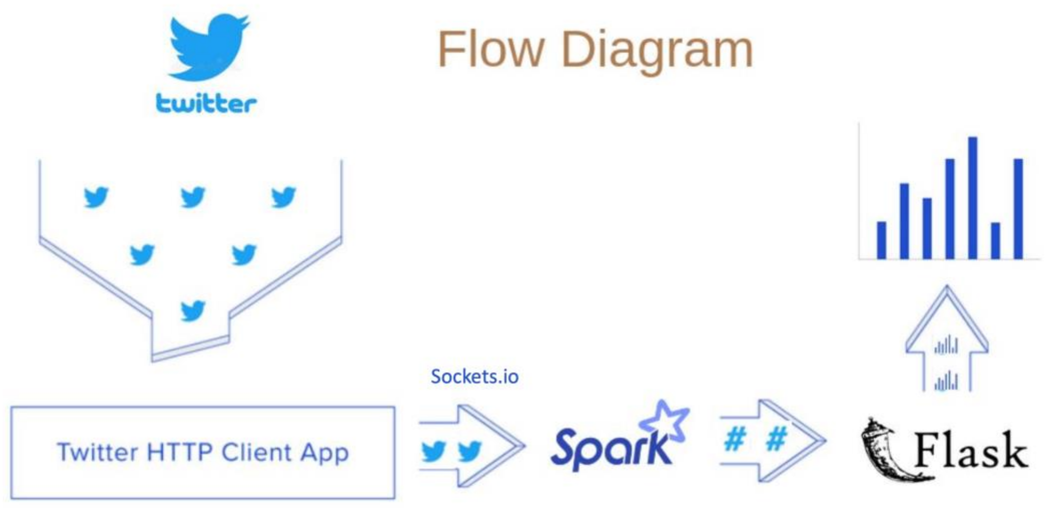
Project report

Creating a pipeline for Twitter streaming using big data

Varun Raghu
A20461361

Abstract:

Twitter is one of the biggest social media websites today, where people tweet and interact with each other about topics they like. Also, many time-sensitive business applications today can Harness the potential of real-time Twitter data and make decisions based on it in fields of IoT application, network intrusion detection, fraud detection. Twitter API allows to stream 50 tweets per second. This needs to be processed, analyzed, and visualized in order to provide valuable information for businesses. A big data pipeline can be used to handle this workload at real time. We will first ingest the data from the twitter API. Then we take this data and stream it into Apache Spark for collecting data in real time for data parallelism and fault tolerance. Now this data can be visualized using Flask.



The goal of the project is to deliver a sophisticated solution to end customers that will provide them with real-time analysis of millions of tweets and dynamic visualization of trending twitter hashtags to assist them better sell their product and raise social awareness about a certain topic.

Literature review:

Twitter:

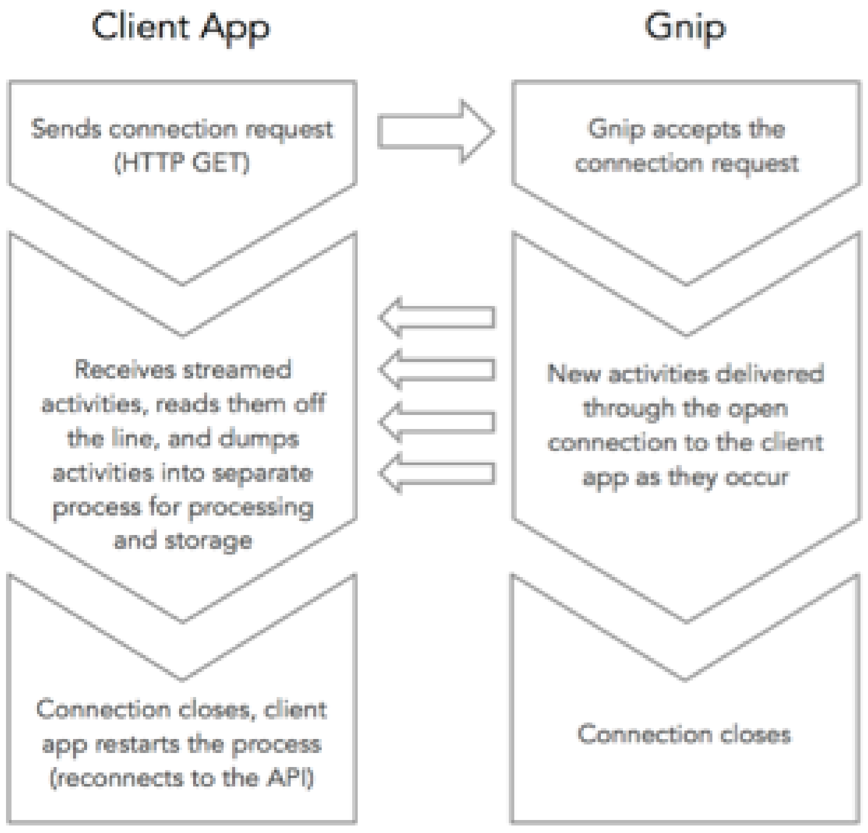
Twitter is a service that allows friends, family, and coworkers to communicate and stay in touch by sending brief, frequent messages to one another. Tweets are posted by users and can include text, images, videos, and links. These tweets are sent to your followers, placed on your profile, and searchable on Twitter.

