AWS Architecture for Streaming Twitter Data

Group Name: Group E

Semester	Fall 2022
Course Code	CBD 3384
Section	Section 1
Project Title	AWS Architecture for Streaming Twitter Data
Group Name	Group E
Student names/Student IDs	Nikita Mitake(C0825140), Pulkit Kapoor(C0824981), Zarna Priyadarshi(C0829290), Yasin Cinar(C0827907), Deep Jagirdar(C0835259)
Faculty Supervisor	William Pourmajidi

Submission date: ***December 5th, 2022***

Template Prepared by: William Pourmajidi
Last update: Dec 3, 2022

Lambton College | In Toronto

Table of Contents

1. Abstract

Big data and Cloud computing technologies have become mainstream in the past two decades, and are being used by every organization to store, process and analyze large volumes of data. The tools for big data have evolved as well, with organizations using a wide range of big-data technologies for a variety of use cases. Furthermore, with the adoption of cloud technologies, the big-data landscape has also transformed with these organizations moving their big-data operations from on-premises architecture to cloud-native solutions.

This paper deals with implementing a cloud-based big-data pipeline to analyze the stock market tweets on Twitter. The main objective of this project is to use multiple cloud-native services to familiarize oneself with the application and interaction of the services to create a working big-data pipeline.

The big-data pipeline implemented in this project is a batch-processing ETL pipeline, where the data source is a real-time streaming source: the Twitter streaming API endpoint, which allows developers to get live tweets as they are posted to the platform. The project makes use of a variety of cloud services to ingest streaming data, store the data, process the data using Spark distributed computing framework, and store the final results of the analysis in a NoSQL database that is used by front-end dashboard applications. The front-end dashboard application enables the user to change variables to deep dive into the collected data and gain useful insights.

2. Introduction

2.1 Importance of Twitter in stock-market analysis?

To understand the importance of Twitter in stock market analysis, we will first answer a few important questions, like What is Twitter and stock, how they relate, and how the cloud plays an important role in this project.

2.1.1 What is Twitter?

Twitter is a US-based social media company. In the beginning, Jack Dorsey, co-founder of Twitter, planned to create a platform based on SMS services. That's why Twitter had 140

characters limit at first. After all past progress on it, Twitter is one of the leading social media products by keeping people up to date with instant posts and easy interactions

2.1.2 What is stock?

Stock can be defined as the partial ownership of a company. Companies can be listed on the stock markets under some rules and their ownership can be transferred between the market and people in exchange for money. Stocks are valuable assets for investing as much as gold and silver. Based on companies' successes, stock prices may increase and decrease. Due to the volatility, stock markets are under strict regulations to protect investors.

2.1.3 How do they relate, and how it affects the market?

Stock prices are highly coupled with buying and selling demands. If many people are willing to buy a stock, its price will go up. So, how do people decide to buy stock? Everyone is looking for a clue that points to a stock price that will increase soon. Therefore, tweets about stocks are more valuable to make an estimation using people's opinions.

2.1.4 Why do we want to analyze?

In our project, we are aiming to gather estimates about the stock market using technical analysis. It is impossible to be aware of every single tweet of stock for human beings. Beyond gathering tweets, we also aim to make lexical analysis and extract some semantic information such as whether they are positive or negative. This will help to understand the community-level emotion for the top tickers in the NYSE index.

2.2 What is Big Data?

Big Data is an amount of data which is harder to process by traditional methods. There are different aspects of it. The first one is volume. Big Data has a high volume and is mostly unstructured such as tweets in different formats. Secondly, it is created very fast and continuously. Sensor data is an example of velocity. The last one is variety. Variety means, the context of the data can be various, and it is necessary to use different kinds of storage options. Text, images, and video in tweets are good examples of the variety of Big Data.

2.3 Big Data services in Cloud Computing

To meet the needs of a Big Data project, such as Volume, Velocity, and Variety, cloud services provide highly available and scalable solutions. Streaming is useful to handle incoming traffic from various resources. It also helps to store them in multiple locations. There are also storage services available for raw and processed data to make the system always consistent with every step of the process.

2.4 Why use the cloud (AWS)?

Even though there are multiple options for Big Data services in Cloud computing, AWS provides various types of services which cover each of the data processing steps. Furthermore, services are not dependent on a certain programming language or operating system. This flexibility decreases the learning curves for beginners. Another powerful reason is a strong community which has enthusiastic people to create boilerplates, and example projects as an opinion. Overall, AWS is our choice since it is easy to start, has lots of content, and has customer support which is helpful during and after development processes.

