# MIDS W205

| Exercise # | 2 | Title | Introduction to the Elements of a Streaming application. |
|---|---|---|---|
| Related Module(s) | 8,9 | Goal | Implement an end to end streaming app. |
| Last Updated | 10/25/16 | Expected duration | 15-25 hours. |

## Introduction

Streaming applications may seem complex but understanding how they operate will be critical for a data scientist. In this exercise, we will explore a streaming application that analyzes the Twitter data. In order to allow you to explore a more complex implementation in a short period of time, you will be developing codes by using an existing code base. You will use Streamparse as seen in Lab 6 with a given topology. The application reads the stream of tweets from the Twitter streaming API, parses them, counts the number of each word in the stream of tweets, and writes the final results back to a Postgres database.

## Scope and Goal

In this exercise, you will capture and process the live Twitter data stream and learn the following tasks:

- How to capturing the live data
- How to setup a stream processing pipeline
- How to process and get insights
- How to store the final processing results to a relational database management system

## Technologies used in this exercise:

Apache Storm, Amazon EC2, python, Twitter API, Streamparse, Postgres, and Psycopg

## Instructions, Resources, and Prerequisites

In the following table you will find all the references related to the tools/libraries and programs that are used in this exercise.

| Resource | What |
|---|---|
| http://storm.apache.org/documentation.html | Apache Storm Documentation |
| https://streamparse.readthedocs.org/en/latest/api.html | Streamparse Documentation |
| https://dev.twitter.com/streaming/overview | Twitter Stream API |
| http://docs.tweepy.org/en/v3.4.0/<br>http://docs.tweepy.org/en/v3.4.0/streaming_how_to.html | Tweepy Documentation |
| http://initd.org/psycopg/ | psycopg |

Note: You are going to use streamparse and install psycopg. We therefore suggest that you do not use your saved instance image with the attached volume from the previous labs or exercise. Instead, we suggest you use the AMI that you used for Lab 6.

## Use Case:

Capturing and analyzing live twitter data around your business interest area can give you a deeper understanding of the current social trends and demands. Historical data can give you information on the mainstream trends over a certain period of time, but live data can provide immediate and real-time insight. For example, a person who manages the ads for the TV programs can capture the live Twitter trends at the time of the live show to engage more TV viewers during a popular TV program (i.e contextual advertising).

So, in this exercise, you will capture live tweets that show people's live interests, process them in real-time to get insights and aggregate the results in a database. Figure 1 shows the overall architecture of the application. Figure 1 also shows the storm topology that you need to develop as part of the application. Using Tweepy library, the application reads the live stream of tweets from twitter in the **Tweet-spout** component. The **Parse-tweet-bolt** parses the tweets, extracts the words from each parsed tweet and emits the words to the next bolt component (i.e **Count-bolt**) in the topology**. Count-bolt** 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. **Tcount** is a postgres database.

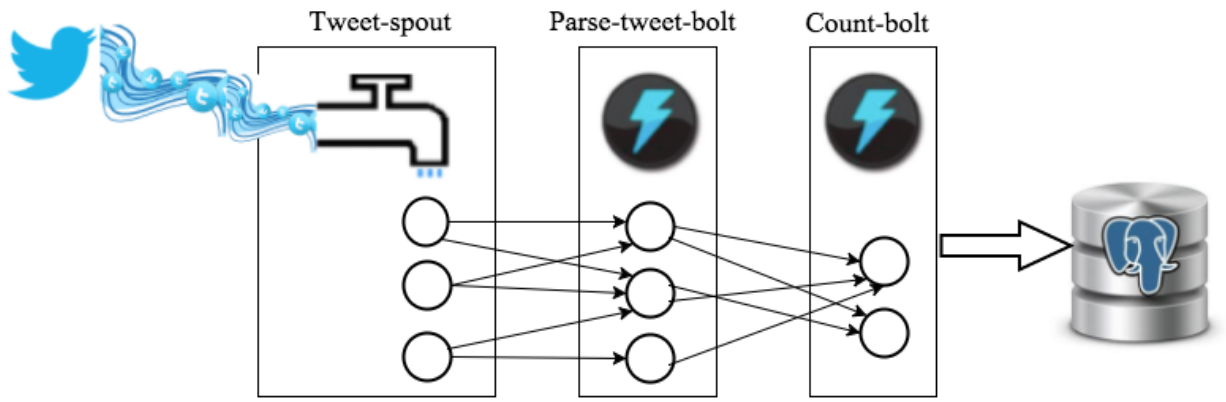



Figure 1: Application Topology

## Overall Guidelines for all steps

In this section we provide the overall guidelines for implementing the real time system as shown in Figure 1. You need to follow the below instructions for each of the steps of your implementation. You would use **UCB MIDS W205 EX2-FULL** AMI for creating your own EC2 server for this exercise. You use your github account to store your programs, data, etc.

