# DBA FINAL PROJECT REPORT

# Real-Time Stock Market Analysis using AWS


## Team Members

Pruthav Jhaveri

Saheb Gabadia

# 1. Overview

**Setup 1:** EC2-RDS (PostgreSQL) for Stock Data Analysis

Components:

- CSV File: Contains stock data with 9 columns (Index, Date, High, Open, Low, Close, Adj_Close, Volume, Close_USD).
- Python Script: Simulates real-time stock data analysis.
- AWS EC2 (Elastic Compute Cloud): Hosts the Python script and manages the connection to RDS.
- AWS RDS (Relational Database Service): PostgreSQL database to store and manage stock data.

Workflow:

- Python Script Execution: The Python script runs on an EC2 instance. This script is responsible for loading the stock data into RDS, simulating a real-time environment.
- Querying Data: Uses SQL queries via PostgreSQL for data manipulation and retrieval.
- Data Processing: The PostgreSQL database on RDS is used to perform real-time analysis of stock data (such as calculating moving averages, identifying trends, etc.).
- Data Output: The output can be in the form of reports or visualizations.

**Setup 2:** EC2-S3-Athena for Stock Data Analysis

Components

- CSV File: Same stock data file as in Setup 1.
- Python Script: Simulates real-time stock data analysis.
- AWS EC2: Hosts the Python script.
- AWS S3: Stores stock data.

- AWS Athena: Serverless query service to perform SQL queries on data stored in S3.

Workflow

- Python Script Execution: The Python script runs on an EC2 instance. This script is responsible for loading the stock data into S3, simulating a real-time environment.
- Querying Data: Uses SQL queries via AWS Athena for data manipulation and retrieval.
- Data Processing: Athena queries the stock data stored in S3 to perform analysis based on the query results (e.g., trend analysis, volatility computation).
- Data Output: The output can in the form of reports or visualizations.

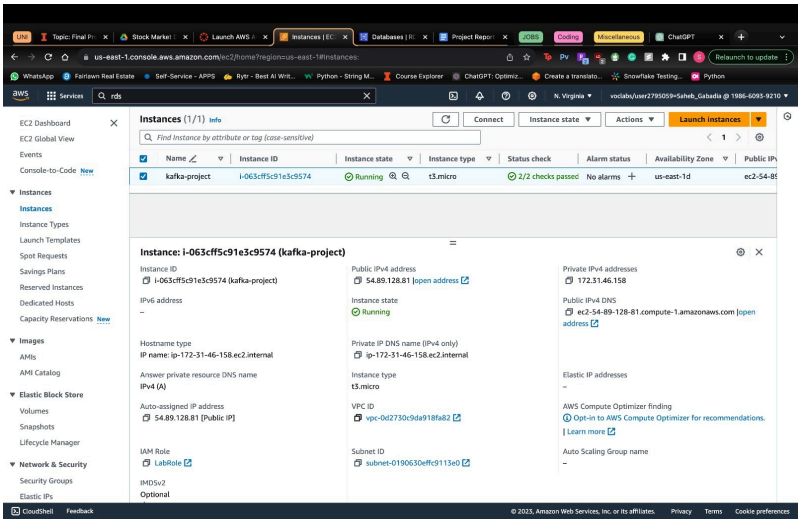
## 2. EC2-RDS (PostgreSQL) for Stock Data Analysis

### Python Script Execution

```python
import pandas as pd
from sqlalchemy import create_engine

# Database connection details
username = 'postgres'
password = '12345678'
host = 'kafka-project.cudcojouqykl.us-east-1.rds.amazonaws.com'
database = 'Stock Market Databse'
port = 5432  # default PostgreSQL port

# Create a connection to the database
engine = create_engine(f'postgresql+psycopg2://{username}:{password}@{host}:{port}/{database}')

# Read the CSV file
df = pd.read_csv("/home/ec2-user/Stock Market Dataset.csv")

# Sample and write data to RDS
sampled_df = pd.DataFrame()
ctr = 0
while True:
    temp = df.sample(1)
    sampled_df = sampled_df.append(temp)

    sampled_df.to_sql('Stock data1', con=engine, if_exists='append', index=False)

    # Reset the sampled_df to avoid re-writing the same data
    sampled_df = pd.DataFrame()

    # Add a break condition or this will run indefinitely
    ctr += 1
```

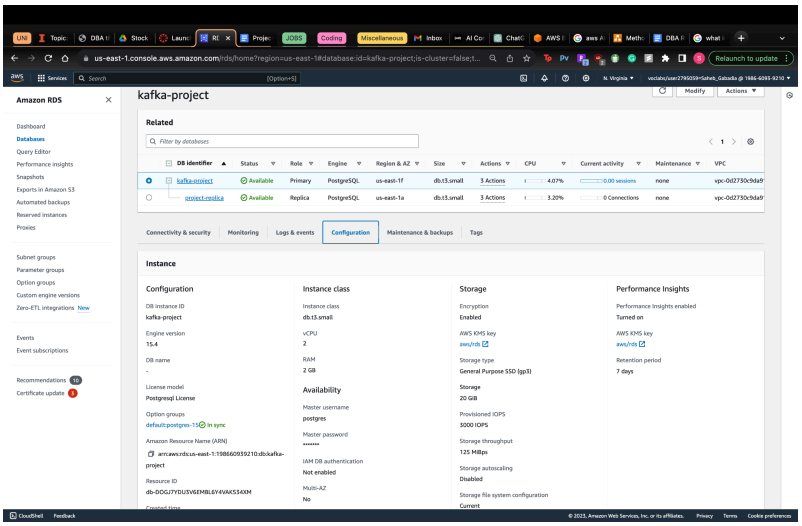
The python script is running on the EC2 instance, where it retrieves one row in real time and loads the data into a RDS database.

## EC2 Instance



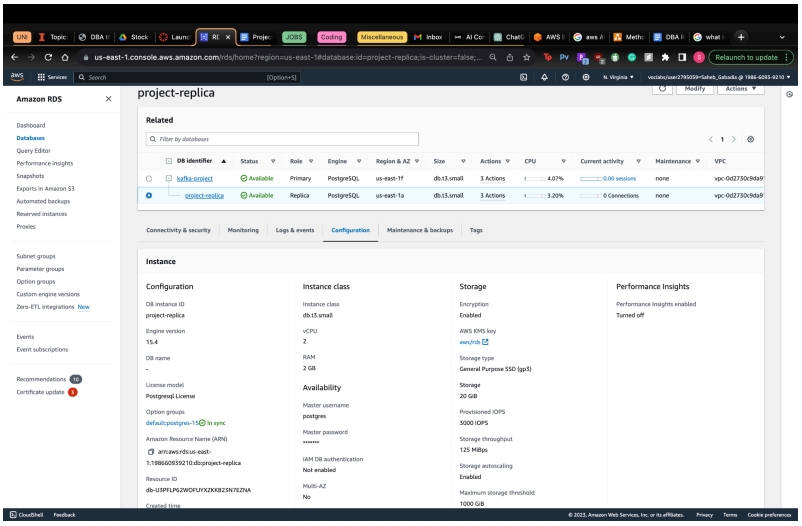

We have created an EC2 instance that runs the python script. The instance type is t3.micro.