2.5 Which AWS cloud services are we using?

We have used a variety of AWS services intending to get familiar with technologies that are used in Big Data projects. The project uses the following tech stack:

- Compute – EC2, Lambda functions, ECS & ECR
- Analytics – Kinesis Firehose
- Big Data Processing – Glue Pyspark ETL
- Data storage – S3, DynamoDB
- Data cataloging – Glue Crawlers, databases, and tables
- Data Query - Athena
- Orchestration – Glue Workflows (for ETL)
- Authorization and Authentication / Security – IAM, Security Groups
- Monitoring - CloudWatch

In addition, the core technologies/libraries used for the development of this project are as follows:

- Languages – Python
- Frameworks / Libraries – Boto3, Pyspark, NLTK, Streamlit, Pandas, Tweepy
- Containerization – Docker
- Source code version control – GIT
- APIs – Twitter API v2

3. Methods

In this project, we have built a cloud-based big-data pipeline to analyze stock-related tweets. We are ingesting streaming Twitter data and processing the same using big-data technologies to create a user-friendly dashboard that can be used to understand the sentiment of stock symbols. For this study, we are collecting data for the top 100 ticker symbols in the NYSE index (reference 1).

3.1 Proposed Architecture

The architecture of the project is based on the ETL methodology that is used very commonly for Big Data pipelines. ETL (**E**xtract, **T**ransform, **L**oad) is about extracting data sources, transforming the data by using business logic, and loading the data into target databases or data warehouses.

Figure 1 shows the overall architecture of the cloud-based ETL pipeline. The overall pipeline consists of **E**xtract, **T**ransform and **L**oad stages which are as follows:

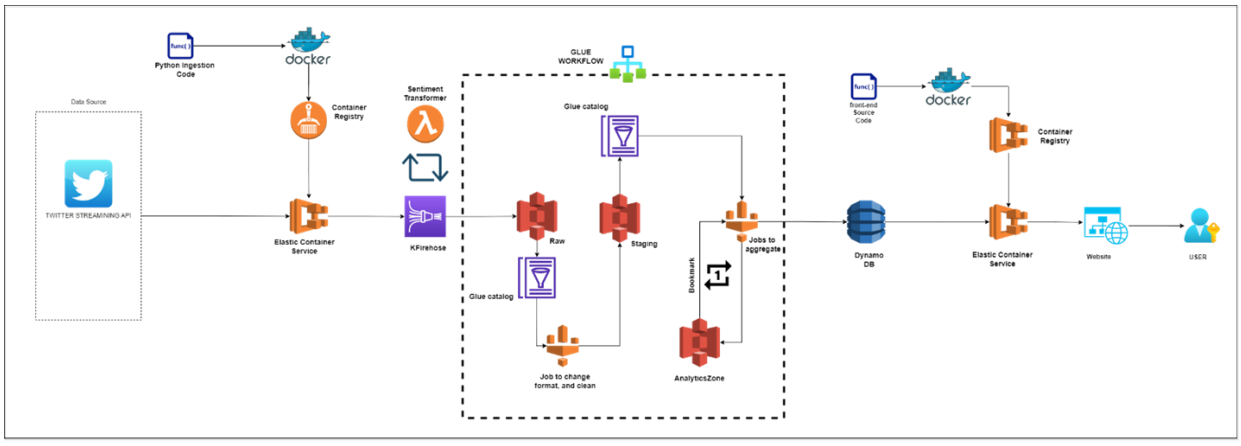

Figure 1: Service and Data Flow

3.1.1 Extract

The extract component of the pipeline consists of ingestion services present in the AWS cloud. The extract stage of the architecture uses AWS services as described below:

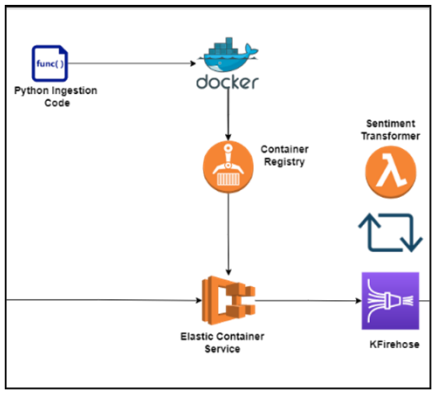

Figure 2: Streaming Data Ingestion

- Elastic Container Registry: Used to store the docker image for Twitter data ingestion code.
- Elastic Container Service: Used to run the docker image within ECR as a persistent container.
- Kinesis Firehose: Used to ingest real-time streaming data from Twitter Fig 2. Streaming Data Ingestion

