Capstone Project Weekly Progress Report

Semester	Fall 2022
Course Code	CBD 3384
Section	Section 1
Project Title	Aws Architecture to Process Streaming Data
Group Name	Group E
Student names/Student IDs	Nikita Mitake(C0825140), Pulkit Kapoor(C0824981), Zarna Priyadarshi(C0829290), Yasin Cinar(C0827907),Deep Jagirdar(C0835259)
Reporting Week	11/09/2022 – 17/09/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed in this week)

   - Task 1: Choosing the project
   - Task 2: Deciding on the technologies for executing the project
   - Task 3: Creating the GitHub repo for the project
   - Task 4: Understand the working of Twitter streaming API
   - Task 5: Create code to get live tweets related to our topic of interest

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two page)

   All the tasks listed in point 1 were completed, details are below:

   - We came up with the project title
   - We decided the technologies that we could use for execution: Twitter API, Kafka/AWS Kinesis, and Elasticsearch
   - We created the GitHub repository
   - We used Twitter API to get live tweet data

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

- There are different ways for implementing the solutions. Hence, we had some difficulties in deciding on the technology stack.
- Difficulty in understanding Twitter API due to its complexity
    - Reached request limit when testing
    - We were not getting relevant tweets at first
    - Making sure we only get tweets related to the topic of interest

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

- Finalized the project schema using the technology stack, version 1
- Start getting hands-on experience for some AWS services (kinesis/kafka)

Capstone Project Weekly Progress Report

Semester	Fall 2022
Course Code	CBD 3384
Section	Section 1
Project Title	Aws Architecture to Process Streaming Data
Group Name	Group E
Student names/Student IDs	Nikita Mitake(C0825140), Pulkit Kapoor(C0824981), Zarna Priyadarshi(C0829290), Yasin Cinar(C0827907),Deep Jagirdar(C0835259)
Reporting Week	18/09/2022 – 24/09/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed in this week)

   - Task 1: Creating the project diagram
   - Task 2: Getting familiar with the technology stack
   - Task 3: Create an analyzing script

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two page)

   All the tasks listed in point 1 were completed, details are below:

   - We created a diagram which shows the technologies we will use and data-flow
   - We planned to get deeper knowledge about our technology stack
   - We created a script which catches a Twitter stream

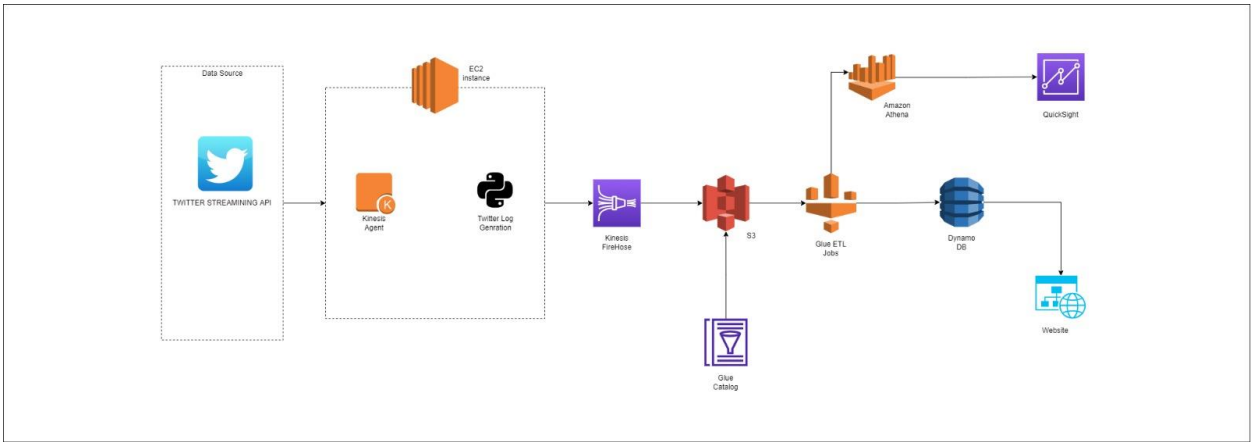


Figure 1: Data flow diagram

As seen in Figure 1, the main data resource is Twitter. AWS Kinesis will listen to the Twitter data stream and the Python script will group tweets as positive/negative. Kinesis Firehose will record logs and detect errors while listening to the stream. Tweets, including positive/negative values, will be stored in DynamoDB and they will be visualized by Amazon Athena/QuickSight.

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

    - We didn't have a certain decision on when to run analyzing script while catching the stream. This pushed us to test every option to find the optimal performance.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

    - Based on our experimental examples, we will code the prototype.
    - We will try to detect bottlenecks and do brain-storm sessions to fix them.

Capstone Project Weekly Progress Report

Semester	Fall 2022
Course Code	CBD 3384
Section	Section 1
Project Title	Aws Architecture to Process Streaming Data
Group Name	Group E
Student names/Student IDs	Nikita Mitake(C0825140), Pulkit Kapoor(C0824981), Zarna Priyadarshi(C0829290), Yasin Cinar(C0827907),Deep Jagirdar(C0835259)
Reporting Week	25/09/2022 – 01/10/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed this week)

   We tried to create a prototype which handles streaming data and stores it in an S3 bucket. We have used AWS solutions to achieve this objective.

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two pages)

First Yasin and Pulkit created a delivery stream by using Amazon Kinesis Data Firehose. This delivery stream is responsible for handling the instant data stream. To test this delivery stream, we have created a simple python class which produces a JSON data stream (Figure 1).

```
class TweetProducer(object):
    def __init__(self):
        self.text = "Created by pulkit!"
        self.id = 12312213
        self.region = "America"
        self.timestamp = 1664604088
        self.created_at = 1664604088
        self.tags = ['tag-a', 'tag-b']

    def get_tweet(self):
        return json.dumps({
                 "twitter-id": 12313,
                 "text": "tweet adsgaaf pulkit",
                 "tags": ['APPL', 'GOOGL'],
                 "timestamp" : 1664604088,
                 "created_at" : 1664604088,
                 "region" : "America"
                 })

    def generate_tweets(self):
        while True:
            sleep(4)
            yield self.get_tweet()
```

Figure 1: Sample stream class

To handle the data stream produced by the class in Figure 1, we created another class which takes data from TweetProducer class and transfers it into the AWS Kinesis Firehose instance (Figure 2).

```
class KinesisFirehoseDeliveryStreamHandler(logging.StreamHandler):

    def __init__(self):
        # By default, logging.StreamHandler uses sys.stderr if stream parameter is not specified
        logging.StreamHandler.__init__(self)

        self.__firehose = None
        self.__stream_buffer = []

        try:
            self.__firehose = boto3.client('firehose')
        except Exception:
            print('Firehose client initialization failed.')

        self.__delivery_stream_name = "twitter-delivery-test"

    def emit(self, record):
        try:
            msg = self.format(record)

            if self.__firehose:
                self.__stream_buffer.append({
                    'Data': msg.encode(encoding="UTF-8", errors="strict")
                })
            else:
                stream = self.stream
                stream.write(msg)
                stream.write(self.terminator)

            self.flush()
        except Exception:
            self.handleError(record)

    def flush(self):
        self.acquire()

        try:
            if self.__firehose and self.__stream_buffer:
                self.__firehose.put_record_batch(
                    DeliveryStreamName=self.__delivery_stream_name,
                    Records=self.__stream_buffer
                )

                self.__stream_buffer.clear()
        except Exception as e:
            print("An error occurred during flush operation.")
            print(f"Exception: {e}")
            print(f"Stream buffer: {self.__stream_buffer}")
        finally:
            if self.stream and hasattr(self.stream, "flush"):
                self.stream.flush()

            self.release()
```

Figure 2: KinesisFirehoseDeliveryStreamHandler class

This class in the Figure 2 basically have 3 functions: __init__(), emit(), and flush().

__init__() function creates the basic variables, such as firehose, and stream_buffer variables, and initializes the firehose connection. During this initialization, if any error occurs it will catch them and throws it as a text.

emit() function, puts the data into stream_buffer variable. In case of a missing firehose connection, Python's logging class uses its default connection and prints the data as a log text.

The last function for this class is **flush().** It is responsible for delivering data and cleaning the stream_buffer variable. It uses a transaction approach. Before delivery, it locks the stream_buffer variable to prevent new data writing operations. After the lock, it starts delivering the process which transfers data in the stream_buffer variable. After delivery is completed, the stream_buffer variable is cleared and the transaction on the stream_buffer variable is unlocked.

On the other side of our delivery stream instance, we used an S3 instance as a data lake. For this part, Nikita created an S3 bucket and she connect this storage with the Firehose delivery system. We store all coming stream data in S3 before analyzing as seen in Figure 3.

Name	Type	Last modified	Size	Storage class
twitter-delivery-test-2-2022-10-01-18-16-24-2924d1bf-9a05-3a23-8fd3-a081ba045b62	-	October 1, 2022, 14:20:50 (UTC-04:00)	16.9 KB	Standard
twitter-delivery-test-2-2022-10-01-18-20-45-ffecda05-ba0a-3e76-8fdf-af93017632d2	-	October 1, 2022, 14:21:47 (UTC-04:00)	3.4 KB	Standard
twitter-delivery-test-2-2022-10-01-18-21-38-42b60649-dfd4-34a4-92b9-d80e0c5110e7	-	October 1, 2022, 14:22:41 (UTC-04:00)	3.1 KB	Standard
twitter-delivery-test-2-2022-10-01-18-22-26-2320c653-92a9-36b6-80e8-76a1955c2136	-	October 1, 2022, 14:23:29 (UTC-04:00)	3.4 KB	Standard
twitter-delivery-test-2-2022-10-01-18-23-18-4f86b791-5ad4-3a08-9cc9-571fbd515b1c	-	October 1, 2022, 14:24:20 (UTC-04:00)	3.4 KB	Standard
twitter-delivery-test-2-2022-10-01-18-24-10-9f4b4764-798a-3e64-ba13-d7ffef9f2c05	-	October 1, 2022, 14:25:12 (UTC-04:00)	3.4 KB	Standard
twitter-delivery-test-2-2022-10-01-18-25-03-44dc37a1-5398-36c4-ad02-56b057b8b207	-	October 1, 2022, 14:26:05 (UTC-04:00)	3.1 KB	Standard
twitter-delivery-test-2-2022-10-01-18-25-51-eba68227-06a3-3623-844c-d23b8eac536f	-	October 1, 2022, 14:26:53 (UTC-04:00)	1.3 KB	Standard

Figure 3: S3 data-lake

To be able to analyze the streaming data we need a Database solution. We had to create a database to group and categorize the data in the S3 instance. Before creating a database, we decided to use a crawler which is gonna help us to retrieve and transform it into a structural database format. Deep had about this need and came up with an AWS solution: AWS Glue Crawlers. It scans whole folders, from the start folder we decide, and groups them. It gives the name of the folders to tables. So, we have a simple structural projection of the files on the S3 instance.