Thousands of people use Twitter to promote their retail centers, consulting businesses, and hiring services, and it is effective. The significant business impact of Twitter drives the analysis of real-time data from Twitter to derive insights that may be put to use.

Filtered Stream API:

The real-time stream of public Tweets can be filtered by developers using the filtered stream endpoint community. This endpoint category's functionality includes a number of endpoints that let you create and manage rules, as well as use some rules to filter a stream of real-time Tweets and return relevant public Tweets. Users of this endpoint community can track the conversation surrounding competitions, observe how patterns change in real-time, and much more. They can also listen in real-time for pertinent issues and actions.

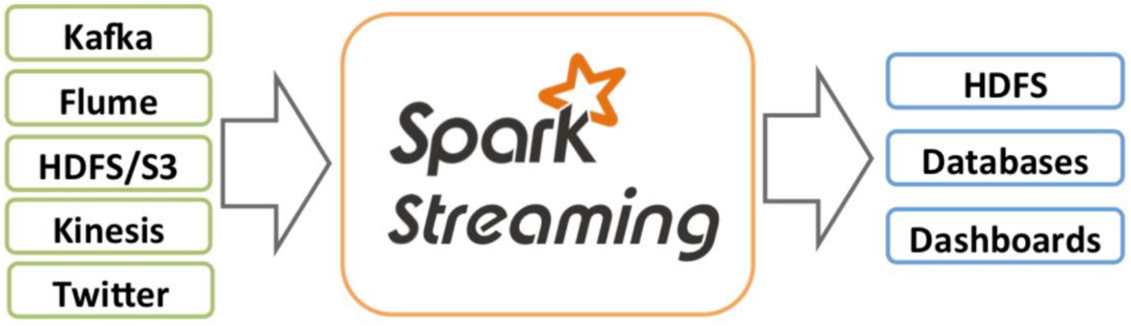
PowerTrack, Volume (such as Decahose, Firehose), and Replay streams all use the Streaming HTTP protocol to transmit data over open, streaming API connections. A single connection is established between your app and the API, and new results are provided via that connection as new matches occur, as opposed to data being delivered in batches through repeated queries from your client app like a REST API would. As a result, a high-throughput, low-latency delivery system is created.


Client App
Gnip
Sends connection request (HTTP GET)
Gnip accepts the connection request
Receives streamed activities, reads them off the line, and dumps activities into separate process for processing and storage
New activities delivered through the open connection to the client app as they occur
Connection closes, client app restarts the process (reconnects to the API)
Connection closes

Apache Spark:

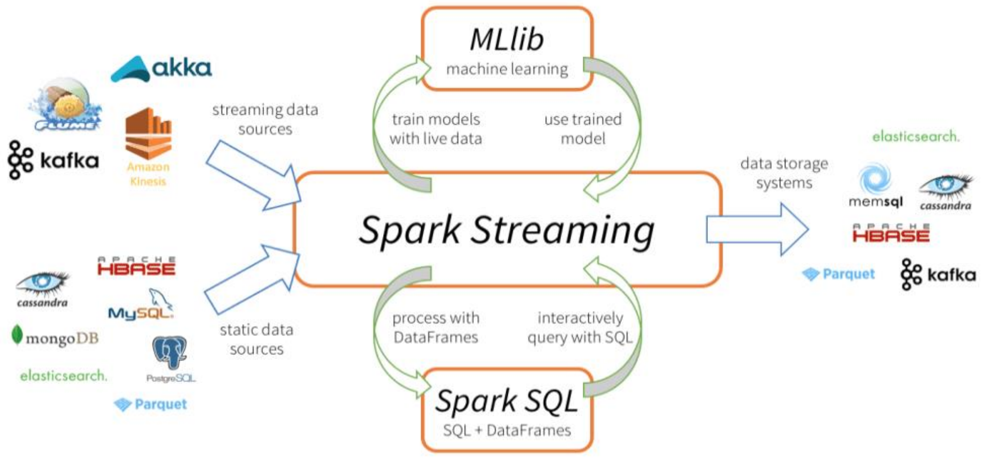
Scalable fault-tolerant stream processing, such as Apache Spark, is capable of handling both batch and streaming workloads. With the help of the Spark Streaming API, real-time data from many sources, including Apache Kafka, may be processed. This transferred information is pushed into databases, file systems, and real-time dashboards. The fundamental abstraction that describes a stream of data broken up into small pieces is a discretized stream, or DStream. DStreams are built on RDDs, the primary data abstraction in Spark. This makes it possible for Spark streaming to effectively interact with other Spark components like MLlib and Spark SQL.

