# W205 Exercise 2 Data Architecture

K. Iwasaki

**Application Idea:**

Companies become more sophisticated in analyzing data and turning into business value. One remarkable development is usage of live data. Live data can provides immediate, real-time insight that companies can act on. Apache Storm provide us unique capability to process live data.

For this exercise, we build a simple data architecture to capture and analyze live twitter data. With the architecture, we aim to learn the architecture, its limitations as well as potentials to develop the architecture further to conduct more complicated analysis.

More specifically, this application allows the user to investigate tweets: count each word in the tweets, store the word count data in a database and so forth.

**Architecture:**

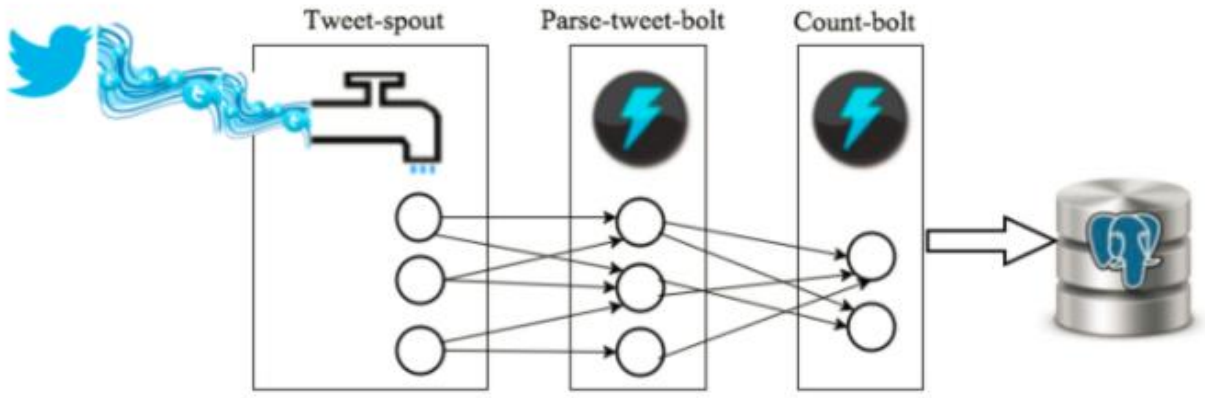


Figure 1: Application Topology

We use the provided AMI to set-up an EC2 instance and set-up the data architecture. Apache Spark is a core of this application. The figure 1 describes high-level data architecture. Twitter data flows through "spout", then to "bolt" for data transformation and processing, then to "count-bolt" for counting, and lastly to a database for storage.

The spout uses Twitter API called Tweepy in order to connect to Twitter and stream live tweets into the application. The spout emits tweets one at a time to be processed.

There are two layers of bolts: the parsing bolt and the counting bolt. The parsing bolt receives the tweets output by the spout module, breaks each tweet into words, filter out words and characters that are not useful, and then emits those words one at a time. When the words are emitted, they are received one at a time by the counting bolt. The counting bolt increments the word's count in the local count and in the database.

**Directory and file structure:**

| Directory/File | Location | Description |
|---|---|---|
| **extweetwordcount** | --- | It contains all streamsparse components |
| **creatdatabase.py** | extweetwordcount/src/bolts | It create a database |
| **parse.py** | extweetwordcount /src/bolts | Parse-tweet-bolt |
| **wordcount.py** | extweetwordcount /src/bolts | Count-bolt |
| **Tweets.py** | extweetwordcount /src/spouts | Tweet-spout |
| **extweetwordcount.clj** | extweetwordcount /topoloties | Topology for the application |
| **wordcount.txt** | extweetwordcount /virtualenvs/ | |
| **congig.json** | extweetwordcount / | |
| **fabfile.py** | extweetwordcount / | |
| **project.clj** | extweetwordcount / | |
| **tasks.py** | extweetwordcount / | |
| **screenshots** | --- | It contains screenshots |
| **README.md** | --- | How to run the application |
| **Twittercredentials.py** | --- | Twitter API Keys |
| **finalresults.py** | --- | When passed a single word as an argument, it returns the total number of word occurrences in the stream |
| **hello-stream.py** | --- | Sample Twitter stream program |
| **histogram.py** | --- | It gets two integers k1,k2 and returns all the words with a total number of occurrences greater than or equal to k1, and less than or equal to k2. |
| **psycopg-sample.py** | --- | Sample code for connecting to psycopg |
| **architecture.pdf** | --- | This document |
| **Plot.png** | --- | A bar chart shows the top 20 words in a Twitter steam |

**File dependencies:**

The code needs to run on an EC2 instance using the provided AMI. This application requires Psycopg2, Postgres, Python, Streamparse, and Tweepy installed on the instance.

**Running the application:**

Follow the step-by-step instructions in Readme.txt for how to run the application.