Zarna searched about AWS visualization solutions: Quicksight and Athena. She is working on creating a visualization example using these technologies. Based on these examples, we are gonna decide which one is more suitable or has better performance for our analyzed data.

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

We didn't face an issue while creating simple examples using the technologies we've found. However, It was a problem to handle streaming data to prevent loss. The structure looks a bit complicated because we didn't want to lose any data either receiving or analyzing.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

For next week, Yasin will complete the database side using DynamoDB or Elasticsearch. Zarna will create simple examples on AWS Athena/Quicksight and demonstrate them. Pulkit and Nikita will decide on the data structure and they will build it as similar to the Twitter stream data structure. They will make changes on AWS Firehose and S3 side if it is necessary. Deep will try to project the tables on AWS Glue Crawlers into our database solutions: DynamoDB and Elasticsearch.

Capstone Project Weekly Progress Report

Semester	Fall 2022
Course Code	CBD 3384
Section	Section 1
Project Title	Aws Architecture to Process Streaming Data
Group Name	Group E
Student names/Student IDs	Nikita Mitake(C0825140), Pulkit Kapoor(C0824981), Zarna Priyadarshi(C0829290), Yasin Cinar(C0827907),Deep Jagirdar(C0835259)
Reporting Week	03/10/2022 – 09/10/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed this week)

Last week, we created a dummy stream, which follows the data structure we created, and successfully handled the stream using AWS Firehose. To feed this stream, we created a simple Python script which produces JSON objects. After connecting the stream with a Firehose connection, we created an S3 instance to store bulk data. This storage instance is used as a data lake to be sure that all data coming from the stream is successfully stored.

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two pages)

This week, we spent more time on data structure with the whole team. We worked on alternative data sources considering the time frequencies of updates.

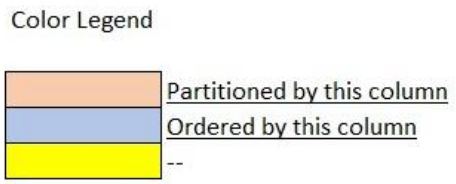


Figure 1: Order&partition indicators as colors

As seen in ***Figure 1***, we used different colours to show which column is a partition or an order column

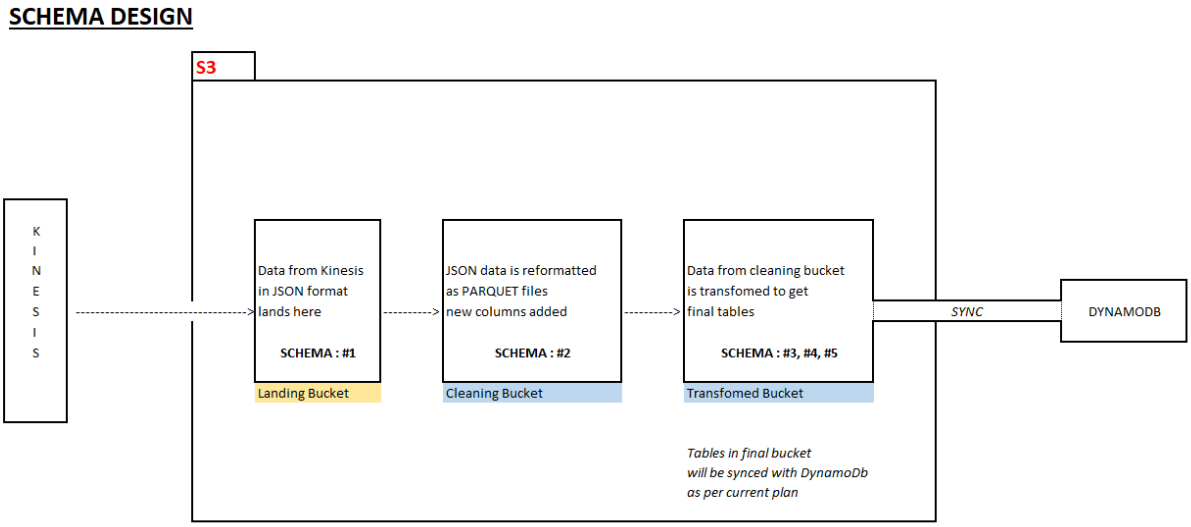


Figure 2: Data flow in S3

We have decided to categorize data first in S3 instances. We are planning to create S3 instances for each process: formatting and cleaning. Data sources may have different data formats, so we will update the data structure to make all data constable. After that, we will create tables using cleaned and formatted data. Before using database solutions, we will apply these changes in other S3 instances **(Figure 2)**.

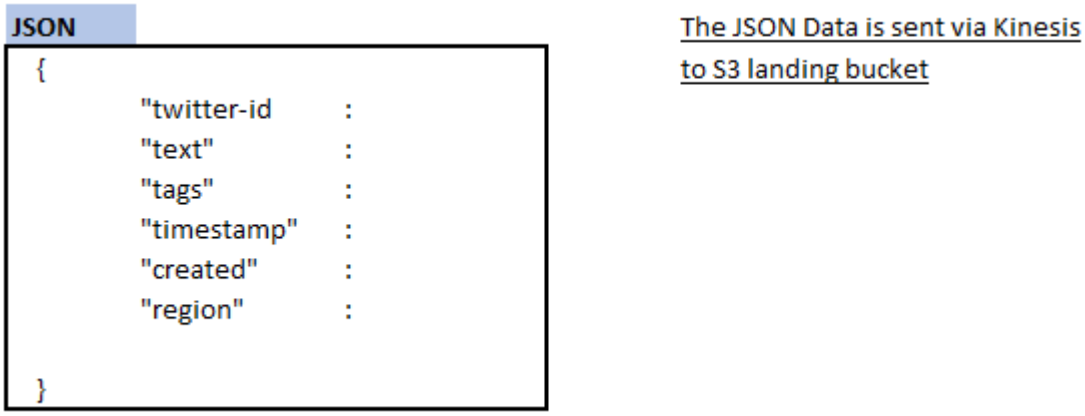


Figure 3: JSON format in the stream

Figure 3 shows the selected data format for the Twitter API data stream. It may differ from further iterations. However, this JSON structure meets the needs regarding group and analysis phases in charts.

FORMAT - PARQUET

stock-list	tweet_text	time-stamp	Year	Month	Week	Day	Hour	Minute
{'AAPL', 'GOOGL'}	aabra jabra dabra	00-00-00 00H:00:00	2022	2	12	4	13	26

Figure 4: Table format for Tweets

As seen in ***Figure 4***, all columns can be used for sorting purposes. 'stock-list' column also provides group tweets. Therefore, our visualization elements can be filtered based on selected stock lists. Even if there is a time-stamp column, we will store every element of the dates to increase query performance.

Date	Hour	Stock cashtag	#mentions	% Positives	T-1-1
Date	**Day**	**Stock cashtag**	**#mentions**	**% Positives**	**T-1-2**
Date	**Week**	**Stock cashtag**	**#mentions**	**% Positives**	**T-1-3**
Date	**Month**	**Stock cashtag**	**#mentions**	**% Positives**	**T-1-4**
Date	**Year**	**Stock cashtag**	**#mentions**	**% Positives**	**T-1-5**

Figure 5: Trending stocks per hour (alternative data source)

Stock cashtag	time-stamp desc	tweet text

Figure 6: Recent tweets

Figure 5 and Figure 6 show the partition keys for different table structures. We will use them to make each row unique and identical. They are primary keys or key pairs. In ***Figure 5,*** there are 5 different time sequences to filter table results. In this case, results could be filtered as hourly, daily, weekly, monthly and, yearly.

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

In terms of difficulties, we didn't face any major issues for the current development duration. However, it was challenging to decide on data structure and use technologies which are efficient and applicable. Our main goal is to create a consistent and available system. Technologies which provide the CA(CAP theorem) approach are highly diverse, so sometimes it is hard to decide not to face any lost or inconsistent data.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

In terms of data-structure decisions, next, we will continue our previous assigned duties this week. Yasin will create&sync DynamoDB and ElasticSearch instances. Zarna will create simple examples on AWS Athena/Quicksight and demonstrate them. Pulkit and Nikita will decide on the trigger functions (Lambda) which will keep data flow alive between S3 instances. Deep will try to project the tables on AWS Glue Crawlers into our database solutions: DynamoDB and Elasticsearch.

Capstone Project Weekly Progress Report

Semester	Fall 2022
Course Code	CBD 3384
Section	Section 1
Project Title	Aws Architecture to Process Streaming Data
Group Name	Group E
Student names/Student IDs	Nikita Mitake(C0825140), Pulkit Kapoor(C0824981), Zarna Priyadarshi(C0829290), Yasin Cinar(C0827907),Deep Jagirdar(C0835259)
Reporting Week	10/10/2022 – 16/10/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed this week)
   Last week, we decided on data structure and storage. We will create 3 S3 buckets which will be used as cold storage. Also, we created a basic JSON format to manipulate the incoming stream data. After formatting the stream data, we will store them in the tables which we agreed on.

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two pages)

   This week, we worked on a couple of activities. First, we brainstormed to finalize the design for the real-time charts of the application. To actualize that, we explored new AWS services that are more compatible with the use case of delivering more real-time data to the front end. As a result of our research, we explored the usage of the AWS API gateway to create serverless web sockets. The serverless web sockets can be used to broadcast data over to the front-end socket listeners as soon as new updates are available at the backend

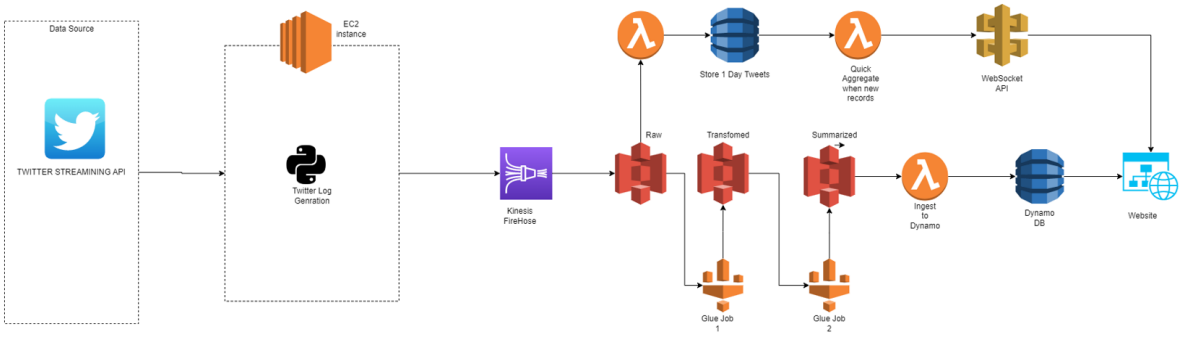


Figure 1: Updated Service Infrastructure

In summary, we decided not to use Kinesis Agent, instead using just boto3 client to put records over to AWS Firehose. We are still using 3 S3 buckets, for landing transformed and aggregated data. Based on some research, we are using AWS Lambda functions to ingest records into DynamoDB after the results are aggregated in final tables. We are also using another AWS Lambda function to generate rolling results for the last 10 minutes window, yet to decide the exact logic

Other than this, we completed the implementation for streaming live Twitter data to the raw bucket in S3.

```
pulkit42041@Pulkit-dell:~/AwsStreamingDataPipeline$ tree kinesisFirehose/ -I __pycache__
kinesisFirehose/
└── producers
    ├── AbsProducer.py
    ├── kinesis-publisher.py
    └── producertwitterstream
        ├── keys.py
        ├── producer_twitter.py
        └── sp500.csv
```

Figure 2: Code Structure

We have defined all the files related to Kinesis in the same directory by the name Kinesis Firehose as seen in ***Figure 2***. Instead of making the code for just one producer, which in our case is Twitter API, we have made sure to write the code so that we can also add more producers in the future if the need arises.

```python
if __name__ == "__main__":
    # logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s")
    # logging.config.dictConfig(config)
    # logger = logging.getLogger(__name__)

    tweet_producer = TweetProducer()

    handler = KinesisFirehoseDeliveryJsonStreamHandler()
    handler.register_producer_stream(tweet_producer, "test")
    handler.run()
```

Figure 3: Main Function

Figure 3 shows the main piece of the code that spins up all processes and sends results to Kinesis.