It makes logical to use a streaming product like Apache Spark Streaming to process the data because it is coming in as a stream. This keeps data in memory until it is time to write it to disk. On the other hand, information that is streaming has significance when it is happening in real time. Apache Spark offers Fast recovery from errors and stragglers. Also, it has better resource allocation and load balancing.



Spark Streaming and Spark Structured Streaming are two methods offered by Apache Spark for working with streaming data. Real-time data processing is made possible by Spark Structured Streaming, a distributed and fault-tolerant streaming engine. It is constructed on top of the Apache Spark SparkSQL engine, which manages streaming data. With the aid of caching and checkpoints, it provides quick recovery from any faults and is fault-tolerant and scalable.

I have used Python's Pyspark interface for Apache Spark and created Spark apps utilizing Python APIs. The initial step was to construct a local SparkSession, which serves as the foundation for all Spark-related activities. Then, using sockets, I have built a streaming DataFrame that symbolizes text data obtained from a server listening on Localhost. After receiving the data from the socket, I split each line into numerous rows using the built-in Spark functions "Split" and "explode" and stored them in a Dataframe. Then, we use Spark's 'groupBy' function to provide a running word count. Finally, I transmit the spark to flask results using REST API.



Flask:

Flask is a micro web framework based on the Werkzeug WSGI toolkit and the Jinja2 template engine. It permits extensions that add application functionalities as if they were built into Flask itself. I chose Flask as the server since it is a lightweight web application framework that is designed to deliver results quickly while also allowing for future expansion. With Flask, the code is always simply what the developers put in it, with no extra code in charge of things we don't use. In this case, the Flask server collects hashtag data from Spark and directs it to the frontend.

I have created three REST APIs in the server out of which one is the default GET API that is used to render HTML template for the dashboard whenever the server starts. After processing Twitter's data, Spark will send hashtag data to the server through POST API "/updateData". Then the frontend will call GET API “/refreshData” to get hashtag data from the server. These two APIs are repeatedly called by Spark and frontend respectively to update hashtag count on the dashboard and the dynamic updating on the frontend is done by the Jinja2 template engine.

Apexcharts:

Apexcharts is a modern, free, and open-source charting library that has its integrations with Vue, React and Angular frameworks. This library has beautiful and interactive charts and aids developers in creating attractive and dynamic web page visualizations. It's MIT-licensed and free to use in commercial applications. Apexcharts has NPM support, and it is better, with bigger datasets, but not by much.

In this case, JavaScript calls the server API to GET hashtag data and then it passes to Apexcharts API to get a responsive and dynamic bar graph. For this implementation of the trending hashtag dashboard, a dynamic apex bar chart has been created which is loaded with the options to provide responsive visualizations on frontend. Frontend will call the backend server’s GET API to fetch the trending top 15 twitter hashtags in real time with their corresponding tweet count. These data values are then passed into the bar chart through the series option. To get real time visualization on frontend , the GET API will regularly be called in an interval of 2 seconds. The bar chart is also packed with the option of responsiveness and adapt to different screen sizes making It easy to read.

Implementation:

Software Installation:

JDK Installation:

1. Go to https://www.oracle.com/java/technologies/javase-jre8-downloads.html and download the java runtime environment for your operating system.
2. After downloading the executable file, follow the prompts to install Java runtime environment.

Python Installation:

1. Download and python 3.6 installer from https://www.python.org/downloads/
2. Execute the python installer and continue with the prompts and install python.
3. Select the path for installation.

Setting up Twitter:

1. A Twitter developer account is needed for this project by going to https://developer.twitter.com/en/apply-for-access
2. After creating the account, create a project and navigate to the applications keys and tokens page, and save your app's access token, access token secret, consumer key, consumer secret, and bearer token.

Program execution:

The first step is to install all the necessary packages in order to execute the program. Go to the project folder in the terminal, and enter the following command:

pip install -r ./requirements.txt

There are two ways to implement the program.

Automatic execution:

1. The first way is to do an automatic run. The user needs to run the run.sh file from the project directory.
2. The **run.sh** file contains default keywords and number of pages.
3. The keywords are “nfl nba football basketball”.
4. The number of pages is 15 per keyword.
5. The process will end after 3 minutes and will go through 15 pages for each keyword.

