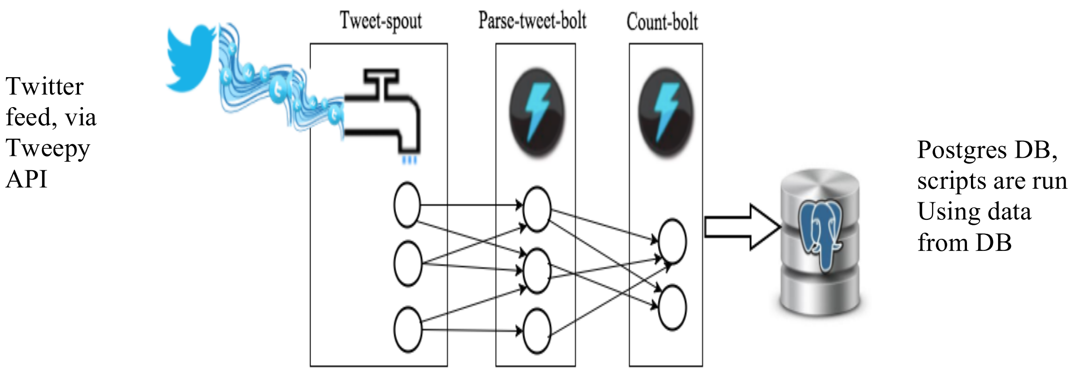
Twitter based Streaming Application - tweetwordcount

1. **Abstract:** Capturing and analyzing live twitter data around a business interest area can provide deeper understanding of current social trends and demands. Historical data can provide information on mainstream trends over a certain period of time, but live data can provide immediate and real-time insight.

   Live tweets show people's active interests, and processing tweets in real-time provides insights. Tweepy, a Python library for accessing Twitter data, is used as the basis of this work. Data is aggregated in a database and is then analyzed.

2. **Architecture:** The figure below shows the overall flow of the implemented application.


Twitter feed, via Tweepy API
Tweet-spout
Parse-tweet-bolt
Count-bolt
Postgres DB, scripts are run Using data from DB

   Twitter APIs are used to access Twitter feeds. **OAuth is the authorization framework** that enables this implemented application to obtain access to Twitter feeds over an HTTP service. OAuth works by delegating user authentication to the service that hosts this user account using (pre-allocated) and Twitter provided Tokens and Keys. OAuth establishes authorization for this application to access the Twitter API.

   A Tweepy library, with OAuth authorization using this application's credentials, provides the live stream of tweets from Twitter to the Tweet-spout component shown above. The Parse-tweet-bolt then parses the tweets, extracts the words from each parsed tweet and emits the words to the next bolt component called Count-bolt. Count-bolt then counts the number of each word in the received tuples and updates the counts associated with each words in the Tweetwordcount table inside the Tcount database.

   At the tail end of the data flow, of the data flow a Postgres instance is created and within it, a database instance called Tcount is made available. A table called Tweetcountwords is then created within the Tcount database.

3. **Directory Structure**: The implemented directory and the file structure follows the lab handout description exactly:
   Implementation variant: Doc, Plot and Scripts directories have been created

   a. **Documentation files**

      tweetwordcount/Docs/Architecture.docx

tweetwordcount/Docs/Architecture.pdf
tweetwordcount/Docs/README.txt

b. Plots and Display files

tweetwordcount/Plot/Exercise_2.ipynb
tweetwordcount/Plot/plot.png

c. Scripts

tweetwordcount/Scripts/finalresults.py
tweetwordcount/Scripts/histogram.py

d. Screenshots (gathered after the implementation is complete)

tweetwordcount/screenshots/screenshot-finalresults.png
tweetwordcount/screenshots/screenshot-histogram-output.png
tweetwordcount/screenshots/screenshot-postgres-setup.png
tweetwordcount/screenshots/screenshot-storm-components.png
tweetwordcount/screenshots/screenshot-twitterstream.png

e. Bolts

tweetwordcount/src/bolts/parse.py
tweetwordcount/src/bolts/wordcount.py

f. Spouts

tweetwordcount/src/spouts/tweets.py

g. Topology

tweetwordcount/topologies/tweetwordcount.clj

4. Project Initialization Steps

Create Streamparse application on AWS:

A streamparse application/project called **tweetwordcount** is created on AWS. The command is

```
[root@ip-172-31-49-109 exercise_2]# sparse quickstart tweetwordcount
```

Configure the Topologies file

The topologies file is then modified to create the components shown in Figure 1. The corresponding Spouts and Bolts files are modified. The topologies closure file captures the need for 1 Spout, 1 Parse-tweet-bolt and 1 Count-bolt.