# Step-1. Environment and Tool Setup

1. Clone the Github Repository for this exercise from git@github.com:UC-Berkeley-I-School/w205-fall-16-labs-exercises.git
   a. Note 1: If you want to just clone the exercise_2 directory, see the appendix.

2. Create an EC2 instance using the following AMI. Note that this AMI is the same as the AMI that you used for lab 6:

   **AMI Name: UCB MIDS W205 EX2-FULL**
   **AMI ID: ami-d4dd4ec3**

3. Install **psycopg** by running:

   ```
   $ pip install psycopg2
   ```

4. Create a project called **EX2Tweetwordcount** in Streamparse.

5. Copy the files from the **tweetwordcount** directory in your cloned repository and paste them in the corresponding folders in the new **EX2Tweetwordcount** project. The description of the files in the code base is provided in Table 1.

6. Modify the **EX2tweetwordcount.clj** to implement the topology in Figure 1.

*Note1: You may not want to keep your EC2 server live when you are not working, as you will run out of credit that way. So, you could save your work in github as you progress and when you launch your sever, you can re -pull the code and use.*

| Name of the program | Location | Description |
|---|---|---|
| Tweets.py | exercise_2/tweetwordcount/src/spouts/ | Tweet-spout |
| Parse.py | exercise_2/tweetwordcount/src/bolts/ | Parse-tweet-bolt |
| Wordcount.py | exercise_2/tweetwordcount/src/bolts/ | Count-bolt |
| Twittercredentials.py | exercise_2/ | Twitter App Keys |
| hello-stream-twitter.py | exercise_2/ | Sample twitter Stream program |
| tweetwordcount.clj | /exercise_2/tweetwordcount/topologies/ | Topology for the program |
| psycopg-sample.py | exercise_2/ | Sample codes on how to use psycopg |

Table 1: Description of the files in the code base

# Step-2. Twitter application setup

Once you have the topology configured with the codes in the cloned repository, you need to modify the spout program (i.e Tweets.py) to pull the tweets from twitter streaming API. In order to get the tweets, you need to set up a twitter application and get access keys for pulling the tweets out of the twitter streaming API.

The following instructions will step you through the process of acquiring data from Twitter. The first thing that you need to do is to install Tweepy, which is a python library for accessing the Twitter API.

## 2.1 Install Tweepy

The easiest way to install Tweepy is by using pip:

```
$pip install tweepy
```

You may also use Git to clone the repository directly from Github and install it manually:

```
$git clone https://github.com/tweepy/tweepy.git
$cd tweepy
$python setup.py install
```

The next step is to get access to twitter data**.** Twitter data can be accessed over the Web by creating a Twitter application and then using the access keys Twitter provide for the application in your program.