Manual execution:

1. There are three different programs to run.
2. First is the flask application. This is named as app.py. Go to the project folder in the terminal and run the program as **python3 ./app.py**.
3. Second is the twitter_app.py. To run this program, go to the project folder in the terminal and enter **python3 ./twitter_app.py -p _<no_of_pages>_ -k _<"keywords">_**.
4. Replace the no_of_pages to the number of pages you want for each keyword to get from twitter. And replace the keywords to the keywords you want in quotes. For example, "nfl nba football basketball".
5. Next enter the series of following commands:

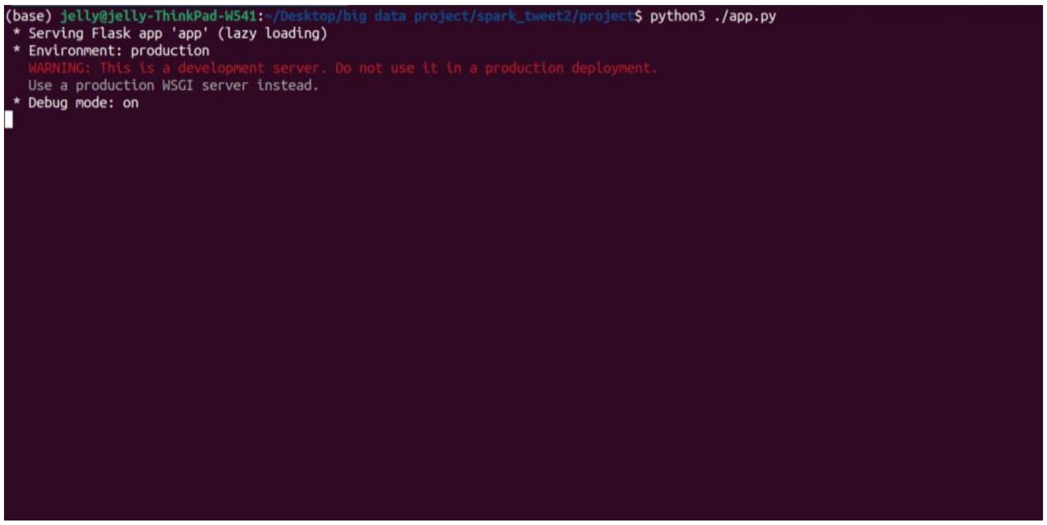
export PYSPARK_PYTHON=python3
export SPARK_LOCAL_HOSTNAME=localhost
python3 ./spark_app.py

6. While executing the above steps, go to **http://localhost:3000/** for visualization.
7. To kill the program, open a new terminal and enter "**killall python3**".

Process:

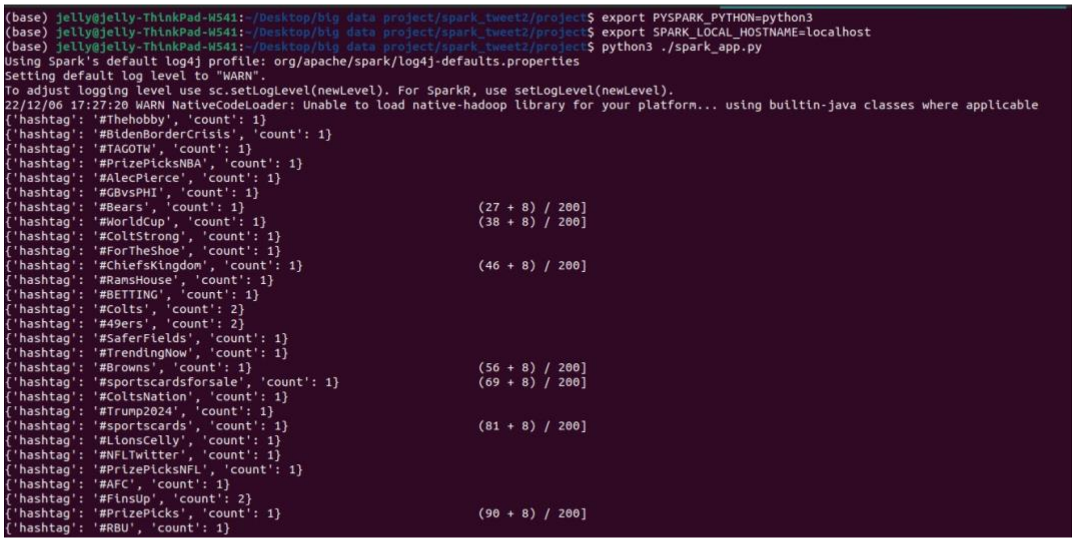
- First, we retrieve tweets from Twitter.
- The tweets are retrieved using the keywords given in the input.
- The data is processed using PySpark, and tweets and hashtags are separated.
- Then we use a TCP Socket to send tweets to spark.
- These trending hashtags are processed using Apache Spark.
- We are utilizing the flask web app to provide the data in a visual format.