Figure 2 shows the overall extract stage of the process. We will now deep dive inside the implementation to understand the major design decisions taken for the extract stage –

3.1.1.a Data Ingestion Python Program

The Python ingestion code plays a major role in the project architecture because it is responsible for extracting the streaming data using Twitter API: we are using the Twitter Developer API that allows our stream ingestion code to get real-time tweets on topics subscribed. Since Twitter produces a huge volume of data per second, we are only interested in the project topic which is stock market information. So, the incoming stream is filtered to pass the tweets that contain hashtags or hashtags for the top 100 ticker symbols.

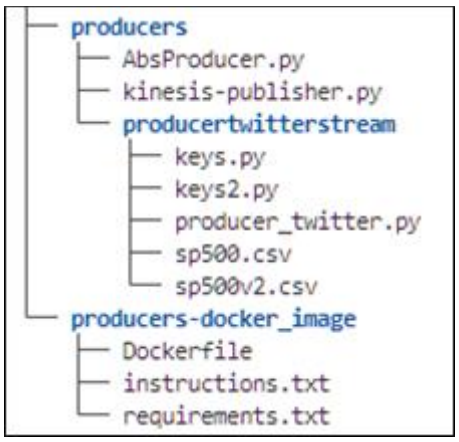

Figure 3: Ingestion Code Structure

Figure 3 shows the high-level design of the tweet extraction python code. The ingestion code has been written to allow any future users of the project to add new data sources by writing producer classes that implement the abstract class *AbsProducer.py*. Also, note that the *sp500v2.csv* contains the static list of tickers we are interested in while fetching the data with Twitter API.

3.1.1.b Elastic Container Service

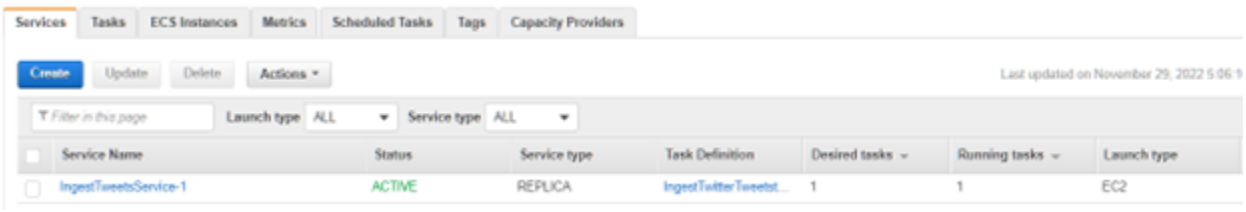

Figure 4: ECS Instance

We are using ECS to run the docker images for the front end as well as the python ingestion program briefly described in part A. The elastic container service allows us to easily deploy containers as well as make sure that a desired count of the container is always running on the available cluster. For the extraction side of the architecture, we wanted to make sure that the

data extraction continues even if there is a temporary disruption such as API errors, connection timeouts, memory errors etc. In these scenarios, the ECS service will try to redeploy the container image without any manual intervention.

3.1.1.c Kinesis Firehose Stream Transformation

Figure 5: Code Ingestion Service in ECS

We are using Kinesis Firehose to process the streaming data sent by the ingestion code and store it in the target S3 location. To process the stream records, we are using lambda transformer which is a feature built into Kinesis Firehose that allows the lambda function to process the records before they are sent to the target data location, which in our case is an S3 bucket.

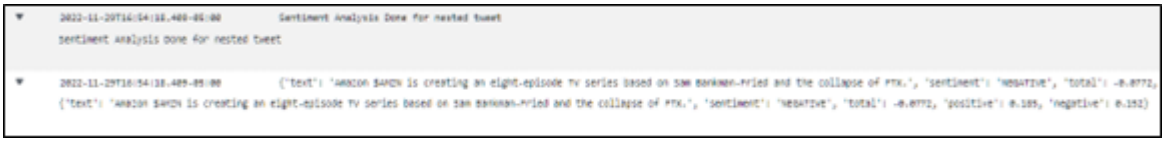

Figure 6: Example of Sentiment Analysis

The stream transformation that takes place is the sentiment analysis of the tweet text. **Figure 6** shows an example of a tweet that was categorized as negative by the transformation function before it landed in the S3 bucket.

