W205 Exercise 2: Architecture Documentation

Lucas Dan

This is a complete Architecture Documentation of my W205 Exercise 2 Twitter Application including directory/file structure, application idea, description of the architecture, file dependencies, necessary information to run the application, etc.

FILE STRUCTURE:

All submission files should be included in the ex2 repository with file structure below:

```
[ec2-user@ip-172-31-29-4 ex2]$ ls
_build  dev-resources  fabfile.pyc  hello-stream-twitter.py  logs  psycopg-sample.py  _resources  tasks.py  test  Twittercredentials.py
config.json  fabfile.py  finalresults.py  histogram.py  project.clj  README.md  src  tasks.pyc  topologies  virtualenvs
[ec2-user@ip-172-31-29-4 ex2]$ cd src/spouts/
[ec2-user@ip-172-31-29-4 spouts]$ ls
__init__.py  tweets.py
[ec2-user@ip-172-31-29-4 ex2]$ cd src/bolts/
[ec2-user@ip-172-31-29-4 bolts]$ ls
__init__.py  parse.py  wordcount.py
```

APPLICATION IDEA:

The idea behind my Twitter Application is to use tweepy, streamparse, and other applications to stream live tweets directly from Twitter's API into my own Postgres database using Apache Storm and my ec2 instance. This is a very common exercise with regards to streaming live data because the Twitter API is accessible and easy to use/connect with streamparse. However, it's important to note that the ultimate goal of the application is to extract insights from these tweets by aggregating words passed in as arguments via python scripts (i.e. psycopg2). These insights can be viewed in the various submission files, as instructed.

DEVELOPMENT PROCESS & ARCHITECTURE DESCRIPTION:

First, I launched an ec2 instance using my personal Amazon Web Services account. Then, I connected to this ec2 instance via SSH using Command Line Interface. After copying over the necessary files, I followed the directions and installed all necessary components on my ec2 instance. To be more concrete, I installed git (cloning all necessary repositories), psycopg2, postgresql, streamparse, tweepy, apache storm, and lein. Note that I configured the postgresql database in a specific way, which can be seen in one of the screenshots. I created a NOSUPERUSER, which I use to create the database and stream live twitter data. I'm also not sure what impact this has on

anyone else's ability to run the application, but this is the way I was able to get the application to run for myself.

After setting up the environment, I followed the instructions to create my own Twitter Application (credentials can be found within the scripts themselves). Following this creation, I inserted the credentials into all of my necessary scripts so that I can use tweepy to interact with the Twitter API, stream/parse tweets, load them into my database for easy extraction (i.e. word counts and analysis of word counts). Of course, I first tested the connection to my Twitter Application using the hello-stream-twitter.py and Twittercredentials.py scripts. To execute these scripts, simply run the command $python hello-stream-twitter.py.

Now that we know this works, I then modified the streamparse project to interact with my Twitter Application by connecting the pieces together to create a full stream tweetword count processing application. To run this streaming application, which will stream in tweets, parse the tweets by extracting words, count all words, connect to a Postgres database instance, and create the Tcount database/Tweetwordcount table, simply run sparse run –d. You can also see screenshots in my submission of me running this streaming Twitter Application.

Regarding the Serving Scripts, my ex2 repository has python scripts called finalresults.py and histogram.py. By running the command python finalresults.py and passing in an argument (i.e. a word) the script will connect to our Tcount

database, select the word count corresponding to the arguments passed in, and output the results (i.e. word counts for each argument) as instructed. Finally, by running the command python histogram.py and passing in 2 arguments (i.e. integers k1 and k2), the script will output all words for which their total number of occurrences in the stream is greater than or equal to the first argument and less than or equal to the second argument.