**Real Time Twitter Streaming Data Pipeline**

1. **Introduction**

Social media platforms generate vast amounts of data every day, which can be used for various applications, including sentiment analysis, trend detection, and customer behavior analysis. Twitter is one of the most popular social media platforms, and its data is particularly valuable for real-time analysis. However, analyzing Twitter data can be challenging due to its dynamic and unstructured nature.

In this project, we have developed a Twitter streaming pipeline that allows for the ingestion, processing, and analysis of live Twitter data. The pipeline consists of several components, including Socket, Apache Spark, Hadoop Distributed File System (HDFS), and Apache Hive. The pipeline architecture is designed to handle large volumes of data in real-time and provide insights into Twitter user behavior.

1. **Project Overview:**

**The aim of the project is:**

* Ingest live Twitter data using Socket and Spark Streaming
* Process and transform the data into a format suitable for storage in HDFS
* Store the data in HDFS as Parquet files
* Create an external Hive table on top of the Parquet files
* Analyze and query the data using Hive and Spark SQL
* Create dimensions and fact tables using Hive and Spark SQL

In the following sections, we will describe the architecture and technologies used in the pipeline, provide instructions on how to set up and configure the pipeline, and explain how to use Hive and Spark SQL for data analysis and querying.

1. **Architecture:**

The Twitter streaming pipeline consists of several components that work together to ingest, process, and analyze live Twitter data. The pipeline architecture is designed to handle large volumes of data in real-time and provide insights into Twitter user behavior.

**The main components of the pipeline are:**

1. ***Twitter Streaming API:*** This is the source of the data for the pipeline. We use the Twitter Streaming API to receive real-time tweets that match specific keywords or hashtags.
2. ***Socket:*** the socket was used to receive the twitter data from the twitter Streaming API.
3. ***Apache Spark:*** Spark is a fast and general-purpose cluster computing system that provides powerful data processing capabilities. We use Spark to process and transform the incoming Twitter data into a format suitable for storage in HDFS.
4. ***Hadoop Distributed File System (HDFS):*** HDFS is a distributed file system that allows us to store large amounts of data across a cluster of machines. We use HDFS to store the processed Twitter data as Parquet files.
5. ***Apache Hive:*** Hive is a data warehousing tool that provides a SQL-like interface to Hadoop. We use Hive to create an external table on top of the Parquet files stored in HDFS. This allows us to query the data using SQL and perform data analysis.
6. ***Spark SQL:*** Spark SQL is a Spark module for structured data processing. We use Spark SQL to create and manipulate Dimensions and Fact Tables and Do some Aggregations using SQL queries on the Twitter data stored in HDFS.

The pipeline is designed to work in a distributed environment, with each component running on a separate node or cluster. The following diagram illustrates the architecture of the Twitter streaming pipeline:

|  |
| --- |
| **Twitter Streaming API --> Spark Streaming --> HDFS --> Hive --> Spark SQL** |
| *Architecture of the Twitter streaming pipeline* |

1. ***Data Collection System***

The streaming Twitter data is written in Python and uses the Twitter API to stream real-time tweets containing a specified keyword, in this case, "**Sudan**" in English language. It extracts various tweet and user-related data and sends it over a socket connection to a listening client. First, the required modules are imported, and the Twitter API bearer token is obtained using the auth function, which retrieves the token from an environment variable.

The **create\_headers** function creates the necessary authorization headers for the API request. The **create\_url** function takes the keyword, start and end dates, and maximum number of results as parameters, and returns a tuple consisting of the search URL and query parameters. It includes various tweet and user-related fields to be returned in the API response.The **connect\_to\_endpoint** function takes the search URL, authorization headers, and query parameters as parameters and sends an HTTP GET request to the Twitter API using the requests module. It returns the JSON response as a dictionary.

The code then sets up a socket server to listen for incoming connections. It then enters an infinite loop where it repeatedly makes requests to the Twitter API to stream new tweets containing the specified keyword.In the inner loop, it checks if there are more results available and extracts tweet and user-related data from the response. It then filters and formats the data into a JSON string and sends it over the socket connection.