We use a simple but effective sentiment analysis algorithm known as Vader Sentiment Analysis. Vader or Valence Aware Dictionary for sentiment reasoning is a model used for sentiment analysis that is sensitive to the polarity (positive or negative) as

well as the intensity of the observed emotion. It makes use of a static mapping of word roots to the emotional intensity which is used to find the average sentiment of a sentence based on the words present in the sentence.

3.1.1.d Transform

The transform component of the pipeline consists of data storage, cataloging and Big Data processing services present in the AWS cloud. The transformation stage of the architecture uses the AWS services as described below.

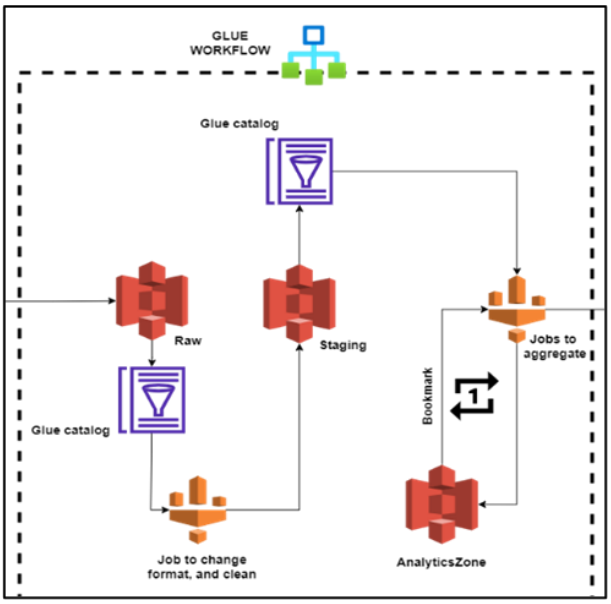

Figure 7: Batch ETL Process Flowchart

- S3: Used to store the data in data lake architecture style
- Glue Crawler: Used to crawl data in S3 to generate hive metastore compatible catalogue
- Glue Pyspark Jobs: Used to run Pyspark jobs to process and transform data in S3
- Glue Job Workflow: Used to orchestrate the batch ETL jobs and crawlers to run on schedule

Figure 7 above shows the complete architecture for the transformation stage of the ETL process. We are using S3 buckets to store data for different stages of the Transformation cycle: Raw, Staging and Analytics. We will be looking at the same in detail in a further section. To understand the transformation step, we will be looking at the key design decisions made for this stage of the ETL:

- **S3 buckets**

We have used cloud storage service S3 to store data at different stages of the ETL cycle. The related architecture as shown in **Figure 7**, makes use of three buckets which are as follows –

Raw Zone:

- This bucket acts as the landing zone and acts as the target bucket for the Kinesis Firehose. Data landing in this bucket is in JSON format
- The data landing in the bucket is partitioned by year-month-day-hour
- Tables contained – sp500v2 (list of stock symbols of interest), raw_tweets_json (raw data from kinesis firehose)

Staging Zone / Stage1

- This bucket holds the same data as the raw zone
- the only difference is that the format of files is compressed parquet to optimize analytical operations
- The data landing in the bucket is partitioned by year-month-day-hour
- Tables contained – sp500v2_parquet, raw_tweets_parquet

Staging Zone / Stage2

- This bucket holds the cleaned data after it is processed from Stage 1 of the staging bucket
- The format of stored data is compressed parquet
- The data landing in the bucket is partitioned by year-month-day-hour
- Tables contained – cleaned_tweets_parquet, spam_tweets_parquet

Analytics Zone

- This bucket contains the data after it is processed from stage 2 tables in the staging
- The format of stored data is compressed parquet
- The tables in this bucket are partitioned by the run_id of the Spark Job that created the tables

- The purpose of storing the results by run_id is to enable incremental processing of stage2 tables, that is we only process new data and use results from the last job run to update the aggregation results

- Glue Crawlers and Databases

The main function of data crawler services is to discover new data in data sources and parse the data to create meta-data about the structure of the data present at the crawler location. We are using Glue crawlers to create and update the tables present in the raw, staging and analytics zone which has been mentioned in section A. By discovering new data, we can process the data incrementally, that is we only process new data as it arrives and does not recompute results. This helps to significantly reduce the job reduce time as well as the operating costs related to the project.