Implementation variant: A topology with 1 Spout, 3 Parse-tweet-bolt and 1 Count-bolt was also tested and shown to work with no other code changes.

```
ns tweetwordcount
  (:use     [streamparse.specs])
  (:gen-class))

(defn tweetwordcount [options]
   [
    ;; spout configuration
    {"tweet-spout" (python-spout-spec
          options
          "spouts.tweets.Tweets"
          ["tweet"]
          :p 1
          )
    }
    ;; bolt configuration
    {"parse-tweet-bolt" (python-bolt-spec
          options
          {"tweet-spout" :shuffle}
          "bolts.parse.ParseTweet"
          ["word"]
          :p 1
          )
     "count-bolt" (python-bolt-spec
          options
          {"parse-tweet-bolt" ["word"]}
          "bolts.wordcount.WordCounter"
          ["word" "count"]
          :p 1
          )
    }
  ]
)
```

Start the Postgres instance on AWS

```
    [root@ip-172-31-49-109 tweetwordcount]# ./start_postgres.sh
```

Log in to access Postgres as user named "Postgres"

```
    [root@ip-172-31-49-109 tweetwordcount]# psql –U postgres
```

Create a database called Tcount

```
   postgres=# create database Tcount;
```

Verify the creation of the database

```
        postgres=# \list
                                         List of databases
           Name    |  Owner   | Encoding |  Collation  |    Ctype    |
        Access privileges
        -----------+----------+----------+-------------+-------------
        +-----------------------
         metastore | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
         postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
         tcount    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
         template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8
```

Connect to the database

```
        postgres=# \connect tcount
        psql (8.4.20)
        You are now connected to database "tcount".
        tcount=#
```

Create the Tweetwordcount table

```
postgres=# CREATE TABLE Tweetwordcount (word TEXT PRIMARY KEY, count INT);

NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index
"tweetwordcount_pkey" for table "tweetwordcount"
CREATE TABLE
```

Check that table has been created

```
         postgres=# \dt
                     List of relations
          Schema |      Name      | Type  |  Owner
         --------+----------------+-------+----------
          public | tweetwordcount | table | postgres
```

```
(1 row)
```

Check the schema in the created table

```
postgres=# \d tweetwordcount
Table "public.tweetwordcount"
 Column |  Type   | Modifiers
--------+---------+-----------
 word   | text    | not null
 count  | integer |
Indexes:
    "tweetwordcount_pkey" PRIMARY KEY, btree (word)
```

Ensure that new table is empty

```
postgres=# select * from tweetwordcount;
word | count
------+-------
(0 rows)  <-- Zero rows to start
```

5. Implement psycopg2 calls within Python in Spout and Bolts

Method of creating Connection to DB and Cursor – refer to psycopg documentation

```
    self.conn = psycopg2.connect(database="tcount", user="postgres",
password="postgres", host="127.0.0.1", port="5432")
    cur = self.conn.cursor()
```

Implement code to DROP (upon initialization and cleanup) and CREATE Tweetwordcount table (on initialization)

```
cur.execute('''DROP TABLE tweetwordcount''')
cur.execute('''CREATE TABLE Tweetwordcount (word TEXT PRIMARY KEY, count INT)''')
```

Implement tie-in between Tweepy input and Postgres (when tweets are parsed)

```
# extract the word and its count, if it exists already
cur.execute("SELECT word FROM Tweetwordcount WHERE word = %s", (word,))

if cur.fetchone() is None:
      # Word does not already exist, INSERT
      cur.execute("INSERT INTO Tweetwordcount (word,count) VALUES (%s, %s);", (word,
                                              self.counts[word]))
else:
      # Word already exists, UPDATE only the count
      cur.execute("UPDATE Tweetwordcount SET count = %s WHERE word = %s;",
                                            (self.counts[word],word)  )
```

6. Run the application

```
[root@ip-172-31-49-109 tweetwordcount]# sparse run --name tweetwordcount
```

If the prior steps have been successfully implemented, the eventual output should look like this…

```
18286 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt yo: 1
18305 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt fam: 1
18307 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt I'll: 1
```

```
18317 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt take: 1
18320 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt your: 1
18370 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt tickets: 1
18372 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt All: 1
18382 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt endings: 1
18386 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt are: 1
18388 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt also: 1
18399 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt
beginnings: 1
18401 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt We: 1
18403 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt just: 1
18421 [Thread-25] INFO  backtype.storm.task.ShellBolt - ShellLog pid:22086, name:count-bolt don't: 1
```

7. Check that the database is picking up the words

```
tcount=# select * from tweetwordcount;
       word        | count
-------------------+-------
 The               |     1
 Tonkay            |     1
 family            |     1
 in                |     1
 Dover             |     1
 Delaware          |     1
 You               |     1
 come              |     1
```

8. Run scripts/tests on the output data using scripts

```
[root@ip-172-31-49-109 tweetwordcount]# python finalresults.py You
Total number of occurrences of 'You' : 1

[root@ip-172-31-49-109 tweetwordcount]# python finalresults.py
family
Total number of occurrences of 'family' : 1
```

Take a look at words that have between 1 and 6 occurrences logged

```
[root@ip-172-31-49-109 tweetwordcount]# python histogram.py 1 6
('(E1', 1)
('-/', 1)
('1', 1)
('94', 1)
 ('Dorp2(20', 1)
('Dover', 1)
('Dungeon', 1)
<cut>
```

Take a look at all words that have been logged

```
[root@ip-172-31-49-109 tweetwordcount]# python finalresults.py
[('-/', 1), ('1', 1), ('94', 1), ('a', 1), ('about', 1), ('across', 1), ('again', 1),
('character', 1), ('Chief', 1), ('come', 1), ('Curtailed', 1), ('death', 1), ('Delaware',
<cut>      ('take', 1), ('Targets', 1), ('than', 1), ('that', 1), ('the', 8), ('The', 1),
1), ('way', 1), ('We', 1), ('wears', 1), ('well', 1), ('whims', 1), ('why', 1), <cut>
```

The project is now functional and ready to be deployed