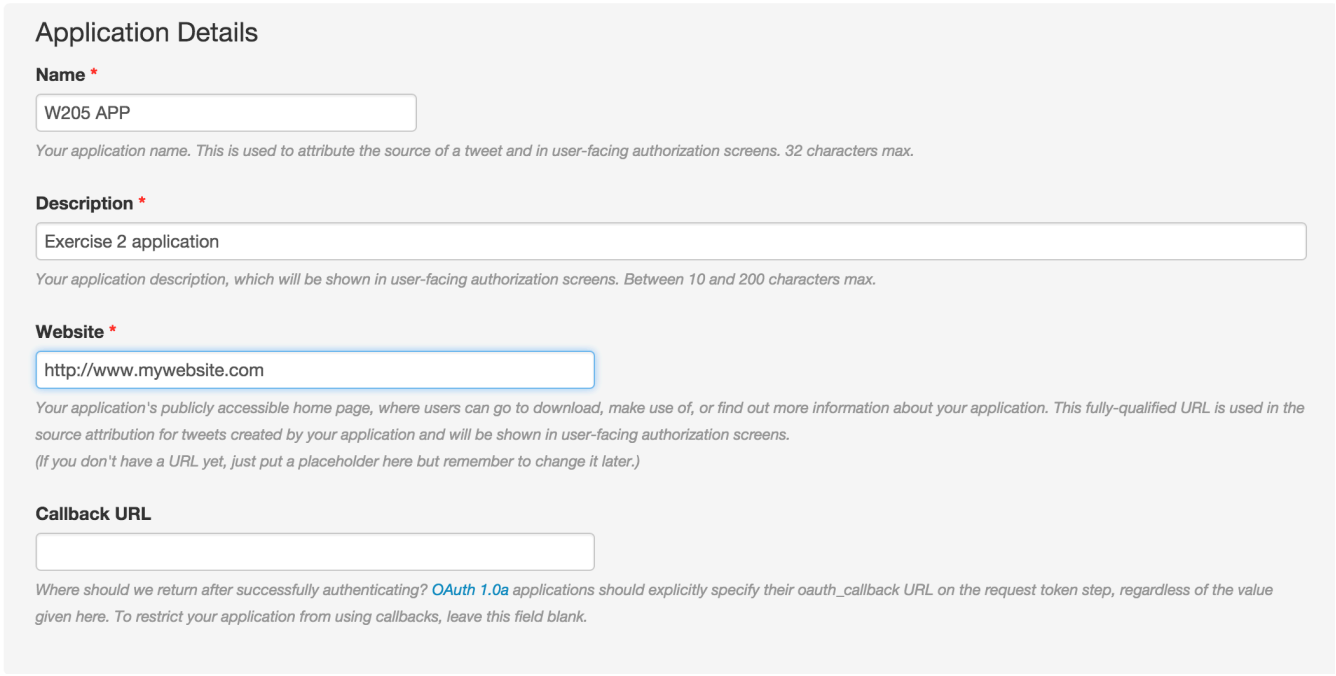
## 2.2 Create an Application

Note: You will need to have a Twitter account to create an application.
To create an application:

1. Login to Twitter (https://www.twitter.com/).
2. Visit https://apps.twitter.com and click on "Create New App".


3. Fill in the application name, description, and Website. The name will be listed in your application list when you return this Website.

# Create an application

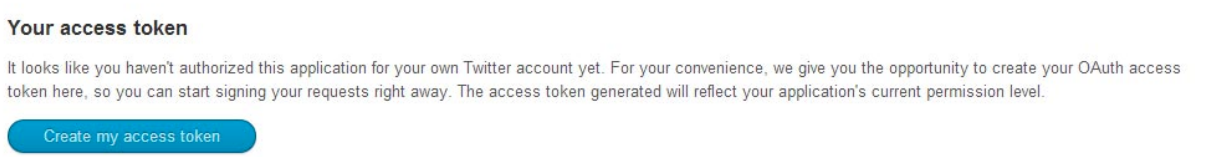



- **Name:** Your app name. It needs to be a unique name across all twitter applications (i.e. no one has used it before)
- **Description:** A short description for your app.
- **Website:** Enter the website address where the app will be hosted.
- **Callback URL:** Ignore this field.