- Pyspark Jobs

To process the data ingested from the ingestion stage, we are using the distributed processing framework called Pyspark: which is a Big Data framework widely used in the industry for Big Data processing tasks. Pyspark can distribute processing tasks across nodes in the cluster and perform in-memory processing. We use Pyspark to perform operations like changing data formats, compressing data, cleaning, and re-partitioning the data, as well as writing to data targets like S3 and DynamoDB.

We have used Glue Jobs to run the Pyspark jobs. Traditionally, to run Pyspark, the first requirement is to set up a computing cluster. Glue Jobs makes the development of ETL jobs easier, as the jobs can be run using a serverless framework. This means that the developer doesn't require to set up and maintain a cluster to run Pyspark jobs. In addition, this has added the benefit of cost saving for workflows that do not use cluster actively, such as batch processing tasks.

The transformation step of the ETL pipeline makes use of three spark jobs which load the data incrementally to perform their respective tasks, which are described as follows:

Pyspark Job1 – Key function: Change Data format

- Source data is S3 raw zone
- Processes the data and converts the format to compressed parquet
- Data target is S3 staging zone – stage1

Pyspark Job2 – Key function: Clean the data

- Source data is S3 staging zone – stage1
- Cleans the data by –
    - Normalizing the data
    - Remove tweets where relevant tickers do not exist
    - Remove spam tweets
- Data target is S3 staging zone – stage2

Pyspark Job1 – Key function: Aggregate the cleaned data

- Source data is S3 staging zone – stage2
- Processes the data to create analytics tables
    - Table 1: contains the summary of tickers at yearly, monthly, daily, and hourly levels
    - Table 2: contains tweet-level data for recent tweets information
    - Table 3: overall summary at a yearly, monthly, daily, and hourly level
- The data target is the S3 analytics zone, and DynamoDB tables

- Glue Workflow

Any ETL job may consist of several steps, that are dependent on each other. In our ETL workflow, we have Glue crawlers and Glue Pyspark jobs working together in tandem to produce the required steps. Glue workflow enables the ETL developer to orchestrate an ETL workflow based on triggers such as events in the cloud, or a fixed schedule. Furthermore, it enables us to monitor all stages of the ETL in one place, to identify failure points.

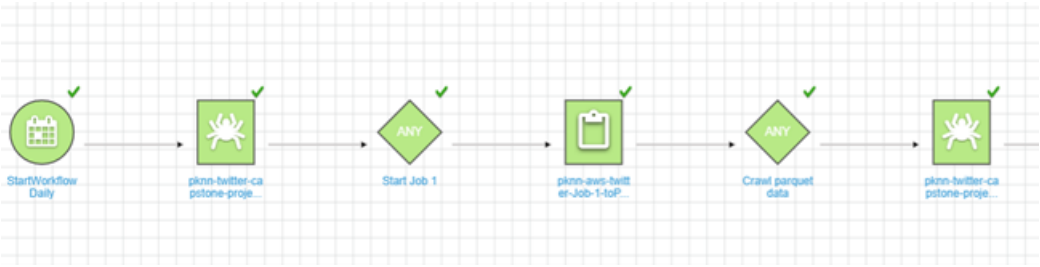

Figure 8: Section of ETL Workflow

Figure 8 shows a subsection of the Glue Workflow that has been created to run the ETL batch operation on schedule at midnight. The ETL pipeline starts running when the trigger conditions are met, and we get the updated results in the analytics tables and the DynamoDB serving layer.

3.1.2 Load and Serve

The load component of the pipeline is responsible for loading / updating the serving layer. We have used DynamoDB for the serving layer, which acts as the primary data source for the front-end user dashboard. DynamoDB is a serverless, distributed and highly scalable key-value document database that is provided by AWS cloud. This is also the key technology used in the serving layer of our architecture.

