Spark Project Documentation

# Architecture Design

Shell Script Coordinator

**1\_pipline.sh(running all the time) 21\_tables.sh (crontab job)**

# Architecture Details

## Data Source System = **01\_continous\_listener.py**

* As I wanted to measure the metrices for the tweets adjacent with the four clubs so the keywords I used

**keyword**="real madrid OR HalaMadrid OR Manchester City OR Cityzens OR Inter MILAN OR intermilan OR ForzaInter OR AC Milan OR ForzaMilan"

each club name of them and the most popular hashtag with it.

* U may need to modify the start start\_time, end\_time to suit your current time.
* This listener is continuous job as I put final block of code in while true so it will be running all thetime**.**
* It sends the data as Jason objects to **7777** port.
* Each Jason object has these keys

('id','text', 'retweet\_count','retweet\_count','reply\_count','like\_count','quote\_count','impression\_count','impression\_count','timestamp')

* This job will run on the **default queue** for yarn service

## Data Collection System = **01\_structure\_stream.py**

* This part is mainly used as the data collector from the port, a link between **Twitter API** and a distributed storage system **HDFS**.
* I made another queue which called streaming and I sent this script to this queue so it will run companied with listener script so it will be like **Long Running Job** and it will keep receiving data from the port that was opened in the previous part of the pipeline.
* Inside it, i received the Jason from 7777 port with a defined schema for the columns to give it structure **(Column Names and Data Types)**,
* I made some transformations for the data so I extracted the year ,month ,day, hour from timestamp column .
* I assumed a function to replace the five measures for each tweet to replace them with only a column **“tweet\_value”.**
* I cleaned the text from any mentions, popular emojis , urls by defining a **remove\_urls**
* I defined a list ‘**words’,** which have words (words related to each club) I need to search about them in the text column
* Defining two udfs to search about any word in the list that can be in the text column and if that happens these words will be in the words column (array type).
* The final step here store the output to **HDFS** in the directory **“/twitter-landing-data”** partitioned by the columns (Year, Month, Day, Hour) that are

## Landing Data Persistence

* It was the final step **01\_structure\_stream.py** in the Here is the place where the data are stored persistently in its base format, just partitioned by (Year, Month, Day, Hour) in a directory called **“twitter-landing-data”,** stored as Parquet.
* Also you will need to create a **Hive Table “tweets”** on top if this directory to be used later in the pipeline.

## Landing To Raw ETL

* Here i run some **HiveQL** queries stored as tables**.sql Scripts** to extract the dimensions from the landing data.
* Dimensions can be “tweet-text (id, text)”, “users (id, name)”, etc… **Be Creative…**
* The output dimension is stored under a directory called **“/twitter-raw-data”** and
* I have in this pipeline only one dimension **“club-raw”**  (id STRING, club STRING, tweet\_value DOUBLE) PARTITIONED BY (year INT, month INT, day INT, hour INT)) , from my point of view I don’t any other dimensions for my goals in this project , I tried to get info about the place but it seems that twitter API doesn’t support that.
* Here I used ‘**case when’** to replace the words array with the names of the club adjacent to this word and I used explode to convert the array to rows .

## Raw To Processed ETL

* Here, the aggregations to create the final **club\_processed** is going to be created it is created by grouping by club, year, month, day and hour
* I used a SparkSQL application for this step.
* The fact table contains **number\_of\_tweets** which is the count of the tweets at a specific point in time, sum of tweet\_value **total\_tweet\_value**
* The output fact table should be stored under a directory called **“twitter-processed-data”** .
* I used **.saveAsTable** to Store the output table should be created directly from the inside of the Spark App.

## Shell Script Coordinator

* I made two shell scripts : **1\_pipline.sh** , **21\_tables.sh**
* **1\_pipline.sh**
  + **01\_continous\_listener.py**
  + **01\_structure\_stream.py**
  + **&** for sending processing this command to the background so it can go forward to the next line.
  + **--deploy-mode cluster**  for making this application to run in the cluster
  + **--queue streaming**  for making this application to run in the streaming queue rather than the default queue

I made a new queue for the yarn service with the name streaming and with a capacity 30 %

* **21\_tables.sh** 
  + HiveQL Script
  + SparkSQL App
  + I used crontab to run the 21\_tables.sh on the start of each hour like

0 \* \* \* \* /bin/bash /home/itversity/itversity-material/final/21\_tables.sh.