For the TweetProducer, we have enforced a strict API using Abstract Base Class, with the methods - connect_source, start_stream and read1. All future producers will implement a similar API which allows for adding more features without modifying the current code.

The handler class can attach multiple producers and publish the stream outputs to the specified Firehose streams.

We will not go through the granular implementation. The code can be reviewed in the GitHub repo for the project: https://github.com/k-pulkit/AwsStreamingDataPipeline

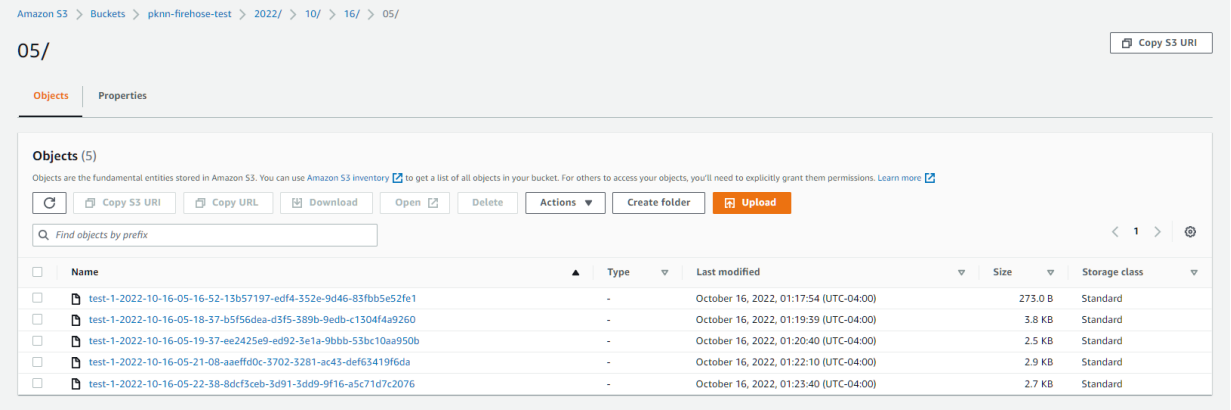


Figure 4: S3 bucket with the Firehose Data

Schema (9)

View and manage the table schema.

Filter schemas

1

#	Column name	Data type	Partition key	Comment
1	text	string	-	-
2	tweet_type	string	-	-
3	sensitive	boolean	-	-
4	created_at	string	-	-
5	hashcashtags	array	-	-
6	partition_0	string	Partition (0)	-
7	partition_1	string	Partition (1)	-
8	partition_2	string	Partition (2)	-
9	partition_3	string	Partition (3)	-

Figure 5: Crawler Schema

We also created a crawler to infer the schema for the ingested data, to query using S3 (***Figure 5***).

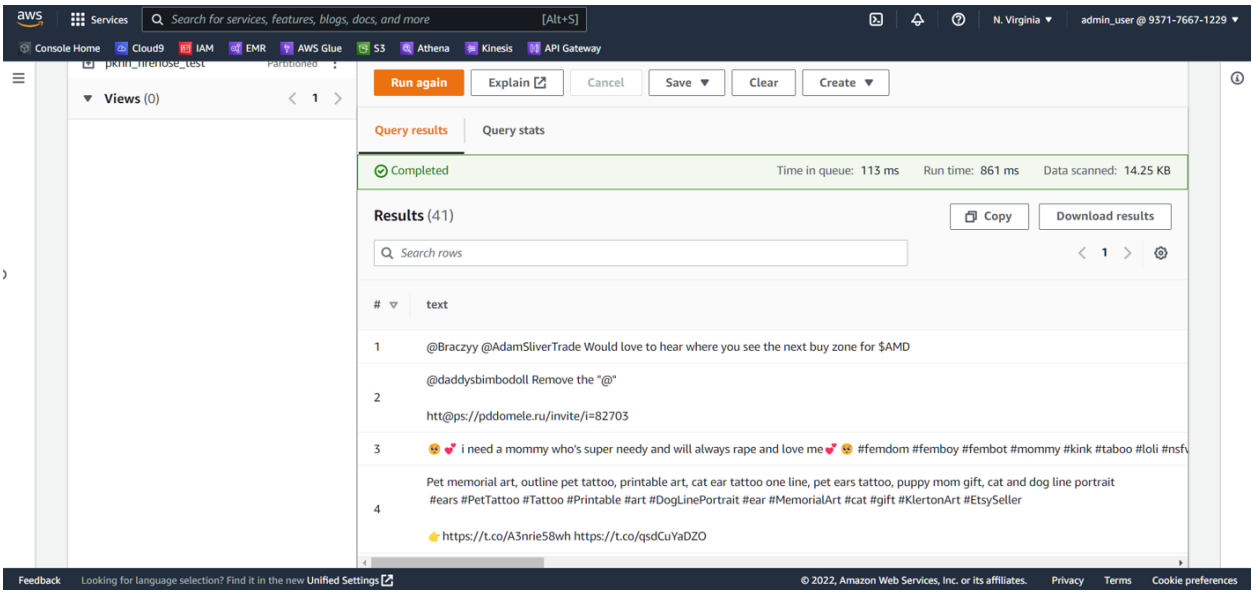


Figure 6: AWS Athena Query

Finally, we were able to query the results using Athena (***Figure 6***).

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

Almost in each meeting, we share our knowledge and discuss solutions we have found. However, this makes us more confused. Every time we decide on a solution, it is changed after a discussion. The reasons behind this issue are meeting our goals and keeping our infrastructure simple so it can be used for real-life problems. Besides that, we worked all together last two weeks for brainstorming sessions. Thus, we didn't have any specific tasks individually.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

For the next week, we will focus on two different infrastructures: S3 buckets' management as a cold storage and database solutions for real-time visualization. For real-time visualization, we decided to use a web socket. We will use a web socket on a serverless architecture like other solutions we have used.

Capstone Project Weekly Progress Report

Semester	Fall 2022
Course Code	CBD 3384
Section	Section 1
Project Title	Enter the title of your capstone project
Group Name	Group E
Student names/Student IDs	Nikita Mitake(C0825140), Pulkit Kapoor(C0824981), Zarna Priyadarshi(C0829290), Yasin Cinar(C0827907),Deep Jagirdar(C0835259)
Reporting Week	17/10/2022 – 23/10/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed in this week)

last week, we had decided to focus on two different infrastructures: S3 buckets' management as a cold storage and database solutions for real-time visualization. For real-time visualization, we had decided to use a web socket.

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two page)

In this reporting week, we did not work on the decided tasks. The week was spent on learning about the technology that we are going to use for transformations (which is the task we wanted to complete).

AWS GLUE

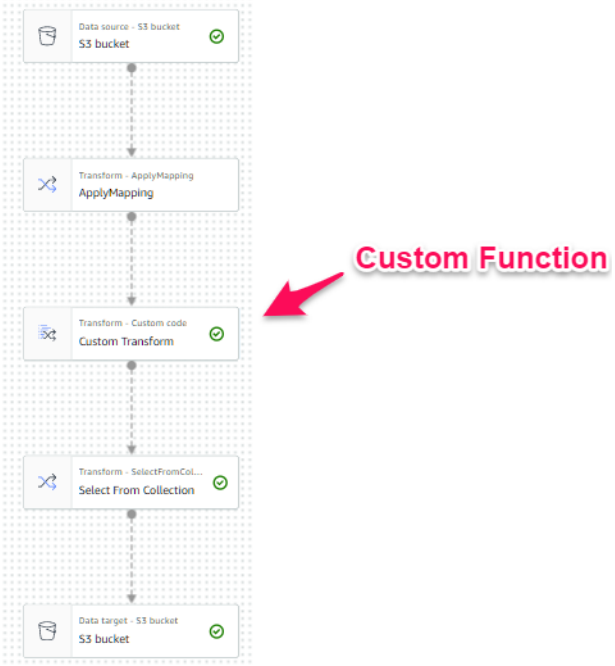
We worked on testing the AWS service called Glue. We tested the service on demo data, where we created a basic job and workflow to change data formats and repartition the files. Using this service we faced issues as the technology is new to us.

Code we used on the demo data

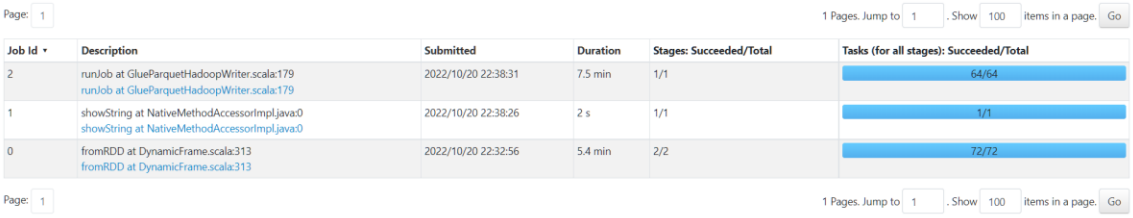
```
import boto3
```

```
from pyspark.sql import functions as f

s3 = boto3.resource('s3')
# List 3 objects from the s3 bucket pknn-github/landing/
x = None
for obj in s3.Bucket('pknn-
github').objects.filter(Prefix='landing/ghactivity').limit(3).all():
x = obj
print(obj)

bucket_name, key = x.bucket_name, x.key
filepath = '/home/glue_user/workspace/jupyter_workspace/demo.json.gz'
bucket = s3.Bucket(bucket_name)
bucket.download_file(key, filepath)

df = spark.read.json(filepath)
# Create the new columns to partition by
df.limit(3)\
.withColumn("year", f.date_format(f.substring(f.col('created_at'), 1, 10), "yy"))\
.withColumn("month", f.date_format(f.substring(f.col('created_at'), 1, 10), "MM"))\
.withColumn("date", f.date_format(f.substring(f.col('created_at'), 1, 10), "dd"))\
.show()
```

Glue Visual script



We also learnt setup and usage of Spark history server for job monitoring. Below is screenshot of completed jobs.



Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

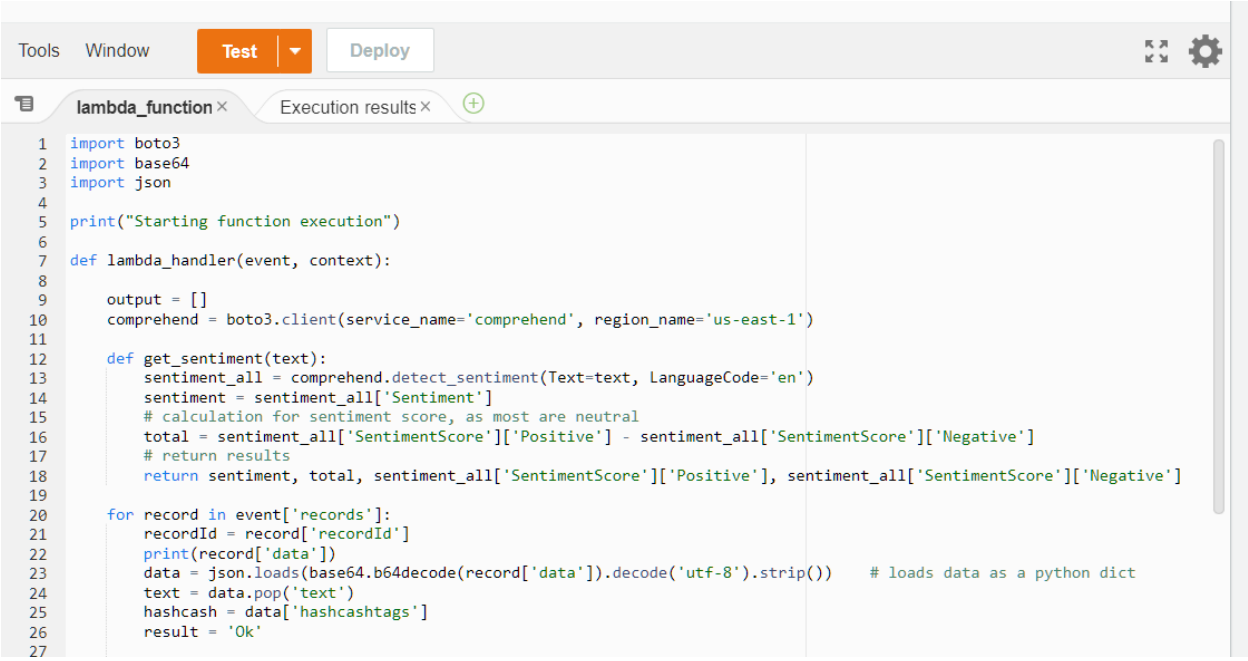
Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2	runJob at GlueParquetHadoopWriter.scala:179 runJob at GlueParquetHadoopWriter.scala:179	2022/10/20 22:38:31	7.5 min	1/1	64/64
1	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2022/10/20 22:38:26	2 s	1/1	1/1
0	fromRDD at DynamicFrame.scala:313 fromRDD at DynamicFrame.scala:313	2022/10/20 22:32:56	5.4 min	2/2	72/72

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

AWS COMPREHEND

We integrated AWS Comprehend service to get the sentiment for the tweets on the fly, before sending them to S3. We did this by using the transform functionality of Kinesis Firehose.

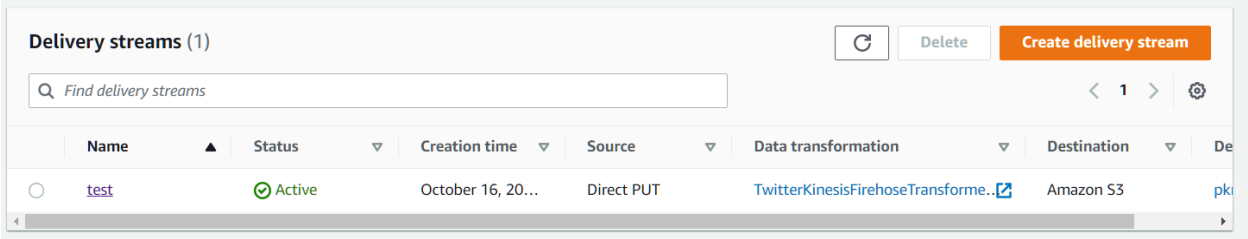
LINK - https://github.com/k-pulkit/AwsStreamingDataPipeline/blob/main/kinesisFirehose/kinesisLambdaTransformer/lambda_handler.py



Tools Window Test Deploy

lambda_function × Execution results ×

```
1  import boto3
2  import base64
3  import json
4
5  print("Starting function execution")
6
7  def lambda_handler(event, context):
8
9      output = []
10     comprehend = boto3.client(service_name='comprehend', region_name='us-east-1')
11
12     def get_sentiment(text):
13         sentiment_all = comprehend.detect_sentiment(Text=text, LanguageCode='en')
14         sentiment = sentiment_all['Sentiment']
15         # calculation for sentiment score, as most are neutral
16         total = sentiment_all['SentimentScore']['Positive'] - sentiment_all['SentimentScore']['Negative']
17         # return results
18         return sentiment, total, sentiment_all['SentimentScore']['Positive'], sentiment_all['SentimentScore']['Negative']
19
20     for record in event['records']:
21         recordId = record['recordId']
22         print(record['data'])
23         data = json.loads(base64.b64decode(record['data']).decode('utf-8').strip())    # loads data as a python dict
24         text = data.pop('text')
25         hashcash = data['hashcashtags']
26         result = 'Ok'
27
```



Delivery streams (1) — Delete — Create delivery stream

Find delivery streams — 1

Name	Status	Creation time	Source	Data transformation	Destination	De
test	Active	October 16, 20…	Direct PUT	TwitterKinesisFirehoseTransforme…	Amazon S3	pk

By making use of this service, we will not have to perfom any sentiment analysis during ETL operations, and we can make use of this sentiment for real time graphs also.

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

AWS RDS (Aurora)

In addition to above, we also tested AWS Aurora, however we were unsuccessful and will continue to work on it the following week.

Also, we faced diffuluties setting up the Glue Spark Jobs, as both the language Spark as well as the service was not familiar. It took multiple retries and debugging sessions to make it work.

We faced less issues integrating the AWS Comprehend service, and it was pretty straightforward.

Major Challenge

As we were analysing the incoming tweet sentiments, we noticed an issue. The sentiment analyser is categorizing most of the comments as NEUTRAL, and on reviewing the individual comments, the sentiments make sense. However, this poses a challenge as we cannot draw lots of insights from this type of data.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

We are working in parallel –

1. Get an understanding of technologies to create real time views, plus experiment
2. Complete the batch pipeline to get final aggregated data. Plus service (step functions) to orchestrate the batch jobs (experiment)

Capstone Project Weekly Progress Report

Semester	Fall 2022
Course Code	CBD 3384
Section	Section 1
Project Title	AWS Architecture to Process Streaming Data
Group Name	Group E
Student names/Student IDs	Nikita Mitake(C0825140), Pulkit Kapoor(C0824981), Zarna Priyadarshi(C0829290), Yasin Cinar(C0827907),Deep Jagirdar(C0835259)
Reporting Week	30/10/2022 – 06/11/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed in this week)

   Last week, we decided to get an understanding of the technologies to create real-time views and also experiment with them. Then we aimed to complete the batch pipeline to get the final aggregated data and step functions to arrange the batch jobs(experiment).

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two pages)
   Tasks performed by:

   Pulkit- AWS glue jobs

   Yasin- Real-time analytics and research

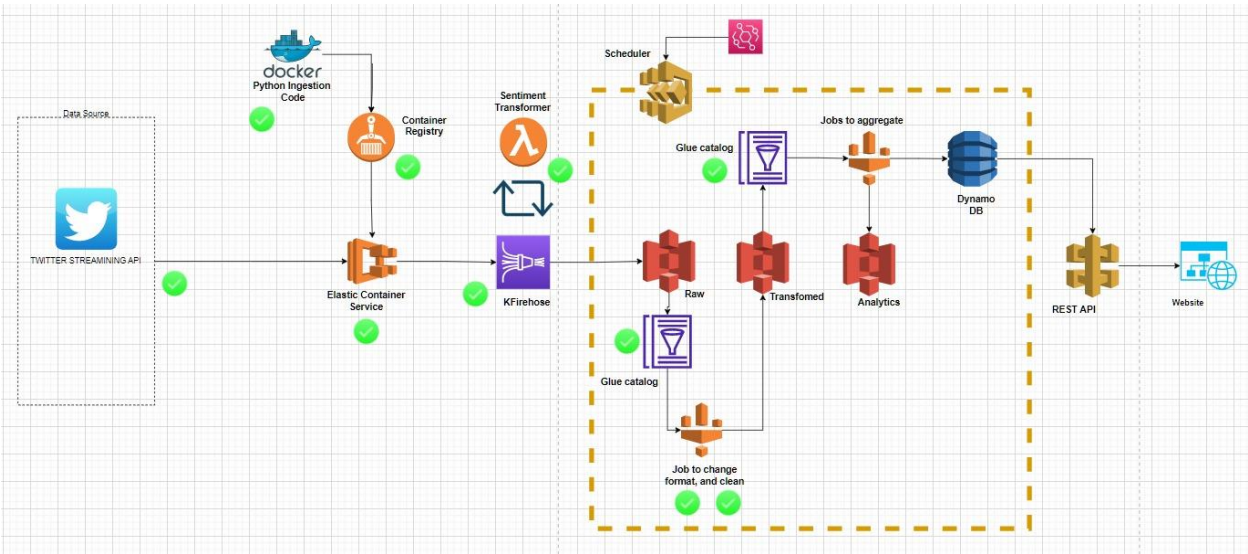
   Zarna- Research for frontend and Report

   Nikita- Report and work on the API gateway

   Deep- Backup and Readme update

Overview of all the components:

Note:- Green check indicates the implemented components.



AWS GLUE

We worked on developing and testing the ETL jobs in the AWS service called Glue. We used the Glue jobs, to transform the raw data first into an optimized parquet file format, and then clean the data for further processing. We tested each of the individual Glue jobs and then created a workflow to execute them in sequence.