## RDS



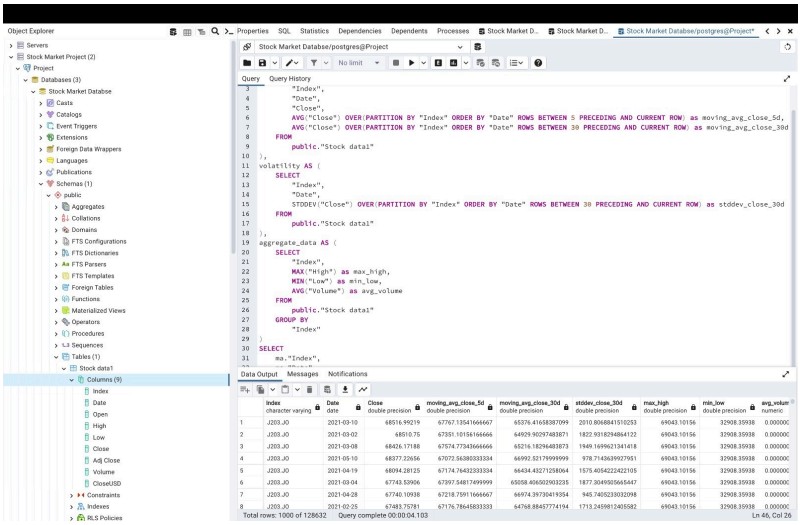

The RDS database stores the data that is loaded by the EC2 instance. It has a PostgreSQL engine so that we can connect to pgAdmin and execute queries to analyse trends in the stock market and understand stock performance.

## RDS Replica Set



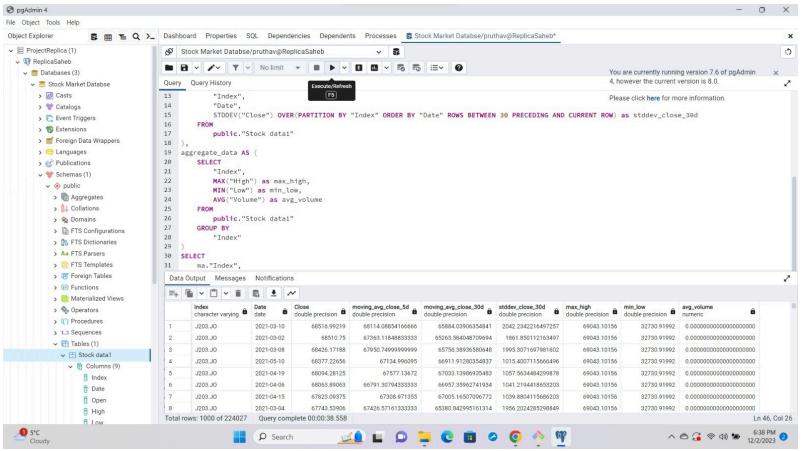

We have also created a read only replica set in the RDS database. The purpose of creating this is to reduce the load on the main database. By using this, we can give read only access to users to analyze stock data without compromising the integrity of the main database.

## Querying on main database



We have executed a complex query on the main database to see the time taken for the query to return the output and also monitor performance.