The code introduces delays between API requests and sends to avoid hitting the Twitter API rate limit and to allow the client to process the received data.Finally, the code updates the start time to the end time of the previous search and repeats the process. The code continues to run until stopped manually or an error occurs.

1. ***Landing Data Persistence***

The first step in this phase is to import the required libraries, including pyspark, SparkSession, functions, and types. Then, a SparkSession object is created, and a **tweet\_schema** is defined, which is the schema of the tweet data to be ingested.

Next, a **tweet\_df** DataFrame is created using the **readStream** method to read streaming data from a socket connection. The DataFrame is then transformed to extract the required fields from the JSON data and to add new columns for date, year, month, day, and hour using built-in functions.

Finally, the streaming data is written to HDFS in the Parquet format using the writeStream method, which is partitioned by year, month, day, and hour. The **awaitTermination()** method is called to keep the application running until the user manually stops it.

It is worth noting that the coalesce method is used to merge all output files into a single file, which can be useful for small datasets. To summarize, this phase is for processes streaming Twitter data and writes it to HDFS in the Parquet format, partitioned by time.

|  |
| --- |
| A screenshot of a computer  Description automatically generated with low confidence |
| *Data Collection System* |

1. ***Landing to Raw ETL***

In this phase HiveQL is used to run queries to extract dimensions from landing data, the first part of the code creates an external table named **twitter\_Data** with columns representing various attributes of a tweet such as ID, text, author ID, retweet count, and others. This table is partitioned by year, month, day, and hour and is stored as Parquet format at the location specified.

The next step is to repair the table using the **MSCK REPAIR TABLE** command. This command is used to repair a table's partitions if they are not automatically detected or if new partitions have been added. Next, a new table called **user\_dim\_raw** is created to store information about users who posted the tweets. This table contains columns such as author ID, followers count, following count, and others. This table is also stored in Parquet format at a specified location. The **INSERT OVERWRITE** command is then used to insert unique records from the **twitter\_Data** table into the **user\_dim\_raw** table. Similarly, a table named **tweet\_dim\_raw** is created to store information about tweets such as ID, text, and other attributes. Unique records from the **twitter\_Data** table are inserted into the **tweet\_dim\_raw** table using the **INSERT OVERWRITE** command.

|  |
| --- |
| A screenshot of a computer  Description automatically generated with medium confidence |
| A screenshot of a computer  Description automatically generated with medium confidence |
| *Landing to Raw ETL* |

1. ***Raw to Processed ETL***

Finally, This stage creates the final fact tables by performing aggregations on the dimensions extracted in the previous stage. The fact table name is twitter\_fact\_raw is created to store aggregated data about tweets. This table contains columns such as tweet ID, author ID, created timestamp, and others. The table is partitioned by year, month, day, and hour, and is stored in Parquet format at a specified location. The INSERT OVERWRITE command is used to insert data into the fact table, which includes tweet counts per hour, day, and month, as well as the number of tweets containing the word "war" per hour. These aggregated values are calculated using window functions such as COUNT(\*) OVER() and SUM() OVER(), and the results are partitioned accordingly.

|  |
| --- |
| **A screenshot of a computer  Description automatically generated with medium confidence** |
| *Raw to Processed ETL* |

1. ***Shell Script Coordinator***

This stage handles the calling of each stage in the pipeline, including the Twitter API script, HiveQL scripts, and the SparkSQL app. The shell script coordinator ensures that the SparkStream is working using crontab job. A script was used to include the paths of other scripts. Then a \*/10 \* \* \* \* / itversity-material/project/final/ twitter\_spark\_hive.sh were added to crontab file.

|  |
| --- |
| A picture containing font, text, typography  Description automatically generated |
|  |

1. **Results**

The pipeline produces several fact tables that provide insights into the Twitter data related to the conflict in Sudan. The fact tables include information on Sudan conflict.