We build the two glue jobs, the summary is as follows -

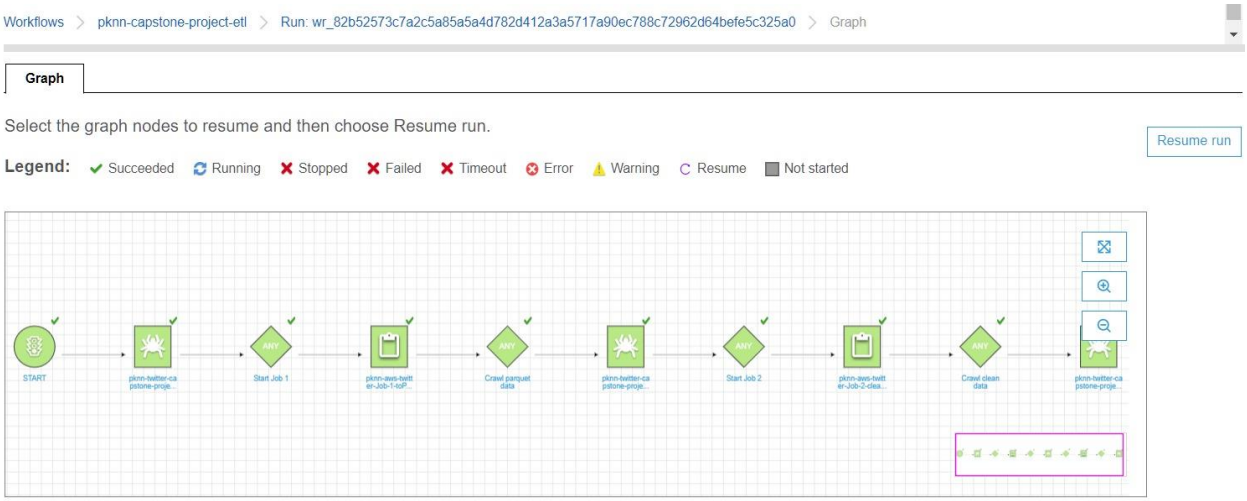
Job 1:

```
"""
Name - Job-1
GlueJobName - pknn-aws-twitter-Job-1-toParquet
Description - This python script makes use of pyspark and glue api
to read the data from input files located in data catalogs in AWS Glue Tables and
and loads the data in incremental fashion in parquet format.py

This script requires that the Glue catalog tables are refreshed before running the script, as
only the partitions present in the catalog are considered for processing.
"""
```

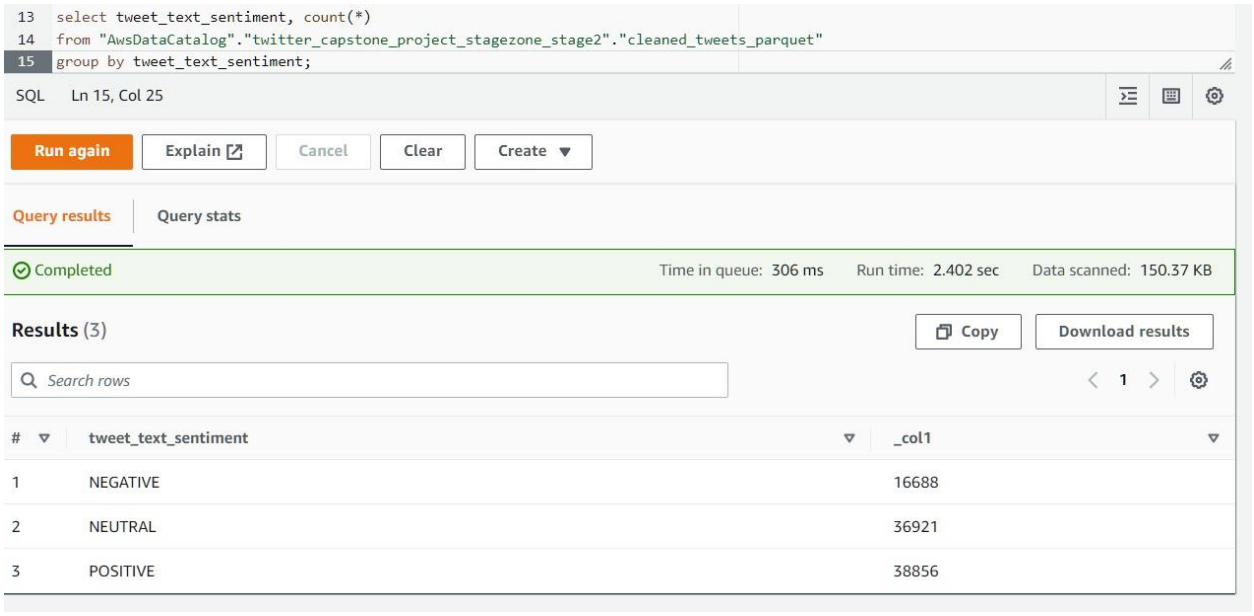
Job 2:

```
"""
Name - Job-2
GlueJobName - pknn-aws-twitter-Job-2-cleanData
Description - This python script makes use of pyspark and glue api
This data is to be executed on the stage1 data, after it is crawled.
The purpose of the job is to clean the data, by combining the information from raw tweets with the sp500 file.

We only are considering the original tweets for analysis, the replies are only used to add extra information in the original tweets
Objective 1 - Normalize the data, remove the nested structure
Objective 2 - Remove tweets when the hashtags and cashtags are not relevant
Objective 3 - Remove the spam tweets
Objective 4 - Save the cleaned data files to stage2 of the staging tables
Result - We can use these cleaned tables to create the aggregated tables in the final bucket, called analytics bucket



This script requires that the Glue catalog tables are refreshed before running the script, as
only the partitions present in the catalog are considered for processing.
"""
```

The graph below depicts the flow of all ETL jobs in AWS Glue working together -



After cleaning the data, we are able to do exploratory analysis by running queries in Athena. To give an example, the below query output shows the count of tweet sentiments from the cleaned tweets.



Simultaneously, we are working on real-time analytics and API creation using AWS API Gateway. Also, some basic project documentation and backup-related activities were completed this week.

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

1. Difficulties with incremental data processing of the Twitter stream, with is the issue working with Glue bookmarks
2. We did not face any technical difficulties but we faced some minor challenges in team activities.
    a. As work activities are not evenly distributed and few ideas are getting scratched after trial and error efforts are not properly distributed.
    b. We are trying to work on this collectively and taking everyone's opinion into account.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

We are trying to work on AWS Glue job -3 for analytics of the data collected. We will use Job-3 to load the final tables in the S3 bucket as well as the DynamoDB tables, which we are using in conjunction with the API gateway for our front-end website. Furthermore, we are also working on real-time analytics and APIs for data visualization, to produce a few near-real-time metrics.

Capstone Project Weekly Progress Report

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)
Reporting Week	06/11/2022 – 13/11/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed in this week)

Last week, we decided to work on AWS Glue job-3 for analytics of the data collected and use job-3 to load the final tables in the S3 bucket as well as the DynamoDB tables, which we are using in conjunction with the API gateway for our front-end website. Furthermore, we had also decided working on real-time analytics and APIs for data visualization, to produce few near-real-time metrics.

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two pages)
   Tasks performed by:

   Pulkit - AWS glue job-3 work partially (10% done)

   Pulkit and Zarna – worked on getting data from DynamoDB to API and Report

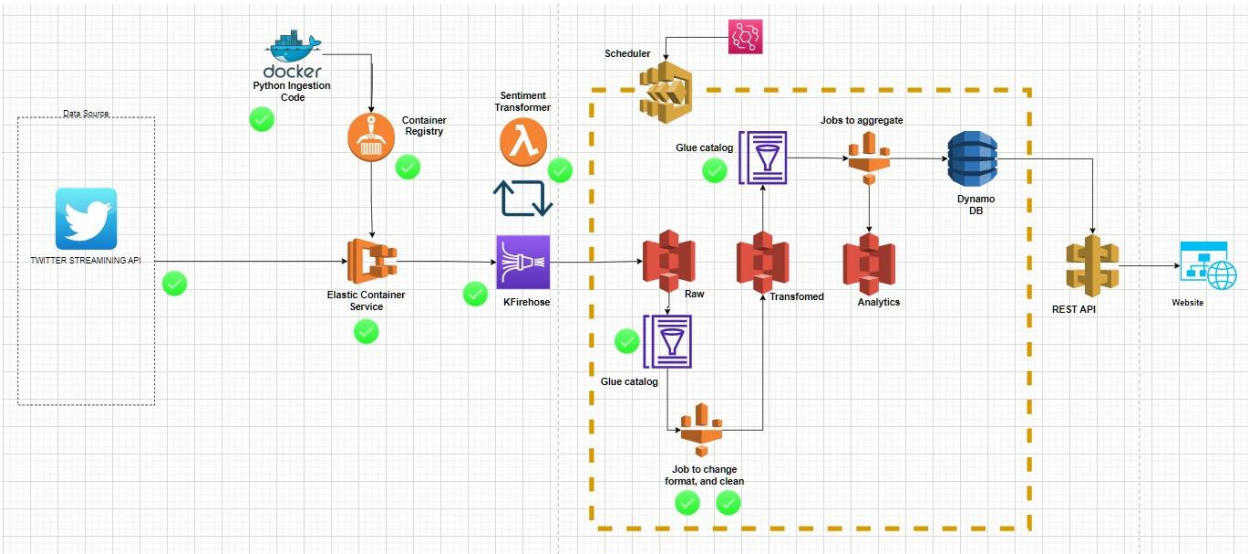
   Yasin- create DynomaDB tables and tested the working of it

   Nikita- she is sick

   Deep- Backup and Readme update

Overview of all the components:

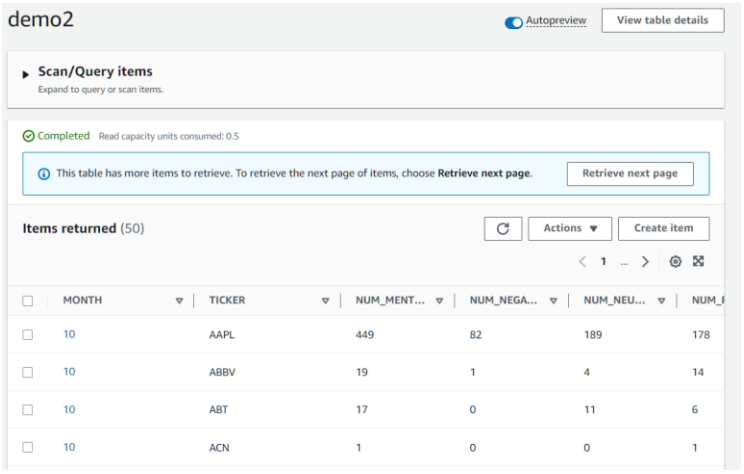
Note:- Green check indicates the implemented components.