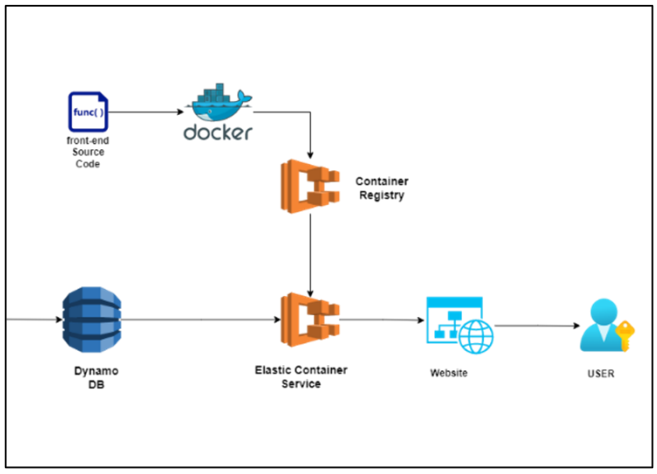

Figure 9: Serving Layer

To support the front-end dashboard queries, we have created three DynamoDB tables, the schema of which has been designed based on the user query patterns that were decided in the planning stage of the project. The design and use of the three DynamoDB tables are as follows -

DDB-Table-1 (Primary Index)

- Partition Key: DetailLevel
- Range Key: TICKER
- Query Patterns
    - 'DetailLevel = 2022' will retrieve summary data for all tickers at a yearly level for the year 2022
    - 'DetailLevel = 2022-11' will retrieve summary data for all tickers at a monthly level for the month Nov 2022
    - 'DetailLevel = 2022-11-04' will retrieve summary data for all tickers at the daily level for the day Nov 4th, 2022
    - 'DetailLevel = 2022-11-04-05' will retrieve summary data for all tickers at the daily level for the day Nov 4th, 2022, at 5 AM
- Usage: Used to create a leaderboard based on the user-selected timeframe

DDB-Table-1 (Secondary Index)

- Partition Key: TickerDetail
- Range Key: TIMESTAMP
- Query Patterns
    - 'TickerDetail = TSLA_MONTHLY' will retrieve summary data for all Tesla reverse chronologically at the month level
    - 'TickerDetail = TSLA_DAILY And TIMESTAMP >= 2022-11-05' will retrieve summary data for all Tesla chronologically (reverse), after the specified date, or range, at a daily level
- Usage: Used to create the sentiment time series for a specific ticker. Allows the user to deep dive into a ticker and understand the sentiment trends

DDB-Table-2

- Partition Key: PARTITION (0-4, used for write sharding to uniformly distribute data)
- Range Key: TIMESTAMP
- Filter Key: TICKER
- Query Patterns
    - ‘PARTITION = 4’ will retrieve tweets in reverse chronological order
    - ‘PARTITION = 4 And TIMESTAMP >= 2022-11-05-15’ will retrieve all tweets in reverse chronological order, after the specified date, or range
- Usage: Used to get recent tweets, overall, or for a particular ticker symbol

DDB-Table-3

- Partition Key: DetailLevel
- Query Patterns
    - ‘DetailLevel = 2022’ will retrieve a summary at the yearly level
    - ‘DetailLevel = 2022-11’ will retrieve a summary at the monthly level
- Usage: Used to create overall metrics for the tweets

4. Results

The product that we have created is a user dashboard, which is a web application. The dashboard application allows the user to deep dive into the tweet summaries by making selections such as the level of detail required (yearly, hourly etc.) or ticker symbol and timeframe of interest. The various elements and functions of the dashboard application are as follows –

Element 1 – Sidebar

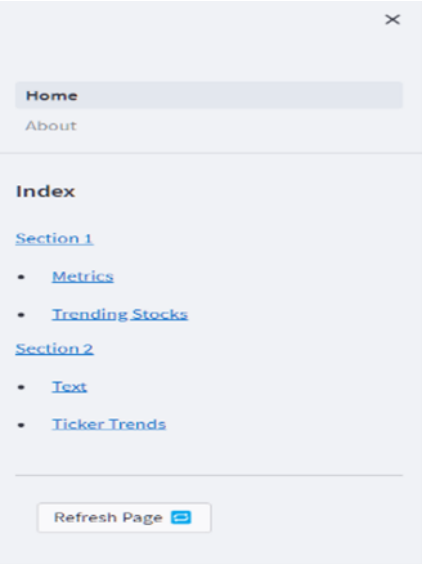

Figure 10: The Sidebar of UI

The first element of design as shown in **Figure 10**, is the sidebar element. The functional value of this element is only to show the index, or the table of contents, and a manual refresh button if the user needs to force refresh the page elements.