Terminal for app.py:

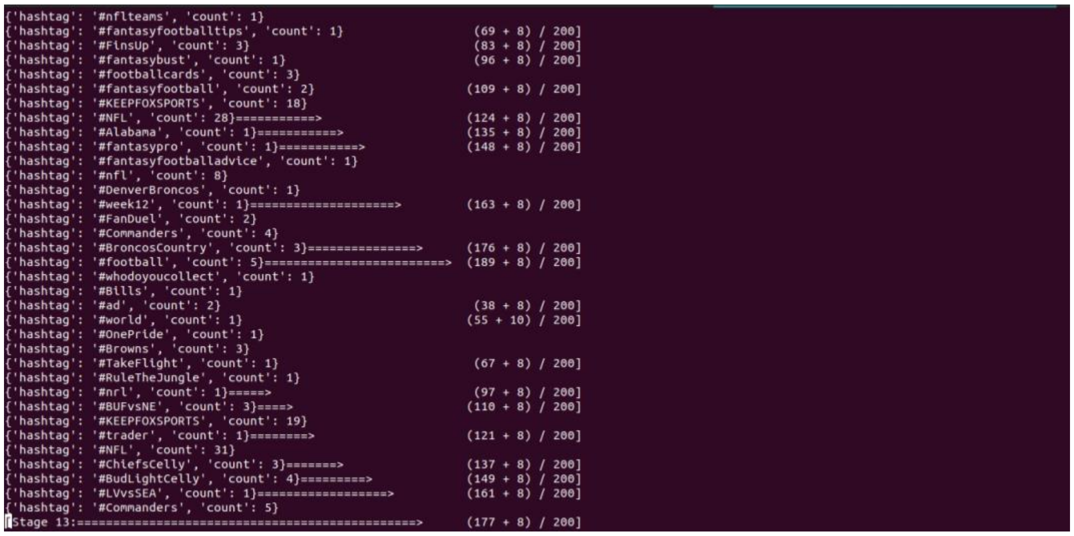


```
(base) jelly@jelly-ThinkPad-W541:~/Desktop/big data project/spark_tweet2/project$ python3 ./app.py
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
```

Terminal for spark_app.py:

