Project Architecture

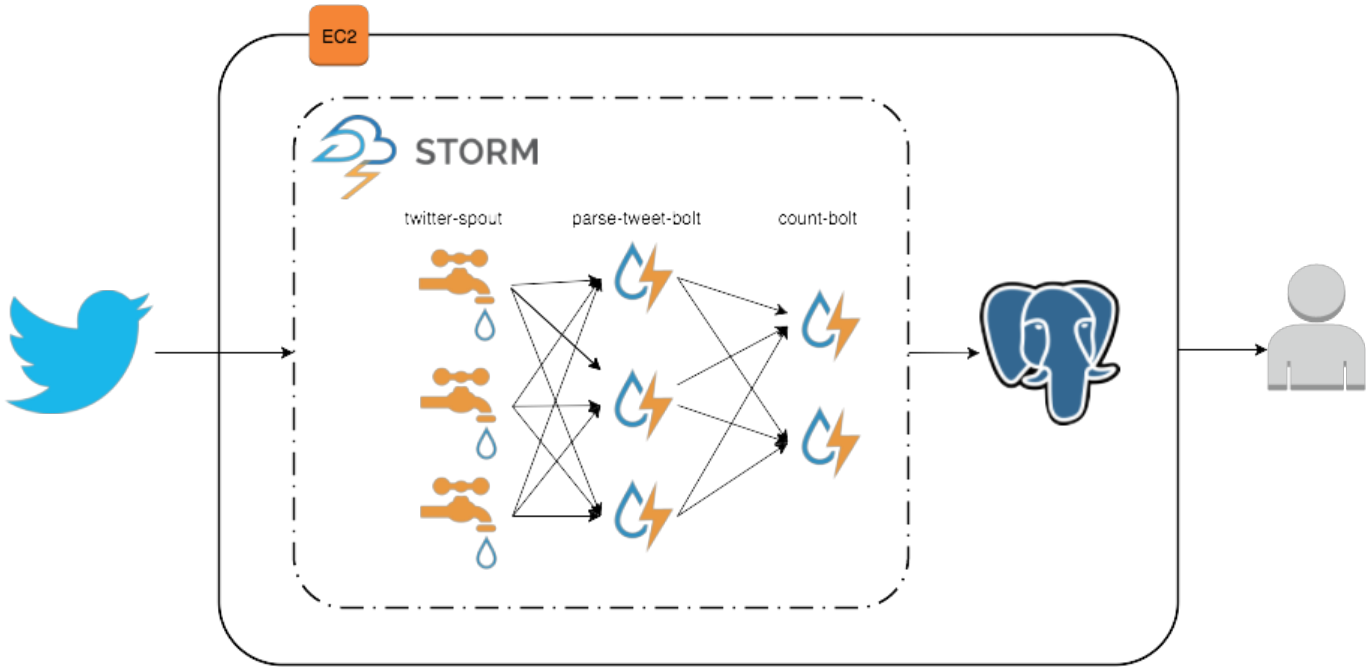
Objective and Application Idea

We would like to create a streaming application that analyzes Twitter data. The application will allow the user to get counts of specific words for a Twitter stream as well as see a histogram of words with counts within a specific range. To collect the necessary data the application will read the stream of tweets from the Twitter streaming API, parse them, count the occurrences of each word in the stream of tweets, and persist the results into storage.

Technologies Used

- Apache Storm and Streamparse
- PostgreSQL and Psycopg
- Twitter Application and Tweepy

Architectural Description



The application uses a Storm topology to collect the necessary data. The topology consists of the following:

- 3 tasks of a `tweet-spout` component, which uses the Tweepy library to collect streaming data from our Twitter Application
- 3 tasks of a `parse-tweet-bolt` which extracts the individual words from each tweet and emits each word to the next component
- 2 tasks of a `count-bolt` which receives each word and updates the appropriate count within the Postgres database

Overall Considerations

Logging

For reliability purposes adequate logging is required in order to diagnose performance of the application. This logging should be adjustable via a configuration file.

Exception Handling

For reliability purposes the application should be fault tolerant to various inputs and conditions. The application should gracefully exit when it enters an exceptional state.

Security

Where applicable proper precautions should be taken to follow best security practices.

Scalability

The choice of technology (Storm) presumes scalability but precautions should be taken to hinder scale out scenarios.

Localization

The application functions on a stream of English tweets and the output language is English. We will not make considerations in the code for internationalization.

Implementation

File Structure

```
exercise_2
|   readme.txt
|   Plot.png
|   postgres_setup.py
|   extweetwordcount.config
|
└──extweetwordcount
|   |   config.json
|   |   fabfile.py
|   |   project.clj
|   |   tasks.py
|   |
|   └──virtualenvs
|   |       wordcount.txt
|   |
|   └──topologies
|   |       tweetwordcount.clj
|   |
|   └──src
|       └──bolts
|       |       parse.py
```

```
│       │       wordcount.py
│       │
│       └──spouts
│               tweets.py
│
└──serving_scripts
        finalresults.py
        histogram.py
        topn.py
│
└──screenshots
        screenshot-finalresults-the.png
        screenshot-histogram-5-8.png
        screenshot-twitter-stream.png
        screenshot-top30-results.png
```

Dependencies

The application assumes that all of the technologies listed in the Technology section above are installed. The application also assumes that PostgreSQL is running and that the credentials present in `extweetwordcount.config` are valid.

Design Decisions

Security

All calls to Postgres use named parameters to avoid SQL Injection attacks

`extweetwordcount.config`

- For security purposes the credentials use to access the Twitter API and the PostgreSQL database are present in this external configuration file.

`config.json`

- The configuration file is set to log at `INFO` level directly to the console. The level and location of the output can be changed by modifying the `envs.prod.log` property.

`postgres_setup.py`

- The tweets will be stored in a database 'tcount' in table 'tweetwordcount'

`tweetwordcount.clj`

- The task parallelism is as defined in the Architectural Description above.
- Input to the `tweet-parse-bolt` uses a shuffle grouping while the input to the `count-bolt` uses a field grouping on the incomming word. This is a crucial configuration step for the correct operation of the `count-bolt` due to its implementation in `wordcount.py` (discussed below)

`tweets.py`

- To avoid internationalization problems the implementation restricts the language of the captured

Tweets to English and further restricts the tweets to those containing the words a, the, i, you, u.

`parse.py`

- Only words that are a part of the body of the tweet are emitted. This exclused hashtags, retweets, user mentions and urls.
- For both storage and security purposes the word must consist of entirely ASCII characters to be emitted. This excludes unicode characters and thus would have to be altered for internationalization.

`wordcount.py`

- The internal word counter is initialized from the database in the event of preexisting word counts.
- The internal word counter is not declared as static and therefore, in order to keep a correct internal word count across multiple `count-bolt` tasks, the topology must specify that the input to this bolt uses a field grouping on the input word.

Results

How to Run Application

In order to run the application follow the instructions present in the `readme.txt` file in the root directory.

Potential Improvements

Scalability

The scripts `finalresults.py` and `histogram.py` are not implicitly scale-out ready. Since they only query the local instance of Postgres each will only have local results. In order to create a truely scale-out solution an application would have to be built across the nodes to run those queries on each instance and then aggregate the results.

Testing

In an enterprise level system there should be adequate unit and integration test coverage to ensure quality and to provide granular tools to diagnose faults in running systems. We do not present unit tests here.