Element 2 – User Selections and Overall metrics

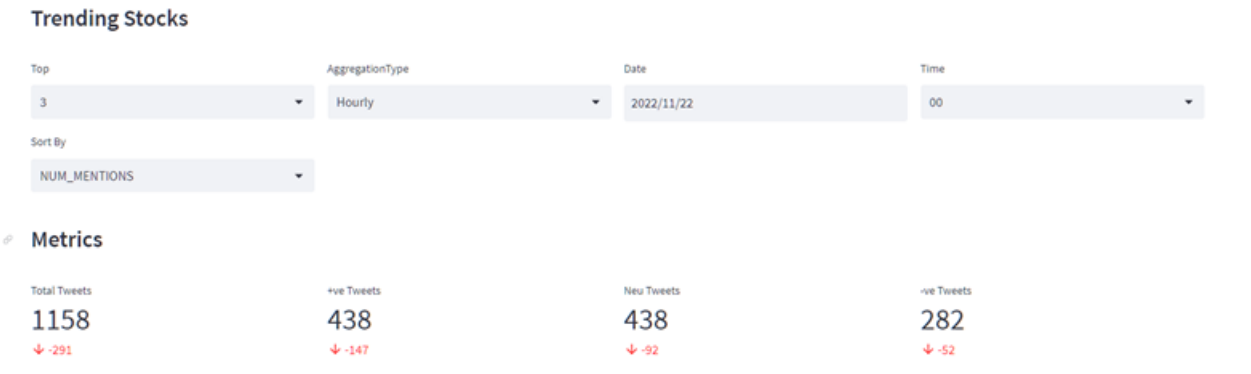

Figure 11: Overall Metrics of Tweets

As can be seen in **Figure 11**, this element of the page, allows the user to select the detail level, for example, the user can select the top trending stocks data on Yearly, Monthly, Daily or Hourly levels. They can see the trending symbols and sort them based on the number of mentions or based on positive, negative, and neutral ratios.

Based on the user selection, the metric data changes dynamically to reflect the total, positive, and negative tweets in the timeframe selected.

Figure 12: Top Trending Tickers

In **Figure 12**, we have the trending stocks chart which is another element of the dashboard. We can see the top 3 or 5, or 15 tickers that match the selection criteria. In addition to the ranking, we can see the proportions of positive, negative, and neutral tweets in each of the top tickers.

Element 3 – Recent tweets analysis

Figure 13: Recent Tweets and Word Cloud

As can be seen in **Figure 13**, the user can make selections, such as fetching the top 100 tweets and filtering the tweets that contain a particular TICKER symbol. The individual tweets listed are highlighted per the sentiment classification of the tweet. We can see a word cloud has been generated for the top tweets. This allows the user to identify any high-level patterns, such as topics or any trendy words in the tweets.

