count), which is dropped.
count-bolt

Postgres

The postgres instance has one database of interest, named *tcount*. That contains a single table of interest, *tweetwordcount*. That table has two columns: the primary column “word”, which is a text column, and another column “count”, which is an int.

Serving Layer

The serving layer consists of two scripts.

- finalcount.py -- final count has two modes of operation. With no arguments, it prints all of the words in the table and their counts, sorted alphabetically. With a single word argument, it prints the count for that number of words
- histogram.py -- given a range in the form “x,y” at the command line, it will print all words whose frequencies occur between x and y times, where 0 < x < y

Running the server

The application makes a few assumptions. Instructions for running can be found in readme.txt. The application assumes:

- An instance created from UCB ami-d4dd4ec3
- A properly formatted /data directory

- Completion of the UCB setup script
- Installation of psycopg2 v2.6.2 and tweepy
- A running postgres instance
- Twitter consumer and access credentials, from https://apps.twitter.com/

Note -- when running, there's frequently a URL platform error, as seen in screenshot-runningtopo.png. This does not seem to have any effect

Screenshots

- *screenshot-finalresults-all.png* - Sample execution of the final results script, returning all results
- *screenshot-finalresults-singleword.png* - Sample execution of the final results script, with a single word
- *screenshot-histogram.png* - Sample execution of the histogram script, results between 150 and 151
- *screenshot-runningtopo.png* - Sample execution of the topology, showing the startup warning
- *screenshot-stormtopo.png* - View of the storm topology layout, showing the architecture.