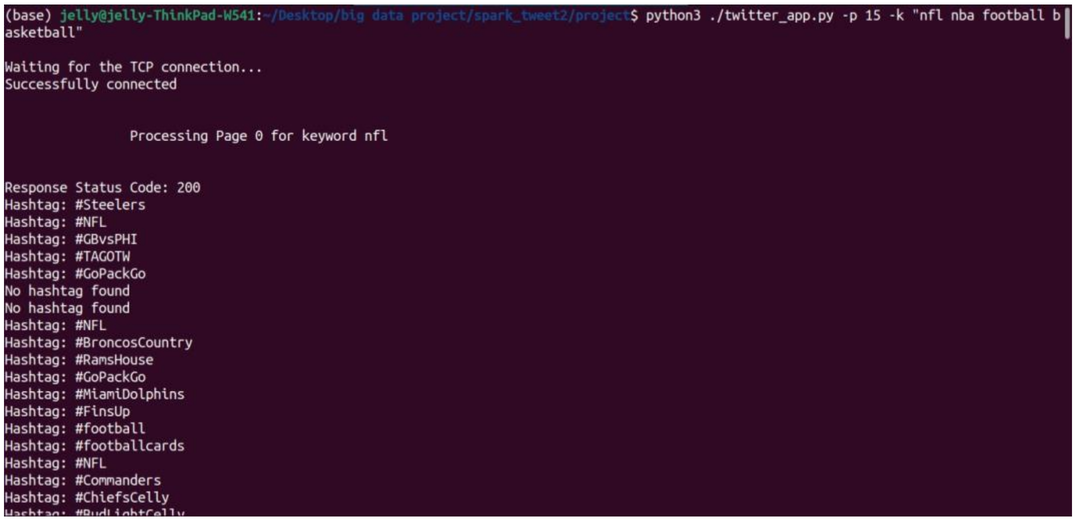


```
(base) jelly@jelly-ThinkPad-W541:~/Desktop/big data project/spark_tweet2/project$ export PYSPARK_PYTHON=python3
(base) jelly@jelly-ThinkPad-W541:~/Desktop/big data project/spark_tweet2/project$ export SPARK_LOCAL_HOSTNAME=localhost
(base) jelly@jelly-ThinkPad-W541:~/Desktop/big data project/spark_tweet2/project$ python3 ./spark_app.py
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/12/06 17:27:20 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
{'hashtag': '#Thehobby', 'count': 1}
{'hashtag': '#BidenBorderCrisis', 'count': 1}
{'hashtag': '#TAGOTW', 'count': 1}
{'hashtag': '#PrizePicksNBA', 'count': 1}
{'hashtag': '#AlecPierce', 'count': 1}
{'hashtag': '#GBvsPHI', 'count': 1}
{'hashtag': '#Bears', 'count': 1}                                (27 + 8) / 200]
{'hashtag': '#WorldCup', 'count': 1}                             (38 + 8) / 200]
{'hashtag': '#ColtStrong', 'count': 1}
{'hashtag': '#ForTheShoe', 'count': 1}
{'hashtag': '#ChiefsKingdom', 'count': 1}                        (46 + 8) / 200]
{'hashtag': '#RamsHouse', 'count': 1}
{'hashtag': '#BETTING', 'count': 1}
{'hashtag': '#Colts', 'count': 2}
{'hashtag': '#49ers', 'count': 2}
{'hashtag': '#SaferFields', 'count': 1}
{'hashtag': '#TrendingNow', 'count': 1}
{'hashtag': '#Browns', 'count': 1}                               (56 + 8) / 200]
{'hashtag': '#sportscardsforsale', 'count': 1}                   (69 + 8) / 200]
{'hashtag': '#ColtsNation', 'count': 1}
{'hashtag': '#Trump2024', 'count': 1}
{'hashtag': '#sportscards', 'count': 1}                          (81 + 8) / 200]
{'hashtag': '#LionsCelly', 'count': 1}
{'hashtag': '#NFLTwitter', 'count': 1}
{'hashtag': '#PrizePicksNFL', 'count': 1}
{'hashtag': '#AFC', 'count': 1}
{'hashtag': '#FinsUp', 'count': 2}
{'hashtag': '#PrizePicks', 'count': 1}                           (90 + 8) / 200]
{'hashtag': '#RBU', 'count': 1}
```



```
{'hashtag': '#nflteams', 'count': 1}
{'hashtag': '#fantasyfootballtips', 'count': 1}                 (69 + 8) / 200]
{'hashtag': '#FinsUp', 'count': 3}                              (83 + 8) / 200]
{'hashtag': '#fantasybust', 'count': 1}                         (96 + 8) / 200]
{'hashtag': '#footballcards', 'count': 3}
{'hashtag': '#fantasyfootball', 'count': 2}                    (109 + 8) / 200]
{'hashtag': '#KEEPFOXSPORTS', 'count': 18}
{'hashtag': '#NFL', 'count': 28}===========>                   (124 + 8) / 200]
{'hashtag': '#Alabama', 'count': 1}===========>                (135 + 8) / 200]
{'hashtag': '#fantasypro', 'count': 1}===========>             (148 + 8) / 200]
{'hashtag': '#fantasyfootballadvice', 'count': 1}
{'hashtag': '#nfl', 'count': 8}
{'hashtag': '#DenverBroncos', 'count': 1}
{'hashtag': '#week12', 'count': 1}====================>        (163 + 8) / 200]
{'hashtag': '#FanDuel', 'count': 2}
{'hashtag': '#Commanders', 'count': 4}
{'hashtag': '#BroncosCountry', 'count': 3}===============>     (176 + 8) / 200]
{'hashtag': '#football', 'count': 5}=========================>  (189 + 8) / 200]
{'hashtag': '#whodoyoucollect', 'count': 1}
{'hashtag': '#Bills', 'count': 1}
{'hashtag': '#ad', 'count': 2}                                  (38 + 8) / 200]
{'hashtag': '#world', 'count': 1}                              (55 + 10) / 200]
{'hashtag': '#OnePride', 'count': 1}
{'hashtag': '#Browns', 'count': 3}
{'hashtag': '#TakeFlight', 'count': 1}                          (67 + 8) / 200]
{'hashtag': '#RuleTheJungle', 'count': 1}
{'hashtag': '#nrl', 'count': 1}=====>                           (97 + 8) / 200]
{'hashtag': '#BUFvsNE', 'count': 3}====>                       (110 + 8) / 200]
{'hashtag': '#KEEPFOXSPORTS', 'count': 19}
{'hashtag': '#trader', 'count': 1}========>                    (121 + 8) / 200]
{'hashtag': '#NFL', 'count': 31}
{'hashtag': '#ChiefsCelly', 'count': 3}=======>                (137 + 8) / 200]
{'hashtag': '#BudLightCelly', 'count': 4}=========>            (149 + 8) / 200]
{'hashtag': '#LVvsSEA', 'count': 1}==================>         (161 + 8) / 200]
{'hashtag': '#Commanders', 'count': 5}
[Stage 13:===============================================>      (177 + 8) / 200]
```