Element 4 – Ticker sentiment trend over time

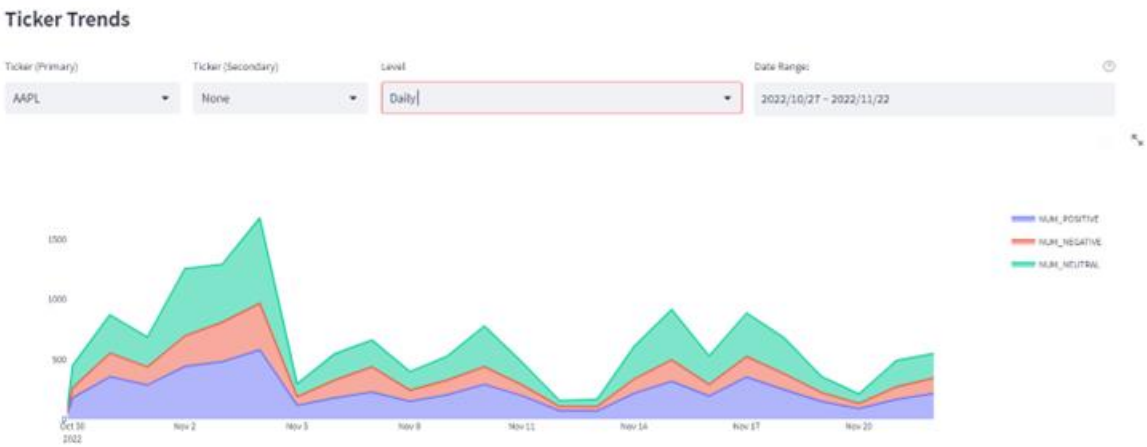

Figure 14: Ticker Sentiment Trends Over Time

In this element of the UI, we help the user understand the trend of a particular TICKER, for example, *AAPL* in **Figure 14**. We can select the level of the data, like monthly, hourly, or daily. The chart depicted is interactive and allows the user to zoom into certain areas of interest in the chart.

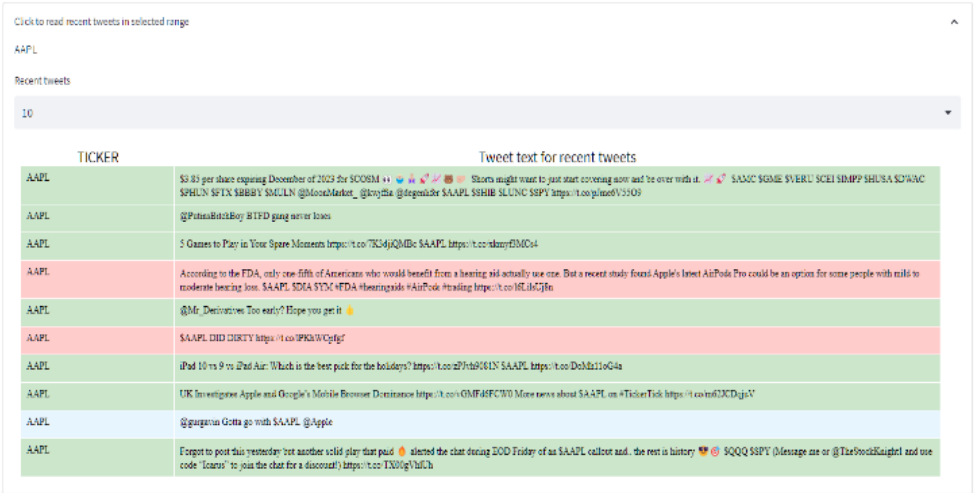

Figure 15: Recent Tweets for Selected Ticker

As a sub-element, we are also displaying to the user the recent tweets for the selected TICKER, in the selected timestamp. The same is shown in figure 14.

In addition to tweets, we show the sentiment by using colour highlights.

Accessing the application

The application dashboard can be accessed at the address **http://3.84.15.57:8080/**

Note – *The website does not use HTTPS and has no domain name, so is blocked on some networks. For example, it is not opening on the Cestar Network.*

5. Conclusions and Future Work

In this project, we have explored key AWS cloud technologies that can be used to build batch ETL processing pipelines. We used the commonly practiced ETL patterns such as incremental loads, and ingesting streaming data. We made use of a NoSQL database to make the application scale as the users grow over time. Some of the improvements or new features that can be added to the current state of the project are as follows –

- Real-Time views – We can introduce Kinesis Streams and Glue Spark Streaming to analyze the stream in real-time and create interesting visualization

- Stock Price data – Stock market price data can be used as an additional data source, which can help show a correlation between stock price and sentiment
- Using API Gateway – Instead of directly querying the DynamoDB tables, a REST API can be created with user authentication
- Upgrading the front-end – The front-end application can be redesigned using modern technologies like React, angular etc.

6. References

- Be A Better Dev. (2020, January 6). *Batch Data Processing with AWS Kinesis Firehose and S3 | Overview*. YouTube. https://www.youtube.com/watch?v=DPT3swb6zgI
- Calderon, P. (2018, March 31). *VADER Sentiment Analysis Explained - Pio Calderon*. Medium. https://medium.com/@piocalderon/vader-sentiment-analysis-explained-f1c4f9101cd9
- *DynamoDB — AWS CLI 1.27.22 Command Reference*. (n.d.). https://docs.aws.amazon.com/cli/latest/reference/dynamodb/index.html
- *Firehose — Boto3 Docs 1.26.22 documentation*. (n.d.). https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/firehose.html
- *S&P 500 Companies - S&P 500 Index Components by Market Cap*. (n.d.). https://www.slickcharts.com/sp500