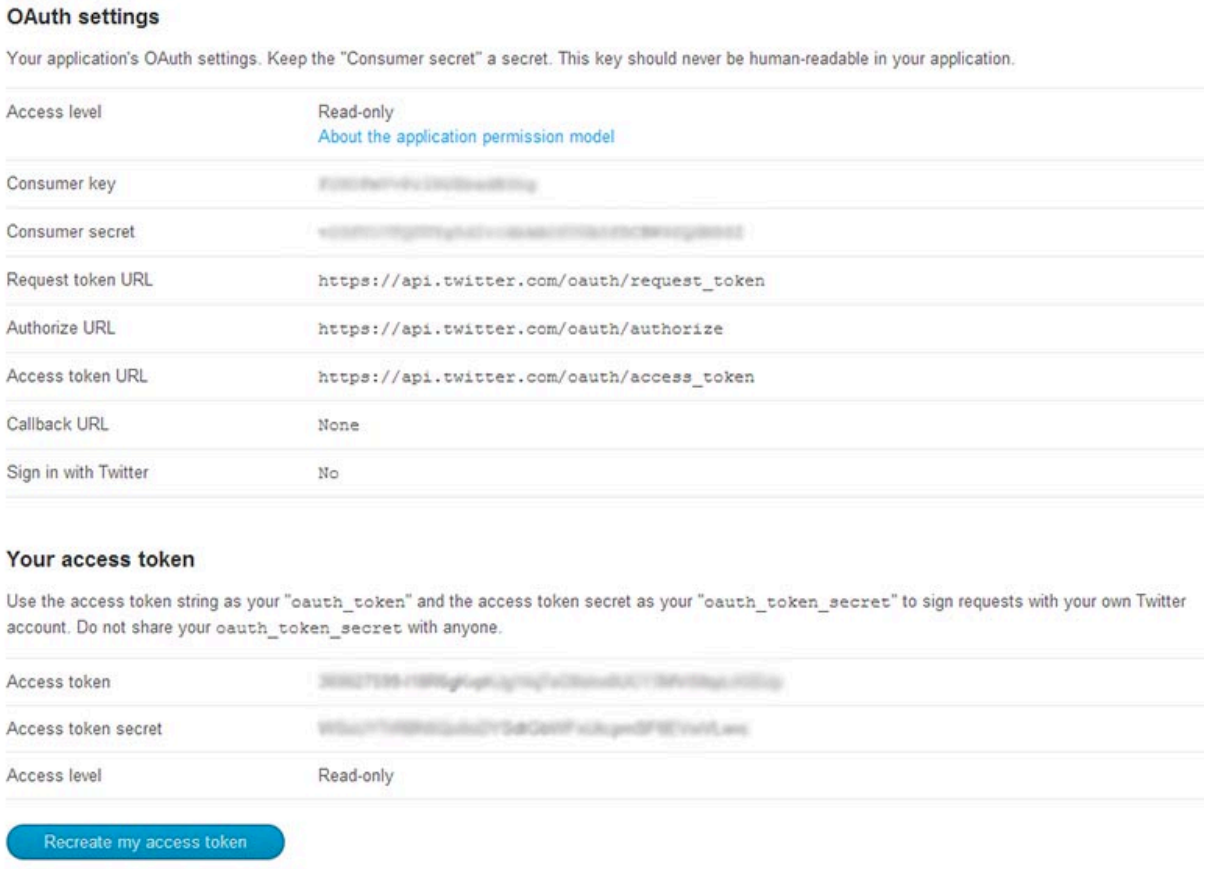
4. Agree to the terms and agreements and click on "Create your Twitter Application"

Once you have successfully created an application, it should take you to the newly created application. Here you must create access keys for subsequent operations by your application. To do so, use the following procedure:

1. Click on the "Keys and Access Tokens" tab.
2. Click on "Create my Access Token" near the bottom of the page.



The response should be relatively immediate.



**OAuth settings**

Your application's OAuth settings. Keep the "Consumer secret" a secret. This key should never be human-readable in your application.

| | |
|---|---|
| Access level | Read-only<br>About the application permission model |
| Consumer key | [redacted] |
| Consumer secret | [redacted] |
| Request token URL | https://api.twitter.com/oauth/request_token |
| Authorize URL | https://api.twitter.com/oauth/authorize |
| Access token URL | https://api.twitter.com/oauth/access_token |
| Callback URL | None |
| Sign in with Twitter | No |

**Your access token**

Use the access token string as your "oauth_token" and the access token secret as your "oauth_token_secret" to sign requests with your own Twitter account. Do not share your oauth_token_secret with anyone.

| | |
|---|---|
| Access token | [redacted] |
| Access token secret | [redacted] |
| Access level | Read-only |

Recreate my access token

Now you have four things (as blurred above):

1. A consumer key that identifies your application.
2. A consumer secret that acts as a "password" for your application.
3. An access token that identifies your authorized access.
4. An access token secret that acts as a "password" for that authorized access.

At any point, you can revoke the access key or regenerated any of these values. To completely disable the application, you must delete the application. This does remove the consumer key, secret, and access tokens from Twitter's system and any program using them will immediately stop working.