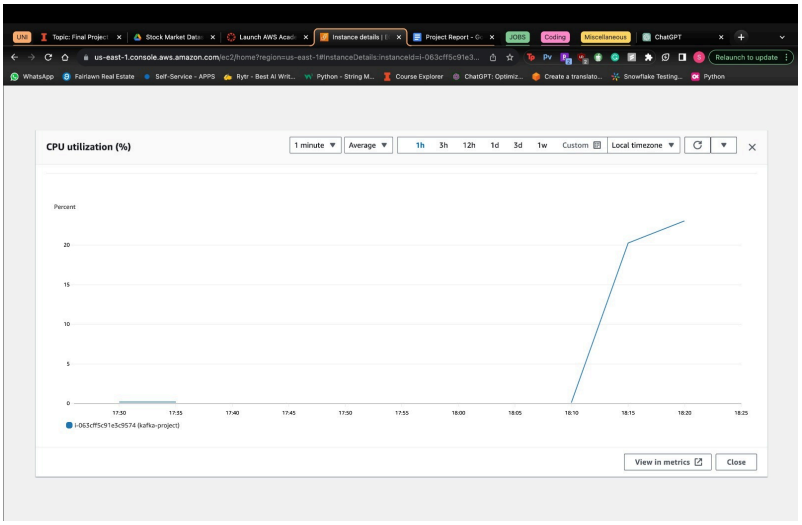
## Querying on replica set

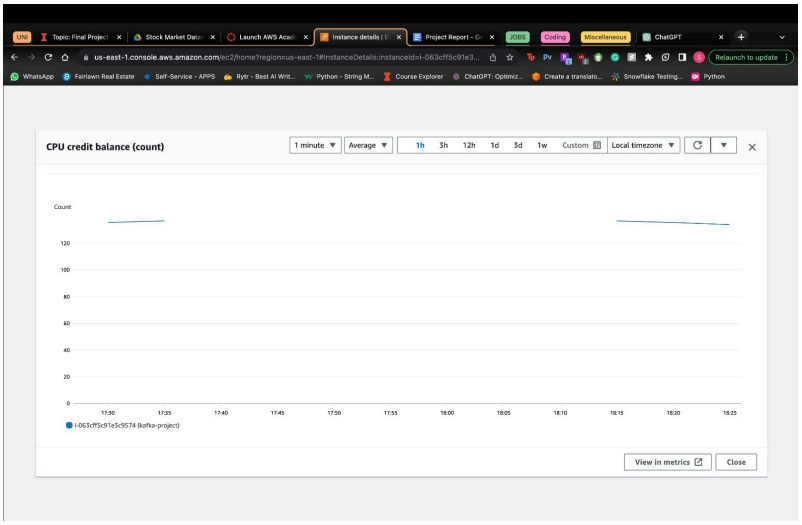


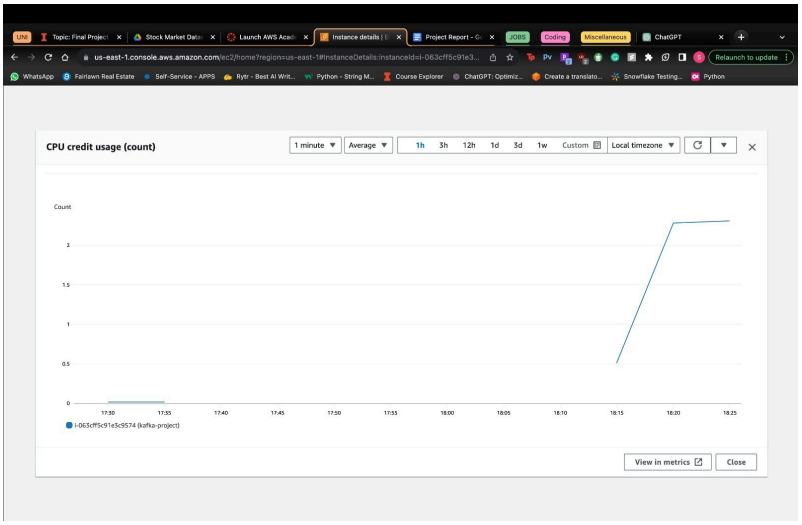
The same query is executed on the replica set as well so that we can understand the time difference, monitor performance and evaluate the efficiency of the system.

# Monitoring

## EC2 Monitoring





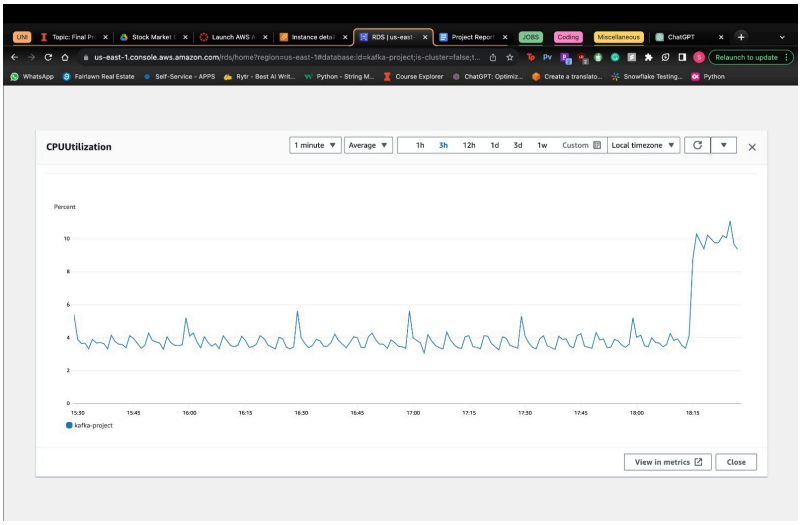



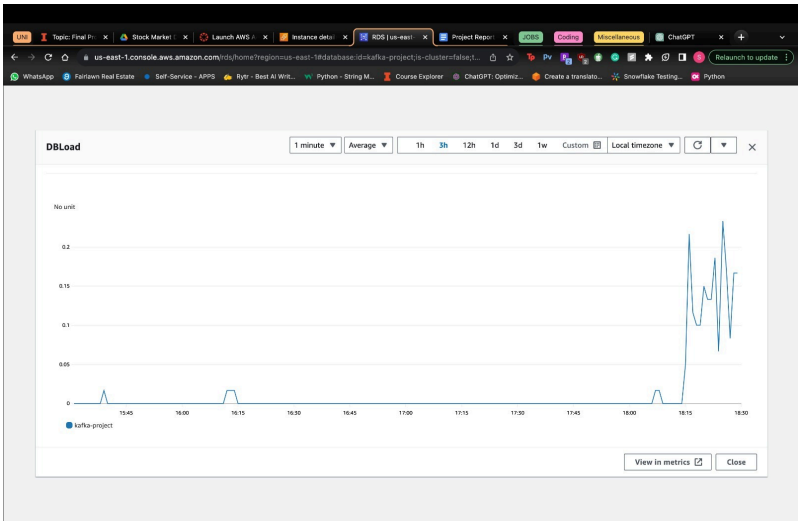

The above pictures monitor the performance of the EC2 instance that loads data in the RDS database.

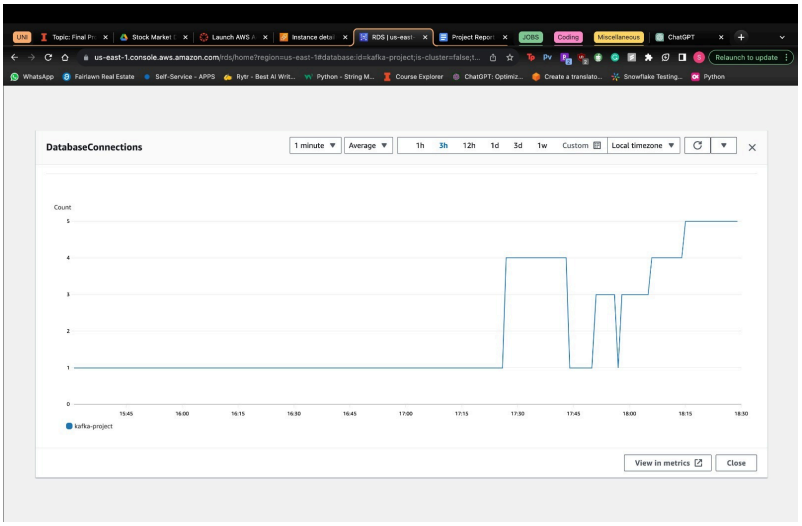
We started the execution of the python script at 18:15 and as we can see through the CPU Utilization graph, that there is a 20% increase in the utilization, which increases even further as more data is loaded.

The CPU credit balance starts to decrease from 18:15 and the CPU credit usage increases from 18:15, indicating the amount of credit being used.