AWS GLUE

We continued worked on developing and testing the ETL jobs in the AWS service called Glue. We used the Glue jobs, to transform the raw data first into an optimized parquet file format, and then clean the data for further processing. We tested each of the individual Glue jobs and then created a workflow to execute them in sequence.

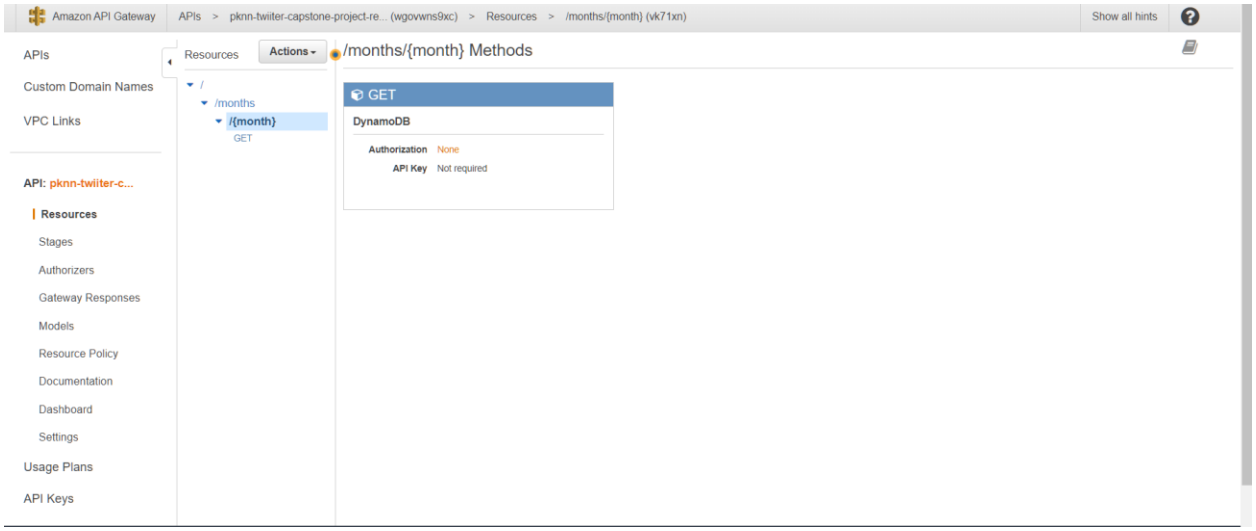
For this week particularly, we worked on Job-3. We are using the cleaned data from Job-2, and we aggregate the results by month to create a summary table as shown below -:



The job-3 will contain multiple transformation (as planned) to aggregate the results at different levels for the different charts in the end product. We have only created one table and connected it with dynamodb and integrated with API Gateway. So as to start creating frontend application.

API Gateway:

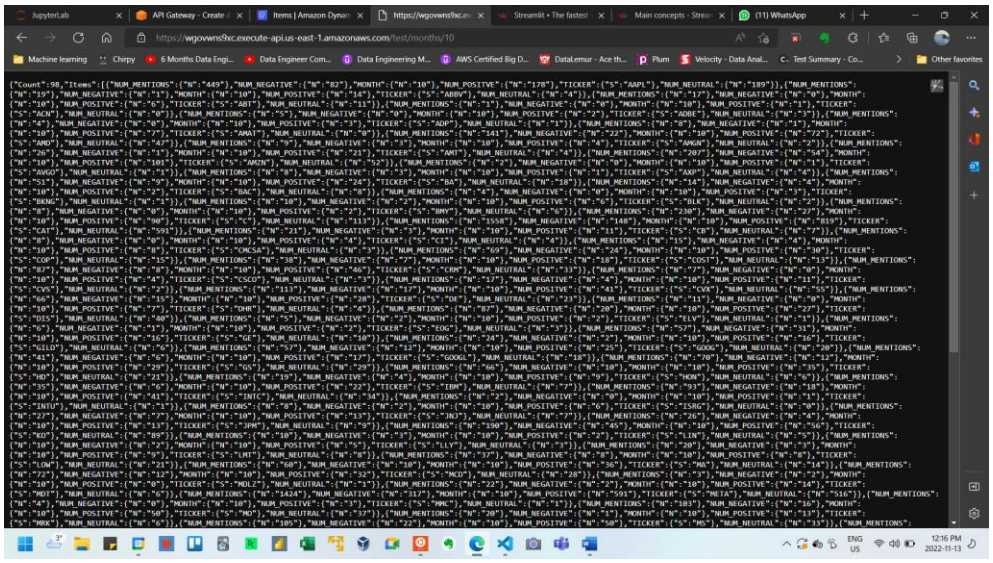
We learnt about the fundamentals of the API Gateway and how to use it as a proxy for reading data for Dynamo-Db without using lambda function in between for access.



We made use of mapping template, to map the endpoint to dynamodb style POST request.

```
{
    "TableName": "demo2",
    "KeyConditionExpression": "#MONTH = :v1",
    "ExpressionAttributeValues": {
        ":v1": {
            "N": "$input.params('month')"
        }
    },
    "ExpressionAttributeNames": {
        "#MONTH": "MONTH"
    }
}
```

As shown above, we have created an endpoint to access the aggregated results by month, that is when we specify a month, the API returns the ticker summary results for the month.



As can be seen in the figure above we were able to access data by using the URL in browser.

We will be using this data to generate ranking and plot the result in our application which is yet to be done. Also, We are planning to work in an iterative fashion where-

step 1: we will aggregate the data and populate the DYnamodb table
step 2: create API Gateway access layer for the populated data
step 3: create the front-end layer with the charts corresponding to the data

This will make sure that we have an end product in case we are not able to complete all the planned tasks.

We have published on changes on the project repo.

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

1. Difficulties with setting up the API gateway to access data from dynamodb table, i.e. understanding and using the VTL language syntax
2. Difficulties loading data to DynamoDB

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

Next week we are planning to work on the front-end to create charts and as mentioned iteratively add more views (which means we will spend time on developing job-3 and api-gateway as well).

Capstone Project Weekly Progress Report

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)
Reporting Week	13/11/2022 – 19/11/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed this week)

Last week, we decided to work on the front end to create charts, how to design, and what to add in the front end and as mentioned iteratively add more views. In addition, we also decided to complete job 3 and the API gateway as well.

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two pages)
   Tasks performed by:

   Pulkit - AWS glue job-3

   Pulkit and Zarna – work on the front end using demo data and report

   Yasin- report

   Nikita- API Gateway

   Deep- Backup and Readme update

CHANGES IN ARCHITECTURE IN THE CURRENT WEEK

Being an agile project, the architecture of the project continues to change. The changes proposed and accepted for the current week are as follows –

1. Schema changes for the Dynamo Tables, to comply with the NoSql constraints, more on which is explained later
2. After cleaning Stage2 in ETL, in the final ETL job, we populate the data directly to DynamoDb for the consumption layer to pick
3. We will directly be running querying data from Dynamo tables from the front end, and not be using an API gateway
    a. The reason for the same is the complexity and learning curve for creating mapping templated in VTL
    b. The second and primary reason is time constraints
        i. Given the final report is to be submitted on Dec 5th, we are planning to complete all implementation by this week to start working on the final project documents, aka, the presentation and report

This week, we worked on the final pieces of the project, each of which is explained below -

AWS GLUE

We were able to complete Job3, which consist of creating final consumption tables for the front-end application. This Job consists of using the cleaned data from stage2 and then aggregating the data at different levels to create summary tables for the output visualizations.

Script Info

```
# type:ignore

"""
Name - Job-3
GlueJobName - pknn-aws-twitter-Job-3-aggregateData
Description - This python script makes use of pyspark and glue api
This data is to be executed on the stage2 data, after it is crawled.
The purpose of the job is to aggregate the data, to produce results that can be displayed to the user

We group the data at different levels to produce the following views -
1. Top Trending Stocks Per Hour / Per Day, Per Week, Per Month
        This table has the following columns
        {Month, count} : [TICKER, count_pos, count_neg, count_neu]
        Partitioned by Month, Sorted by count
2. Recent Tweets for each ticker
        {TICKER+SENTIMENT, timestamp} : [text]
        ==> We want to run a job to cleanup this table, as we only want 1 day of tweets in this table
3. Stock Timeseries
        {TICKER, Month} : [count, count_pos, count_neu, count_neg]

This script requires that the Glue catalog tables are refreshed before running the script, as
only the partitions present in the catalog are considered for processing.
"""
```

Figure 1: Glue Job

We have populated the tables in AWS DynamoDB only, although the initial plan was to comply with the data lake architecture and populate in S3 and then from s3 to DynamoDb. But for the project's use case, we only went ahead with one target database.

The tables created, plus their query pattern, in the DynamoDb are as follows, (next page)

DetailLevel	TICKER	NUM_MENT...	NUM_NEGA...	NUM_NEU...
2022-11-05-01	AAPL	24	5	14
2022-11-05-01	ABT	4	0	1
2022-11-05-01	AMAT	1	0	1
2022-11-05-01	AMD	3	0	1
2022-11-05-01	AMZN	15	3	4
2022-11-05-01	BA	3	0	2

Figure 2: Partition Table 1

PARTIT...	TIMESTA...	CLEANT...	ID	SENTIME...	TEXT	TICKER
2	2022-10-2...	2022 10 30...	158645599...	NEUTRAL	#Other #M...	KO
2	2022-10-2...	based on pr...	158645634...	POSITIVE	#TopTen #s...	GILD
2	2022-10-2...	10 31 WL S...	158645649...	POSITIVE	10/31 WL S...	AMAT
2	2022-10-2...	lost some p...	158645652...	POSITIVE	lost some p...	AMT
2	2022-10-2...	Peanuts ple...	158645737...	POSITIVE	@jedimark...	META

Figure 3: Partition Table 2

	NUM_MENTIONS	NUM_POSITIVE	NUM_NEGATIVE	D_NUM_NEUTRAL	D_NUM_POSITIVE	D_NUM_NEGATIVE	LEVEL	D_NUM_MENTIONS	NUM_NEUTRAL
0	288394	120088	58552	0	0	0	MONTHLY	0	109754
1	18302	7642	3740	6810	6719	3350	DAILY	16879	6920
2	697	333	131	200	264	118	HOURLY	582	233
3	288394	120088	58552	0	0	0	YEARLY	0	109754

Figure 4: Partition Table 3

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

We have difficulties using the API gateway and due to time constraints, we decided to take data directly from dynamodB using the boto3 library. We have to learn the python library straemlit to design the front end.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

Next week we are planning to work on designing the front-end using the Twitter Data that we have collected.

Capstone Project Weekly Progress Report

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)
Reporting Week	20/11/2022 – 27/11/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed in this week)

   Due to the ETL change, we had to update some parts of the structure. First, we added new tables which is responsible the data for visualizing. Secondly, we updated ETL flow and populate data directly to DynamoDB after cleaning Stage 2. As we are getting closer to finalizing the project, we started to create UI part for visualizing the processed data such as creating charts, adding, filters, and connecting data sources to the front-end.