Terminal for twitter_app.py:

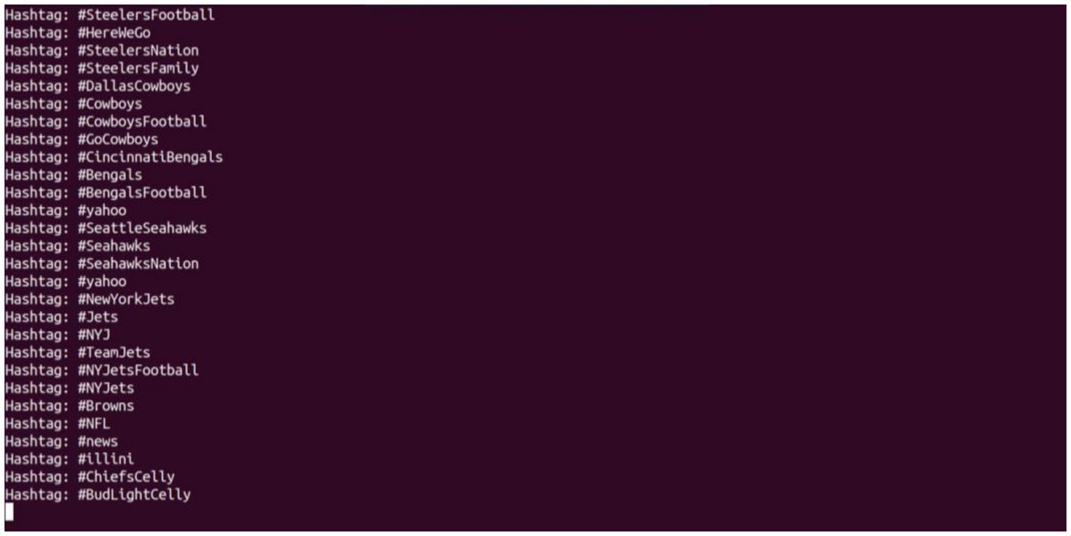


```
(base) jelly@jelly-ThinkPad-W541:~/Desktop/big data project/spark_tweet2/project$ python3 ./twitter_app.py -p 15 -k "nfl nba football b
asketball"

Waiting for the TCP connection...
Successfully connected


                Processing Page 0 for keyword nfl


Response Status Code: 200
Hashtag: #Steelers
Hashtag: #NFL
Hashtag: #GBvsPHI
Hashtag: #TAGOTW
Hashtag: #GoPackGo
No hashtag found
No hashtag found
Hashtag: #NFL
Hashtag: #BroncosCountry
Hashtag: #RamsHouse
Hashtag: #GoPackGo
Hashtag: #MiamiDolphins
Hashtag: #FinsUp
Hashtag: #football
Hashtag: #footballcards
Hashtag: #NFL
Hashtag: #Commanders
Hashtag: #ChiefsCelly
Hashtag: #BudLightCelly
```