## RDS Monitoring:



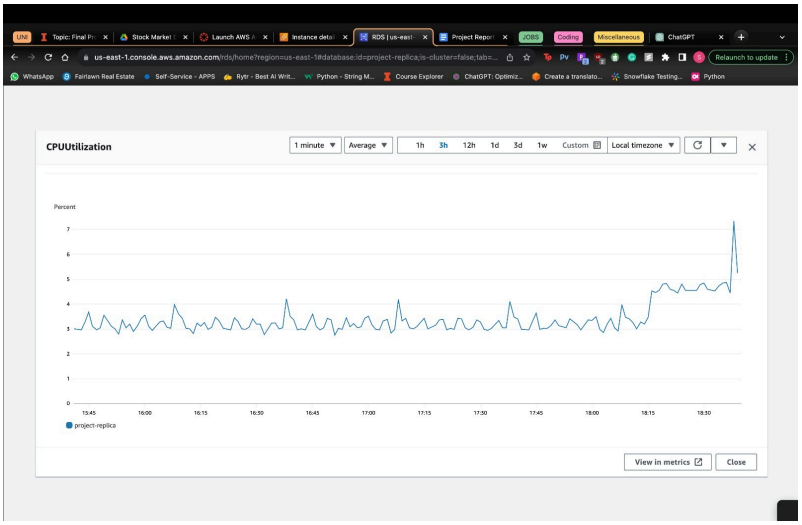



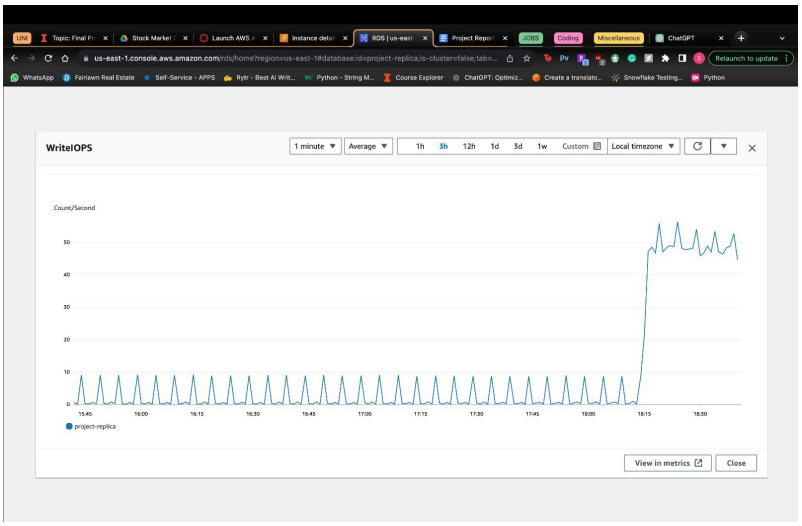

The above pictures monitor the performance of the primary RDS database when a query is executed.

We executed the query at 18:16, after starting the data loading into RDS. The CPU Utilization had an instant spike, increasing by about 6%.

The DBLoad and Database Connections also increased from 18:16.

## RDS Replica Set Monitoring



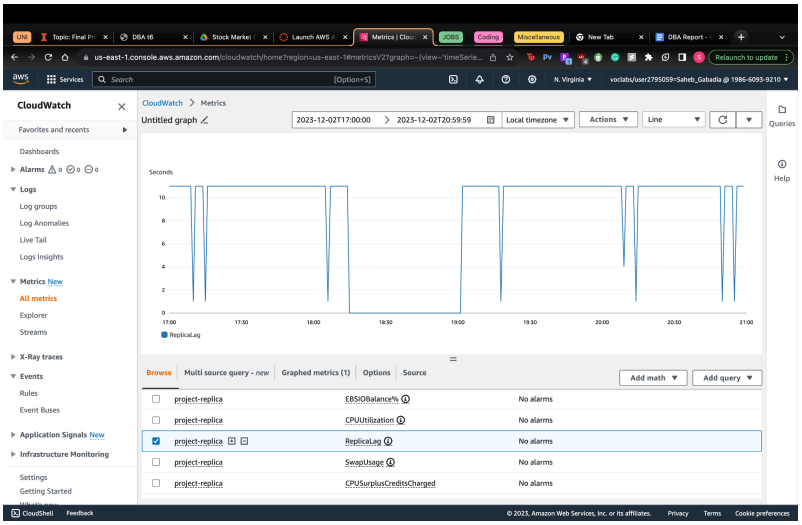




The above pictures monitor the performance of the replica RDS database when a query is executed. We fired the exact same query on both databases.

We executed the query at 18:16, after starting the data loading into RDS which caused the data to be replicated in the replica set.

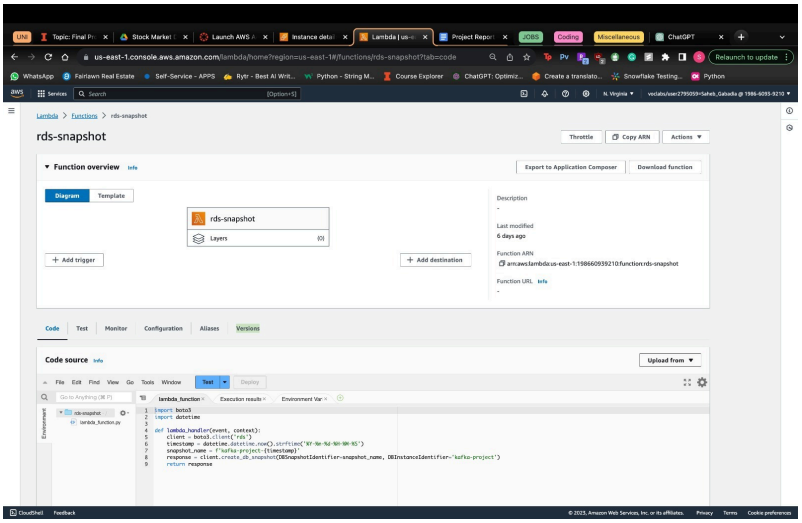
The CPU Utilization had an instant spike, increasing by about 1.5%.

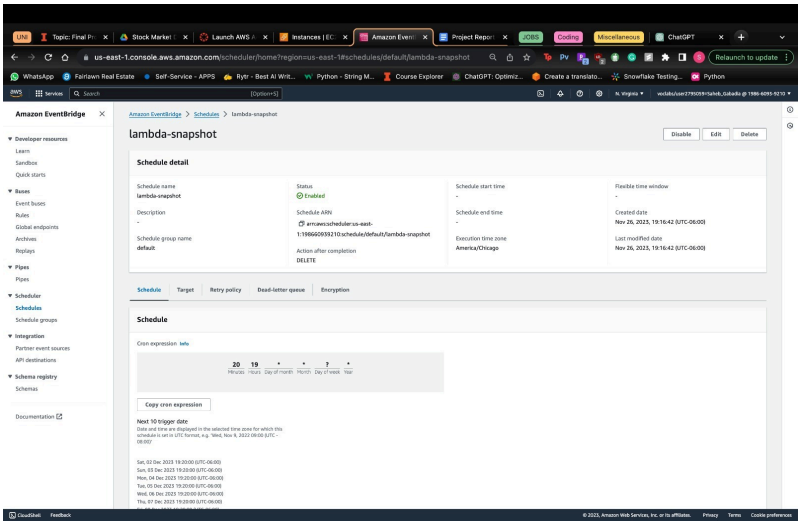
## Replica Set Lag:

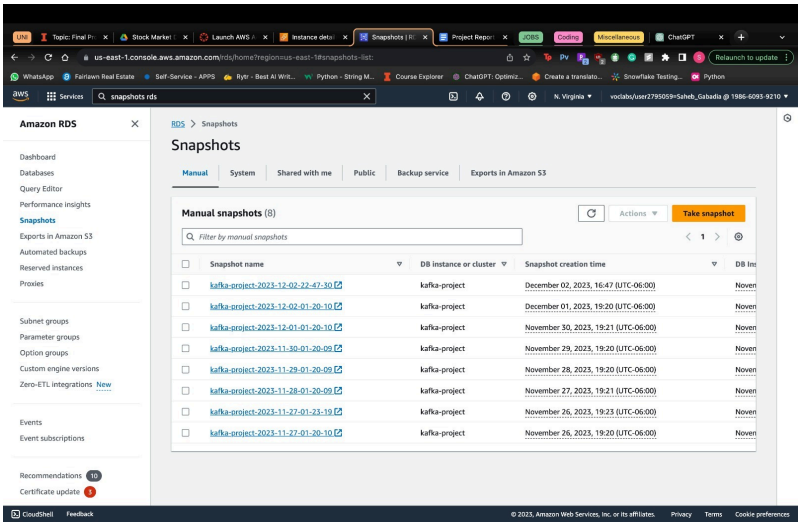



The replication lag was also monitored and it was approximately 9 seconds when we fired the query.

# Backup

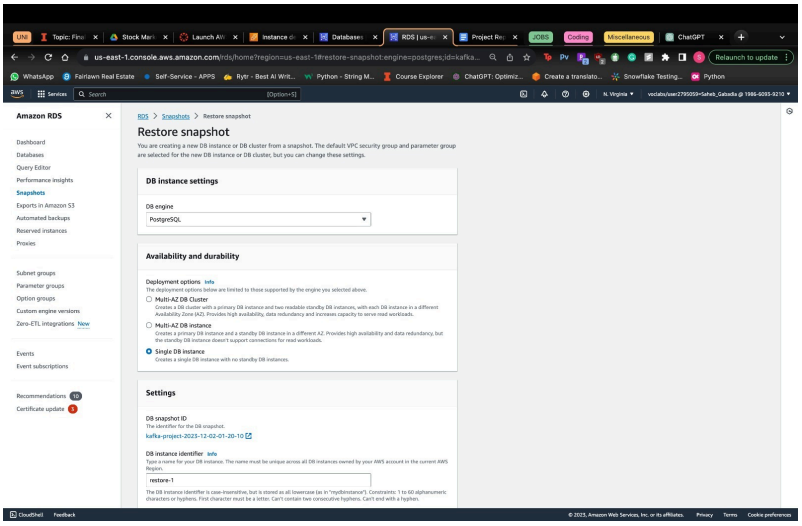


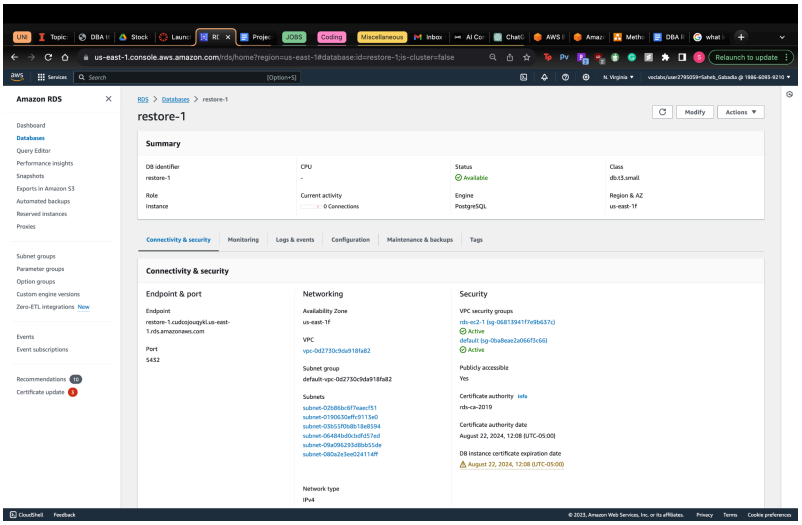



Data backups in RDS work through storing a snapshot of the entire database. We have created snapshot named ‘rds-snapshot’ which uses a lambda function named ‘lambda-snapshot’ that has a CRON job scheduler to take backups once every day at 19:21. We have also shown the snapshots captured so far, from when we started the scheduler.

# Restoring Backup



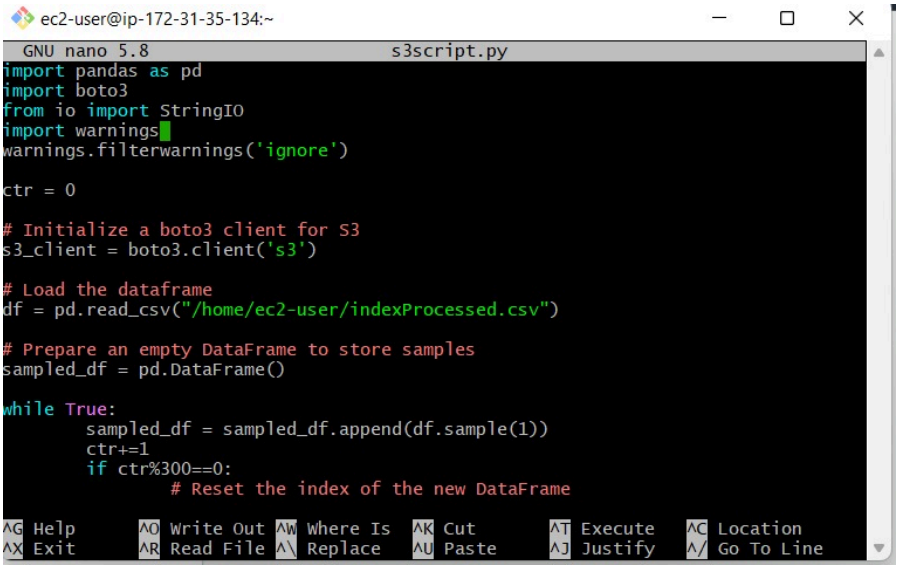

For restoring a backup in RDS, the snapshot has to be restored as a RDS instance. This is a new RDS instance that is created, so it uses the same space as the main database and levies the same cost again. The above pictures show the restoration of a snapshot and the restored snapshot.

## Security and managing access

- To manage security and access we have created various security rules in the EC2 instance.
- One group contains roles granting only the administrator access to the EC2 instance, primary RDS instance with privileges to create replicas, take snapshots for backup and restore incase of failure.
- Another group has access only to the read replica of the RDS instance. Thus ensuring they cannot connect to the main RDS instance using pgAdmin and change data or create replicas and backups.
- In order to add any new user who would like to use pgAdmin for querying and analysis on the data on the replica we would have to add their IP to the list of security rules, only after which they would be allowed to use the RDS endpoint to query the data.
- In order to allow our EC2 instance to connect to the primary RDS instance and write data into it we have used the IAM LabRole.