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two pages)

   Tasks performed by:

   - Pulkit and Zarna worked on the front-end design and coding.
   - Yasin created and submitted the weekly report.
   - Nikita and Deep worked on the About section of the front-end and the README.md file of the project that will be pushed to the project's repository.

Overview of all the components:

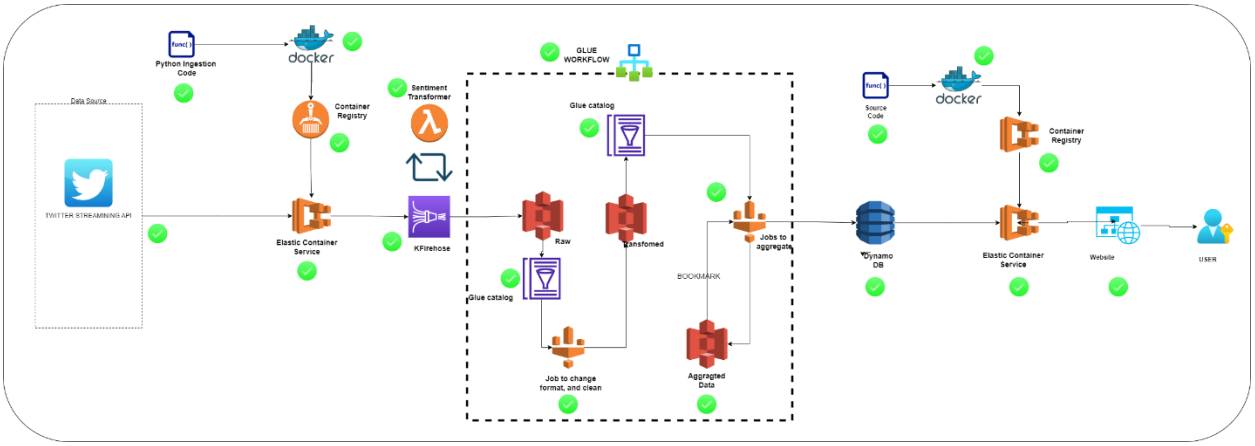


Figure 1: Current Project Architecture (Green checks indicate the implemented components)

Changes - As can be seen in the ***Figure 1***, our front-end application is communicating directly with the DynamoDB service to get the data. We are not using API Gateway as a mediator, but have instead created a FrontEnd IAM user with limited DynamoDb rights permission, to access the data.

FRONT-END CODE AND IMPLEMENTATION

In this particular week, we focused on writing the front-end code. We implemented the front-end in Python using a library called Stream-Lit. This library allows us for creating data science applications and is suited for the requirements of this project. Using this library, we had a learning curve, and we had to spend a good portion of the week on understanding the inner working and design patterns used by streamlit to create applications.

FRONT-END CODE STRUCTURE

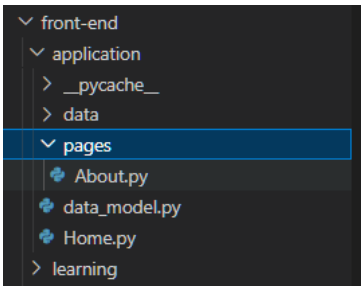


Figure 2: File Structure of the Front-end

The code of the front-end majorly consists of 2 files (as can be seen in ***Figure 2***):

- **Data_models.py**: This file encapsulates the logic for communicating with DynamoDB for fetching the data and then transforming it in formats needed for
- **Home.py**: This file contains the streamlit code for running the application and displaying the front-end elements

FRONT END REPO STRUCTURE

- **Data_models.py**

In this python file, we have created a class "ChartData", which encapsulates the logic for communicating with DynamoDB.

```python
def table1_query1(self, dateTimeStr, ticker=None):
    """
    For Trending Stocks chart
    """
    table1 = self.table1
    base_exp = "DetailLevel = :level"
    base_kv = {":level":dateTimeStr}
    nextToken = None
    if ticker is not None:
        base_exp += " and TICKER = :tick"
        base_kv[":tick"] = ticker
    result = []
    query = partial(table1.query, KeyConditionExpression = base_exp,
            ExpressionAttributeValues = base_kv,
            ScanIndexForward=False
            )
    while True:
        query_r = query(ExclusiveStartKey=nextToken) if nextToken else query()
        items = query_r["Items"]
        result.extend(items)
        nextToken = query_r.get("LastEvaluatedKey", None)
        if nextToken is None: break
    return result
```

Figure 3: DynamoDB Table1 Query Function

As can be seen in ***Figure 3,*** the function *table1_query1*, queries DynamoDB table based on the function parameters. The argument dateTimeStr is the primary key of Table1 in DynamoDB, e.g. 2022 gives the results at yearly level, and 2022-11 will fetch result at monthly level. This table provides data at different levels based on schema design.

FRONT END DESIGN

For the front-end of the website, we have created 2 sections based on the type of chart they contain. The two sections are as follows:

SECTION 1

- **Sidebar**

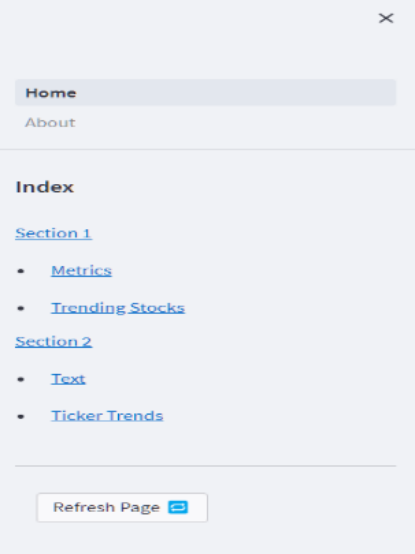


Figure 4: Sidebar for the Front-End

The first element of design as shown by ***Figure 4***, 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 user needs to force refreshing the page elements.

- **Filters and Metrics**

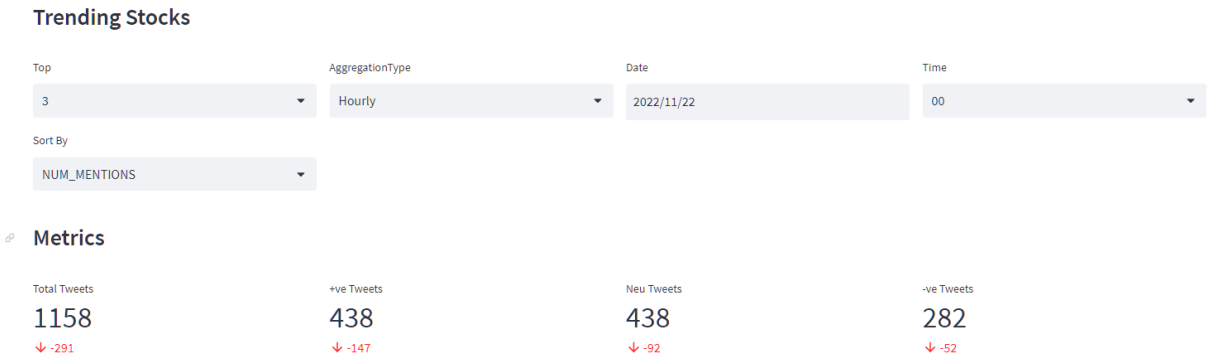


Figure 5: Filters and Metrics for Trending Stocks

As can be seen in figure 5, this element of the page, allows the user to select the detail level, example, user can select to the **top trending stocks** data on Yearly, Monthly, Daily or Hourly level.

They can see the trending symbols and sort it based on 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, negative tweets in the timeframe selected.

Figure 6: Top 3 Trending Stocks

In the other piece of the Section-1, we have the trending stocks chart as shown in the ***Figure 6***. We are able to see the top 3, 5 or 15 tickers, that match selected criterias. In addition to the ranking, we are able to see the proportions of positive, negative, and neutral tweets in each of the top tickers.

SECTION 2

- **Text Data**

In this section of the UI, we wish to help users for inspecting recent tweets, understanding the top tweets and what the users are mentioning based on selections.

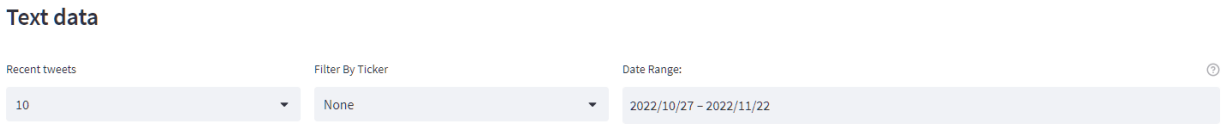


Figure 7: Text Data Filter

As can be seen in ***Figure 7***, users can make selections, such as to fetch the top 100 tweets, and filter the tweets that contain a particular TICKER symbol. The individual tweets listed are highlighted per the sentiment classification of the tweet.

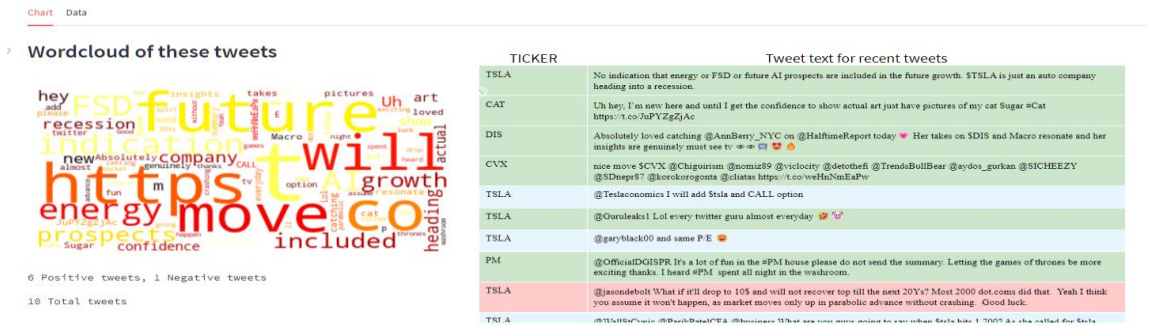


Figure 8: Lexical Analysis

In ***Figure 8***, we can see a wordcloud has been generated for the top tweets. This allows the user to identity any high level patterns, such as topics or any trendy words in the tweets.