## 2.3 Test your Application

In the code base that you cloned, `hello-stream-twitter.py` is a sample application that pulls tweets from the twitter streaming API. This program uses `Tweepy` to work with the streaming API. Use this program to test your application. Change the code in `Twittercredentials.py` and insert your consumer key, consumer secret, access token, and access token secret. You should then be able to just run the program and get tweets:

```
$python hello-stream-twitter.py
```

# Step-3. Application deployment

Now you have a streamparse project and a twitter application, your task is to write codes to connect necessary pieces together as to create a full stream tweetword count processing application as depicted in Figure 1. Your application has the following pieces:

1. A spout connected to twitter streaming API that pulls the tweets from twitter stream and emits them to the parse bolt. You need to modify the necessary codes to complete this part.
2. A tweet-parse bolt that parses the tweets emitted by the spout and extracts the words out of the received tweets.
3. A tweet word count bolt that counts the number of words emitted by the tweet-parse bolt and updates the total counts for each word in a corresponding table inside a database. In order to be able to update the results in a DB, you need to create a Postgres DB called **Tcount** and a table called **Tweetwordcount** table inside **Tcount** database. You also need to modify the code in the `Wordcount.py` in order to update the word count in the **Tweetwordcount** table for each word in the tweet stream. To interact with Postgres, you can use **psycopg** library**.** You can find sample codes on how to use **psycopg** in `psycopg-sample.py` file.

# Step-4. Serving Scripts

In this step, your task is to develop two simple scripts that query the database and return specific results as follows:

1. `finalresults.py`
   This script gets a word as an argument and returns the total number of word occurrences in the stream. For example:

```
$ python finalresults.py hello
$ Total number of occurences of “hello”: 10
```

2. Running finalresults.py without an argument returns all the words in the stream and their total count of occurrences, sorted alphabetically in an ascending order, one word per line. For example:

```
$ python finalresults.py
$ (<word1>, 2), (<word2>, 8), (<word3>, 6), (<word4>, 1), …
```

3. `histogram.py`
   This script gets two integers k1,k2 and returns all the words that their total number of occurrences in the stream is more or equal than k1 and less or equal than k2. For example:

```
$ python histogram.py 3,8
$     <word2>: 8
      <word3>: 6
      <word1>: 3
```

## Submission Instructions:

General guidelines:

1. All code outlined above **must be** committed and pushed to your GitHub repository.
2. All code must be runnable by your instructor in the **UCB MIDS W205 EX2-FULL**.
3. Your GitHub repository **must be** shared with your section instructor via pull request.

Your Github repository should include:

1. Complete and fully functional twitter application codes based on the description above.
2. `Architecture.pdf`: a complete documentation (max 4 pages) of your twitter application including directory and file structure, application idea, description of the architecture, file dependencies, any necessary information to run the application, etc.
3. A directory called `screenshots/` that has at least three screenshots of an end-to-end execution of your application of your choice (name screenshots consistently, e.g.: `screenshot-twitterStream.png`, `screenshot-storm-components.png`, `screenshot-extract-results.png`).
4. `Readme.txt`: a file that shows the step-by-step instructions on how to run the application.
5. `Plot.png`: a bar chart showing the top 20 words in your twitter stream.

## Appendix

In this section we introduce some additional instructions and provide the grading rubric for this exercise.

## Cloning a subdirectory

Git clones a complete repository. If you only want to clone the `exercise_2` directory, you need to use "sparse checkout". Follow the below instructions to clone only the `exercise_2` sub directory.

```
$mkdir ex2
$cd ex2/
$git init
$git remote add -f origin git@github.com:UC-Berkeley-I-School/w205-fall-16-labs-exercises.git
$git config core.sparseCheckout true
$echo "exercise_2" >> .git/info/sparse-checkout
$git pull origin master
```

## Exercise 2 Grading Guidelines:

| Aspect | Description | Grading |
|---|---|---|
| Functionality | **Complete and fully functional spout and bolts programs based on the description**:<br><br>Max 80 pt | • Correct and executable spout: 25pt<br>• Correct and executable bolts: 30pt<br>• Correct topology: 5pt<br>• Finalresults.py: 10 pt<br>• Histogram.py: 10 pt |
| Design/Architecture | **Complete documentation of the system**:<br><br>Max 20 pt | • Correct Project/Folder/ File/DB/table names: 3pt<br>• Description/Execution instructions (readme.txt): 3pt<br>• Architecture.pdf : 10pt<br>• Screenshots: 2pt<br>• Plot.png : 2 pt |