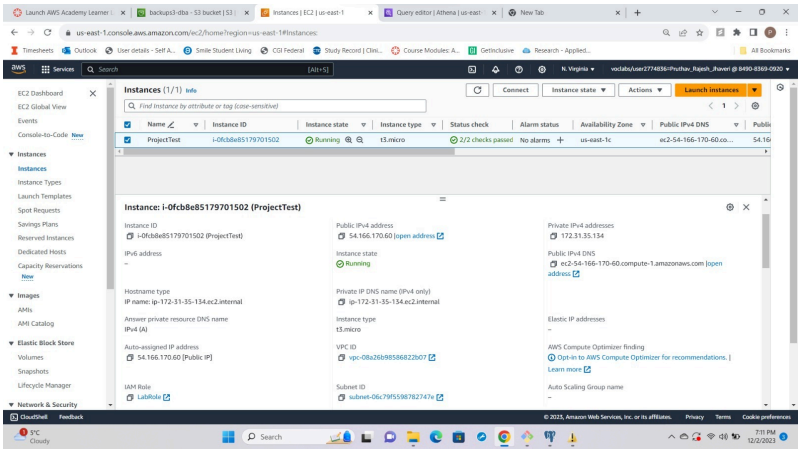
# 3. EC2-S3-Athena for Stock Data Analysis

## Python Script Execution



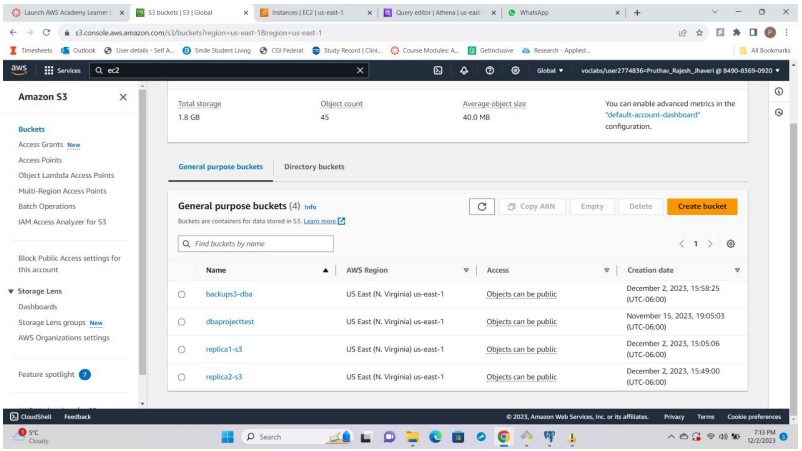

The python script is running on the EC2 instance, where it retrieves one row in real time and loads the data into a S3 bucket.

## EC2 Instance



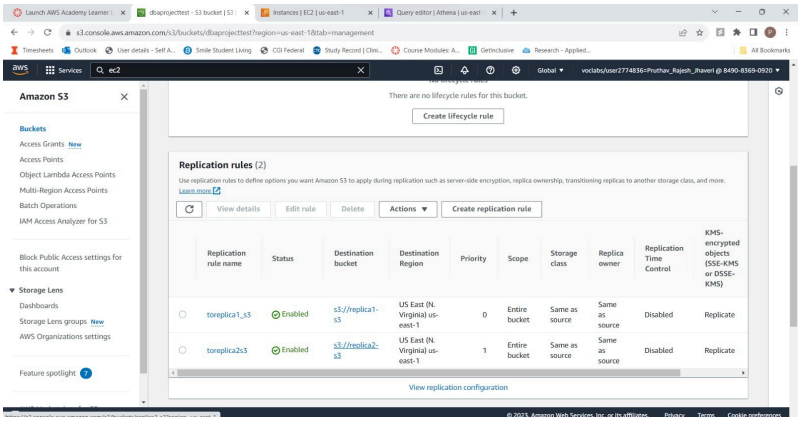

We have created an EC2 instance that runs the python script. The instance type is t3.micro.

## S3 Buckets



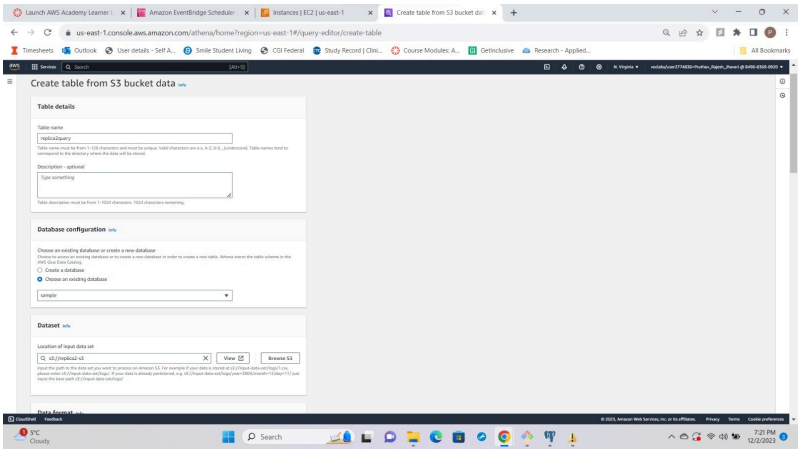

The S3 bucket stores the data that is loaded by the EC2 instance. The bucket can connect to Athena to execute queries to analyse trends in the stock market and understand stock performance.

## S3 replication sets



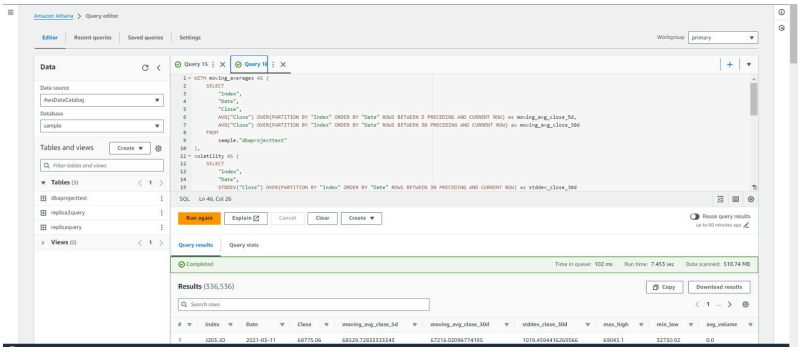

We have also created a replica set in the S3 buckets. By using this, we can give read only access to users to analyze stock data without compromising the integrity of the main bucket.