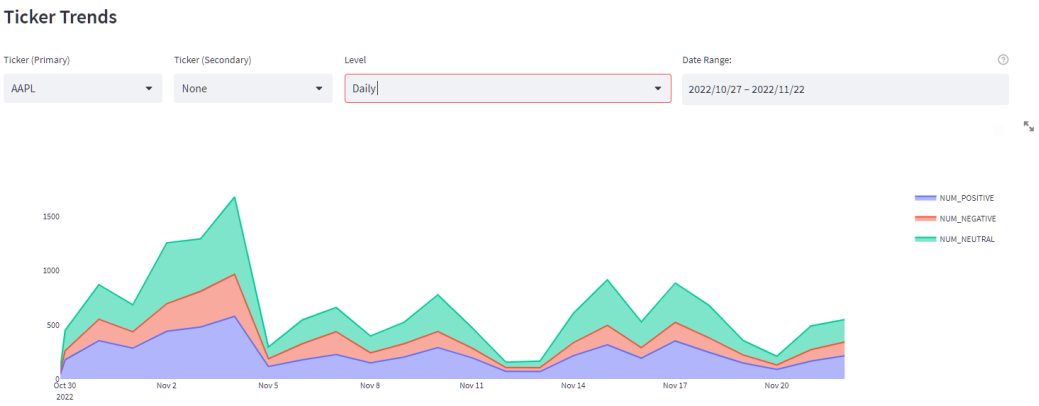


Figure 9: Ticker Trends

In this element of the UI, we help the user understand the trend of a particular TICKER, example *AAPL* in the ***Figure 9***. We are able to select the level of the data, like monthly, hourly, daily. The chart depicted is interactive and allows the user to zoom into certain areas of interest in the chart.

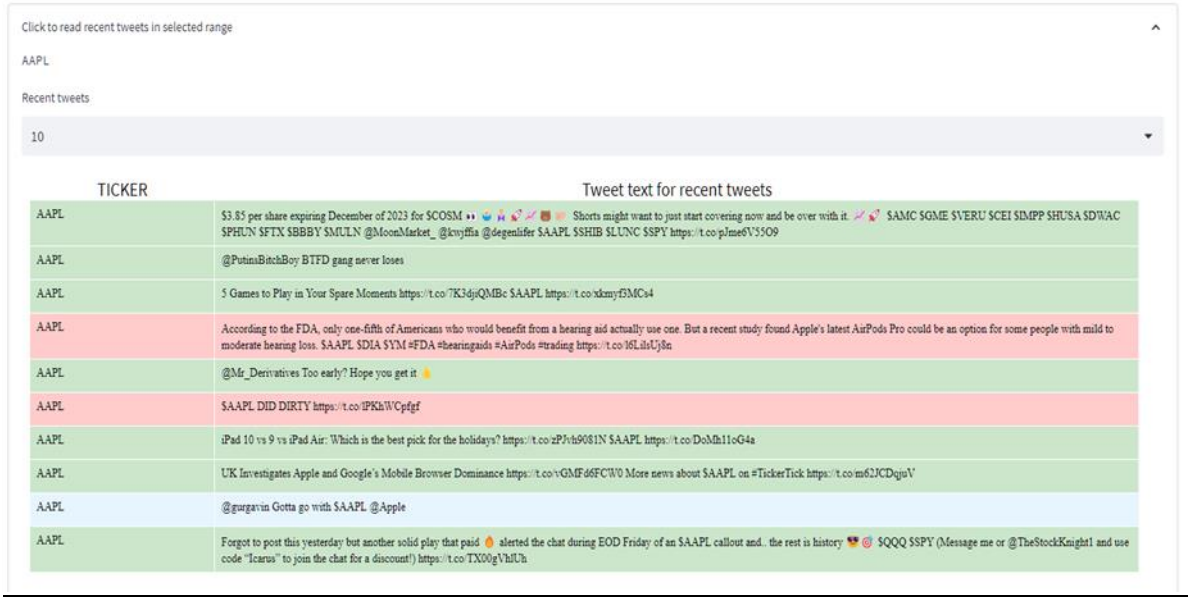


Figure 10: Recent Tweets for Selected TICKER

As a subelement, we are also displaying to the user the recent tweets for the selected TICKER, in the selected timestamp. The same is shown in the ***Figure 10***. In addition to tweets, we show the sentiment by using color highlights.

FRONT-END DEPLOYMENT

For the deployment of the front-end, we have used Docker. We dockerized the front-end code, and pushed the same to AWS ECR registry as shown below in figure.

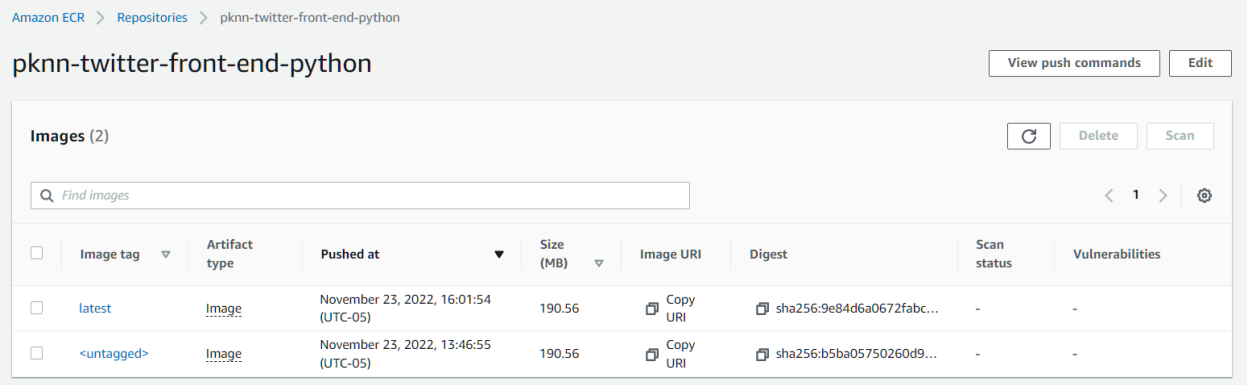


Figure 11: ECR Registry

Using the ECR image available, we create a task in ECS Cluster and run the task to access the front-end application at port 8080 and public IP address of the EC2 instance the docker container is hosted on.

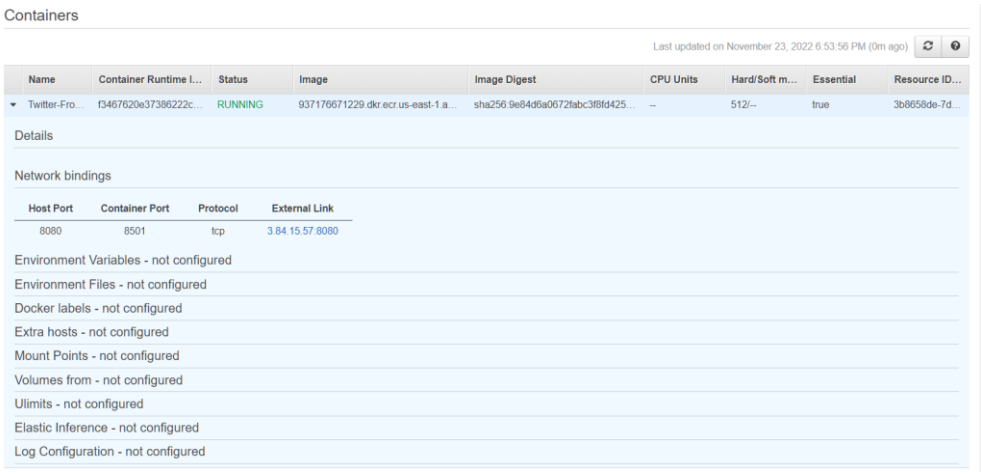


Figure 12: Task Container

Figure 12 shows the container hosting the front-end. As can be seen in the container image above, we have mapped the application container port 8501 to 8080 on EC2 machine.

The web-ui can be accessed at the address **http://3.84.15.57:8080/**

Note: The website is not using https protocol and doesn't have a domain name, so it is blocked on some networks(e.g. cestar database network).

The changes that are mentioned above are available on project's Github repository: https://github.com/k-pulkit/AwsStreamingDataPipeline

3. Difficulties Encountered in Reporting Week (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

First difficulty that we had is about DynamoDB queries. Every single page refresh creates a request and run a query that returns same results during a period of time. In order to decrease the query count, we used caching mecahism for server side, so new query will not run again unless it is in a new period of time. For example; 1 query will run for hourly results and rest of the requests retrieve data from cache. Other than that, we have noticed that Job3 Glue code produce inconsistent data format. It was hard to catch it because it was a logical error. After a couple of hours non-stop investigation, we were able to detect the problem and fixed it.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

We will be working on the final report and presentation in next week since we are done with the development and deployment phases. Along with the report and presentation, we will fix some UI issues and we are planning to finalize About section and Readme.md file.

Capstone Project Weekly Progress Report

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)
Reporting Week	28/11/2022 – 04/12/2022
Faculty Supervisor	William Pourmajidi

1. **Tasks Outlined in Previous Weekly Progress Report** (Provide detailed information on the tasks to be completed in this week)

The previous week, we were working on the last part of our project: the front end. Even though it was the final part, It was one of the important pieces of our project due to the demonstration. We tried to demonstrate every piece of information and insights we gathered using tweets. Apart from that, there were a few non-emergent duties that we completed such as the ReadMe.md file of the GitHub repository and the About section on our website. Overall, the previous week was the last week for us to develop or add new features.

2. **Progress Made in Reporting Week** (Provide detailed information on the progress that you made in the reporting week. Limit your write-up to no more than two pages)

The week we passed was the week to work on the final report and PowerPoint. To do that, we scanned our previous weekly reports, reviewed our codes, checked the cloud services we used and re-read the sources we used. Pulkit and Zarna created our final presentation and boilerplate of the final report. We wanted to mention all our features detailed. Unfortunately, the presentation was too long to pitch so we had to exclude some pages. After these revisions, PowerPoint was almost done except for our voiceovers. The final report was created by using the information in the previous reports and the final PowerPoint. Then we shared topics and each of us created a voiceover for the presentation.

Deep and Niki were checked and determined the issues overall. We discovered some minor issues on the API side such as unsuccessful requests while filtering tweets and some changes to the content of the About sections. Then we immediately fixed the issues.

Yasin worked on creating the last weekly report, the final report using the draft created before and finalized the PowerPoint. Yasin spent most of his time fixing typo issues and styling the content. He also merged the voiceovers and submit all documents that we have.

3. **Difficulties Encountered in Reporting Week** (Provide detailed information on the difficulties and issues that you encountered in the reporting week. Limit your write-up to no more than one page)

As we are about to finish our 3rd semester, time management was the most challenging problem we had. We had to meet and discuss our documents. Somehow we achieved that and shared tasks among us. Parallel working let us save time and we were able to fix the issues undiscovered before. Beside, voiceover is a new concept for us. We worked on that and created many records. As we created voiceovers, we were getting experience and we were able to record a record optimal length. Meeting the proper length and satisfying content of the voiceovers was challenging for us since it was content for us.

4. **Tasks to Be Completed in Next Week** (Outline the tasks to be completed in the following week)

We believe that we completed the project successfully and met the goals that we set in the beginning. Even though we reached our goals, this project is open to adding new features. This project taught us new disciplines, new technology stacks and being a team player. Even each of us successful individuals