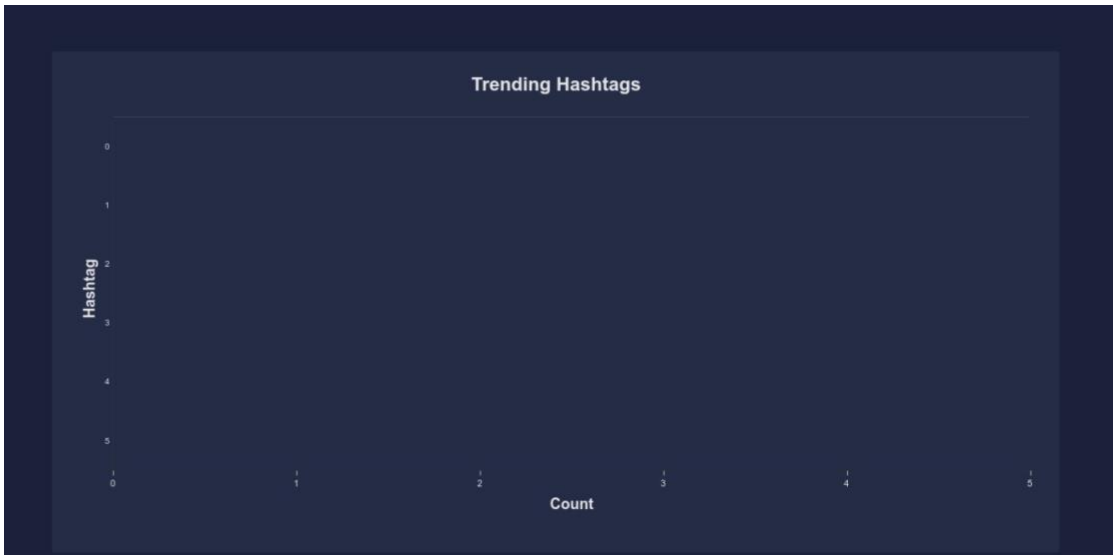


```
Hashtag: #SteelersFootball
Hashtag: #HereWeGo
Hashtag: #SteelersNation
Hashtag: #SteelersFamily
Hashtag: #DallasCowboys
Hashtag: #Cowboys
Hashtag: #CowboysFootball
Hashtag: #GoCowboys
Hashtag: #CincinnatiBengals
Hashtag: #Bengals
Hashtag: #BengalsFootball
Hashtag: #yahoo
Hashtag: #SeattleSeahawks
Hashtag: #Seahawks
Hashtag: #SeahawksNation
Hashtag: #yahoo
Hashtag: #NewYorkJets
Hashtag: #Jets
Hashtag: #NYJ
Hashtag: #TeamJets
Hashtag: #NYJetsFootball
Hashtag: #NYJets
Hashtag: #Browns
Hashtag: #NFL
Hashtag: #news
Hashtag: #illini
Hashtag: #ChiefsCelly
Hashtag: #BudLightCelly
```

Output for visualization:

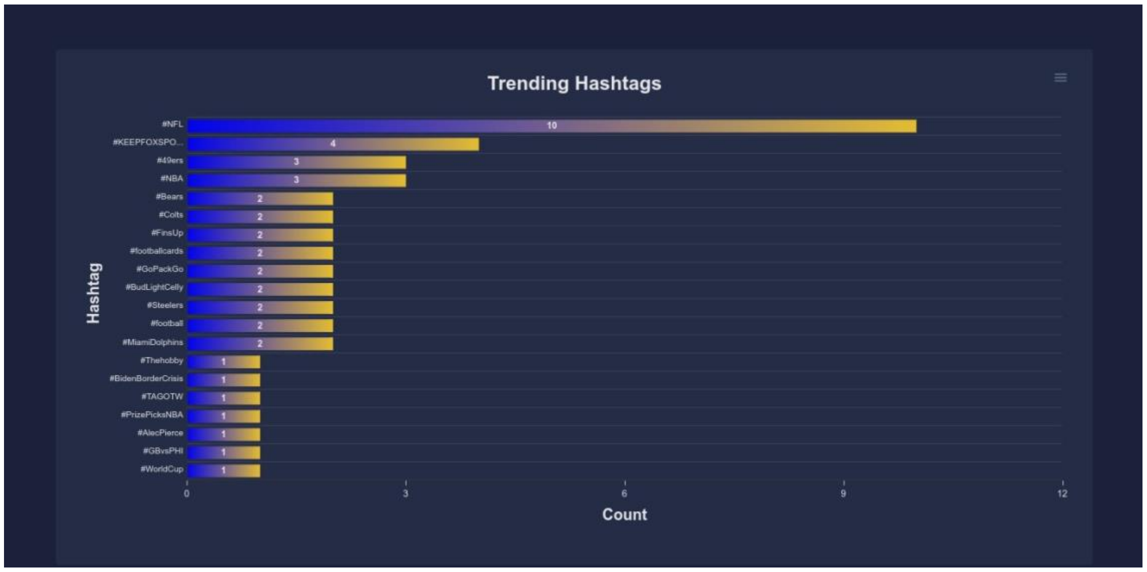
The bar graph dynamically changes with the streaming process.

When the program starts:



Once the data comes in:

From the below graph we can see the number of tweets from the corresponding hashtag. This value constantly changes as more data is being pulled from the twitter API. The graph is dynamic and changes while the program is being executed. It becomes static once the program stops and displays the result. The user can stop the program anytime they want in manual execution and it stops after 3 minutes in automatic execution.



References:

1. https://developer.twitter.com/en/docs/twitter-api
2. https://spark.apache.org/docs/latest/api/python/
3. https://apexcharts.com/docs/installation/
4. https://docs.python.org/3/
5. https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/api-reference/get-tweets-search-stream
6. https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data
7. https://medium.com/analytics-vidhya/exploring-twitter-streaming-data-using-python-and-spark-3f4f189ec660
8. https://developer.twitter.com/en/docs/tutorials/stream-tweets-in-real-time
9. https://www.npmjs.com/package/apexcharts
10. https://www.freecodecamp.org/news/how-to-use-rest-api/