## Setting up table for querying the main S3 bucket



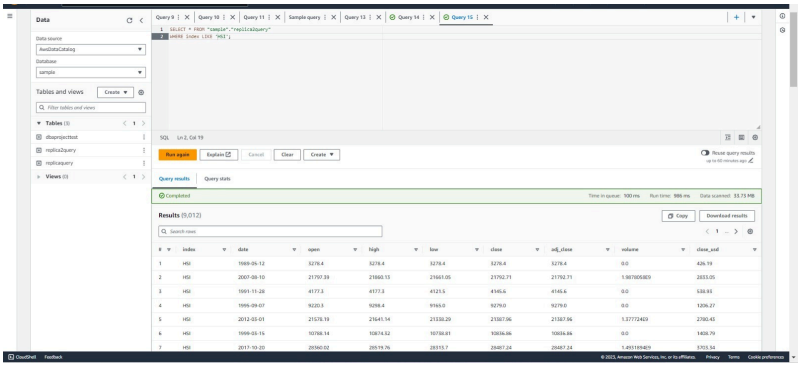

Here, we are setting up an empty table in Athena, to store the data results from querying. Athena can directly connect to a S3 bucket.

## Query on main S3 bucket




Here we execute the same query that we executed in PostgreSQL using RDS. The number of records are higher than when we executed queries in PostgreSQL because the number of records fetched by S3 is higher. So the number of records scanned depend on the number of records that are fetched by the S3 bucket.

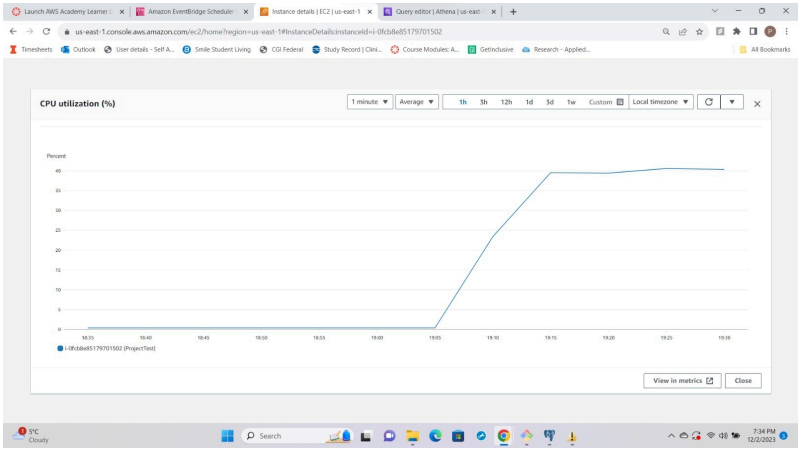
## Query on replica bucket

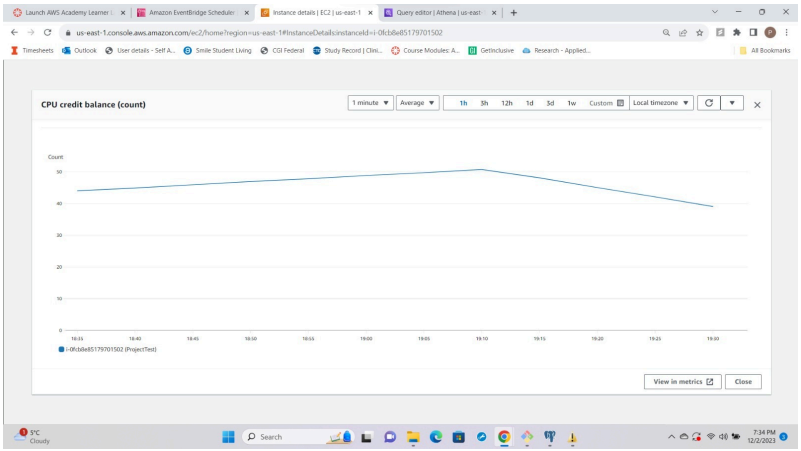


We have also executed a simple query on the replica bucket for monitoring.

# Monitoring

## EC2 Monitoring



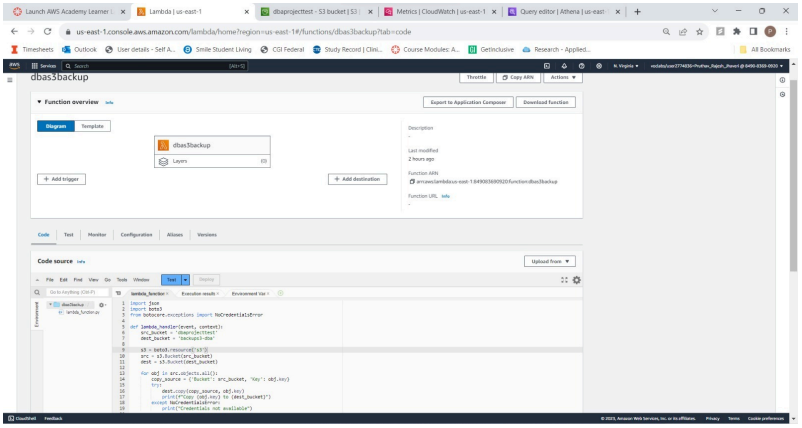

The above pictures monitor the performance of the EC2 instance that loads data in the S3 bucket.

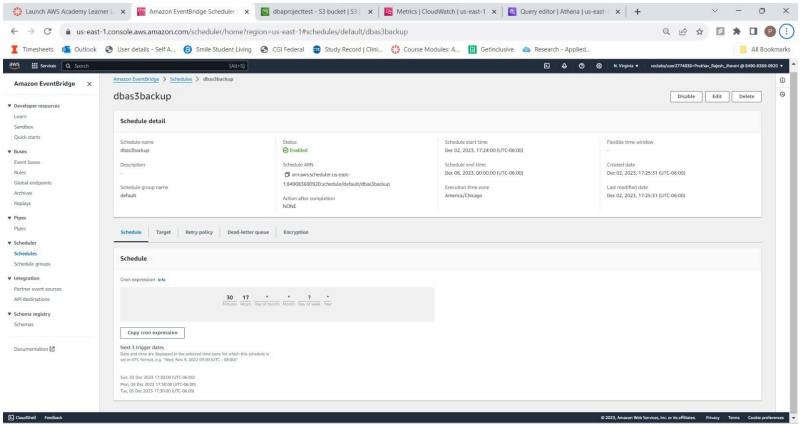
We started the execution of the python script at 19:10 and as we can see through the

CPU Utilization graph, that there is a 15% increase in the utilization, which increases even further as more data is loaded.

The CPU credit balance starts to decrease from 19:10, indicating the amount of credit being used.

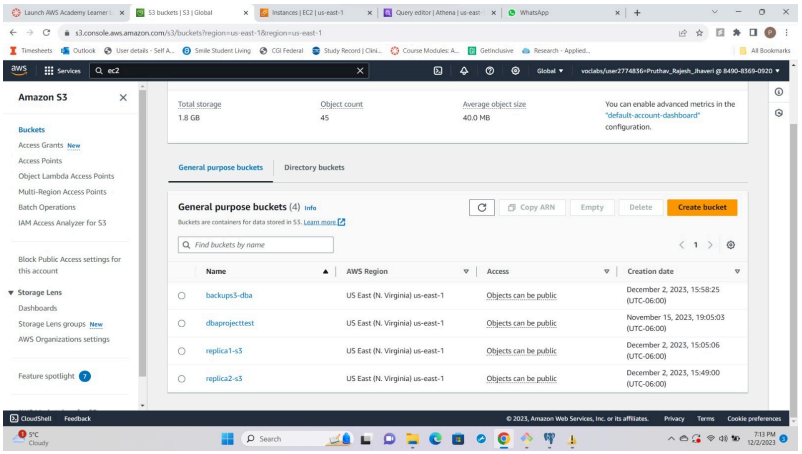
## Backup





Data backups in S3 work through storing a copy of the entire bucket. We have created bucket named ‘dbas3backup’ which uses a lambda function named ‘dbas3backup’ that has a CRON job scheduler to take backups once every day at 17:30.

# Restoring Backup




For restoring the backup, we create an empty bucket which is where the data stored after backing up by the CRON job scheduler.

# Security and managing access

- To manage security and access we have created various security rules for the EC2 instance.
- In order to allow our EC2 instance to connect to the primary S3 instance and write data into it we have used the IAM LabRole.
- Using the LabRole we are also able to create replicas and a backup bucket.
- In order to allow other users to connect replicas, one method is to create roles for them and grant them read only access to the buckets. After which they can go to Athena and load the data into a table from the bucket.
- Another method is to enable cross origin resource sharing by editing the file in JSON format to permit other users to access the S3 replicas.
- However, these methods could not be tested by us given the limitations of AWS academy.

## 4. Comparative Analysis

EC2-RDS (PostgreSQL):

- Data Consistency: Exhibits strong data consistency, ensuring reliability in transaction handling and data accuracy.
- ACID Compliance: Robust ACID compliance maintains data integrity during transactions, essential for precise financial calculations.
- Query Performance: Utilizes optimized indexes, facilitating fast and efficient query responses ideal for real-time or near-real-time data analysis.
- Storage Cost: Involves higher RDS storage costs, which may impact the overall budget of the data analysis project.
- Data Management: Simplifies the creation and maintenance of replicas and backups, reducing administrative overhead and enhancing data security.
- Data Update Frequency: Offers near-real-time data updates from EC2 to RDS, crucial for time-sensitive stock market analyses.
- Data Suitability: Best suited for well-structured data, enabling organized and efficient processing of complex queries.
- Operational Complexity: Generally easier to manage in terms of database administration compared to file-based systems.

EC2-S3-Athena:

- Data Consistency: Has weaker data consistency, which might be a concern for scenarios where transactional integrity is paramount.
- ACID Compliance: Weaker ACID compliance, potentially leading to challenges in maintaining data integrity in complex transaction scenarios.
- Query Performance: Faces higher latency, especially for real-time or frequent queries, which can be a limiting factor for dynamic stock analysis.
- Storage Cost: Offers lower S3 storage costs, making it a cost-effective solution for large-scale data storage needs.

- Data Management: Management of replicas and backups is more complex, requiring additional planning and resources.
- Data Update Frequency: Writes data in blocks to S3 from EC2, which might not support the immediacy required for certain financial analyses.
- Data Suitability: More adaptable for handling raw or unstructured data formats, offering flexibility in dealing with diverse data sources.
- Operational Complexity: Necessitates a more sophisticated setup for managing data access and retrieval, especially when dealing with large datasets.

## 5. Cost Analysis

All the costs provided below are reported in USD. Assumption made involves the services being used for 24 hours a day, and 30 days for a month.

**EC2-RDS (PostgreSQL):**

| Services | cost/hour | cost/day | cost/month | cost/year |
|---|---|---|---|---|
| **EC2** | 0.0104 | 0.2496 | 7.488 | 89.856 |
| **RDS (Primary)** | 0.036 | 0.864 | 25.92 | 311.04 |
| **RDS (Replica)** | 0.036 | 0.864 | 25.92 | 311.04 |

| | cost/gb/month | cost/day | cost/month | cost/year |
|---|---|---|---|---|
| **RDS Snapshot** | 0.095 | 1.9 | 22.8 | 159.6 |

| | |
|---|---|
| **Total** | **871.536** |

**EC2-S3-Athena:**

| Services | cost/hour | cost/day | cost/month | cost/year |
|---|---|---|---|---|
| **EC2** | 0.0104 | 0.2496 | 7.488 | 89.856 |

For S3 up till first 50TB costs 0.023/gb/month. Hence considering each S3 bucket uses about 20gb the following cost has been calculated

| | cost/month | cost/year |
|---|---|---|
| **S3** | 0.46 | 5.52 |
| **S3 replica** | 0.46 | 5.52 |
| **S3 backup** | 0.46 | 5.52 |

The cost of Athena varies upon the query and data scanned. It costs $5/TB, if we assume data to be 10GB, and 20 queries a day for 30th days a month the cost is $30.

| **Total** | **136.416** |
|---|---|

# 6. Conclusion

Summarising our findings we were able to carry out a comparative analysis for two methods to analyse stock market data in a simulated real-time scenario. We explored the creation and working of replicas, backups, and restore while ensuring adequate access control was in place. By executing queries of equal complexity in both methods followed by meticulous monitoring of performance and cost we got an in-depth understanding of the pros and cons of the aforementioned methods.

In accordance with the above comparative analysis we can conclude that both methods have their equal advantages and disadvantages. Using a hybrid approach would be best to maximise efficiency and minimise costs. For near real-time querying, ease of replication, lower latency, and accessibility to individuals with lesser understanding about AWS one should use the RDS instance with pgAdmin, whereas for storage, historical analysis and ease of recovery one should employ S3 with Athena.

## 7. References


- AWS S3 documentation https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
- AWS Athena documentation https://docs.aws.amazon.com/athena/latest/ug/what-is.html
- AWS Lambda documentation https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- AWS RDS documentation https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html
- Past assignments and instructions for navigating through RDS and pgAdmin
- AWS EventBridge https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-create-rule-schedule.html
- Boto3 https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
- Sqlalchemy https://docs.sqlalchemy.org/en/20/