

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

```

import pandas as pd
from sqlalchemy import create_engine

# Database connection details
username = 'postgres'
password = '12345678'
host = 'kafka-project.cudcojounykl.us-east-1.rds.amazonaws.com'
database = 'Stock Market Database'
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

The screenshot shows the AWS EC2 Instances page. A single instance, 'kafka-project' (ID: i-063cff5c91e3c9574), is listed as 'Running'. It is an 't3.micro' instance type, located in 'us-east-1' (N. Virginia). The instance has a public IPv4 address (54.89.128.81) and a private IPv4 address (172.31.46.158). It is associated with a VPC (vpc-0d2730c9da9) and a subnet (subnet-0190630effc9113e0). The instance is part of a security group named 'Labhole'. The status check shows 2/2 checks passed with no alarms.

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

RDS

The screenshot shows the AWS RDS Databases page. There are two database instances: 'kafka-project' (PostgreSQL, db.t3.small, us-east-1f) and 'aws-test-rds' (PostgreSQL, db.t3.small, us-east-1a). Both are currently 'Available'. The 'Configuration' tab is selected for the 'kafka-project' instance, showing details like engine version (15.4), storage (2 GB), and performance insights settings (AWS KMS key, retention period 7 days).

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

The screenshot shows the AWS RDS console with the 'Amazon RDS' sidebar open. Under 'Databases', there is a table with two rows: 'taha-project' (Primary) and 'project-replica' (Replica). The 'project-replica' row has a status of 'Available', engine 'PostgreSQL', region 'us-east-1a', instance class 'db.t3.small', and a 3.20% utilization rate. The 'Actions' column for this row contains a 'Modify' link.

The main area shows the 'Configuration' tab for the 'project-replica' instance. It lists various configuration parameters such as DB instance ID, Engine version (15.4), DB name (''), License model (PostgreSQL License), and Resource ID (db-1G3PLN2WOFUTXK8B2NTZNA). The 'Performance Insights' section indicates it is turned off.

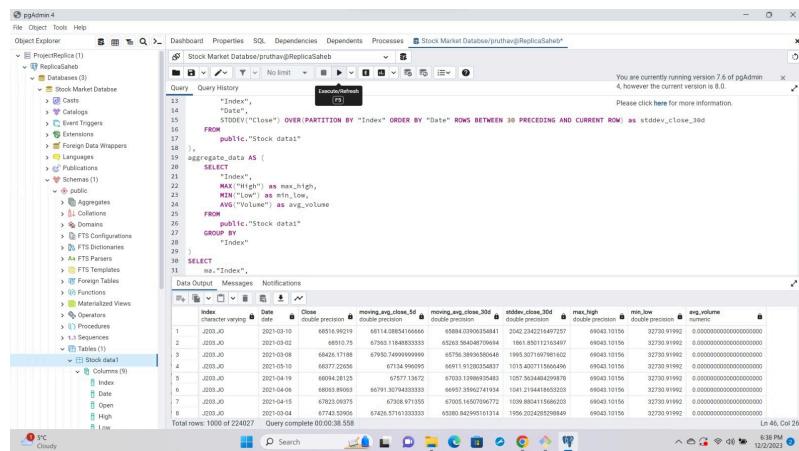
We have also created a read only replica set in the RDS database. The purpose of creating this is to reduce the load on the main database. By using this, we can give read only access to users to analyze stock data without compromising the integrity of the main database.

Querying on main database

The screenshot shows the PostgreSQL Query Editor with a complex query being run against the 'Stock Market Database'. The query calculates moving averages and standard deviation over a 30-day period for stock data. The results are displayed in a table with columns for Date, Open, High, Low, Close, Adj Close, Volume, and Dividends.

Date	Open	High	Low	Close	Adj Close	Volume	Dividends	moving_avg_close_5d	moving_avg_close_30d	stddev_close_5d	stddev_close_30d	max_low	min_low	avg_value
2023-03-10	48516.9219	48516.9219	48516.9219	48516.9219	48516.9219	1000	0	48516.9219	48516.9219	0.00000	0.00000	48516.9219	48516.9219	48516.9219
2023-03-02	48516.75	48516.75	48516.75	48516.75	48516.75	1000	0	48516.75	48516.75	0.00000	0.00000	48516.75	48516.75	48516.75
2023-03-09	48426.1718	48426.1718	48426.1718	48426.1718	48426.1718	1000	0	48426.1718	48426.1718	0.00000	0.00000	48426.1718	48426.1718	48426.1718
2023-03-16	48077.2594	48077.2594	48077.2594	48077.2594	48077.2594	1000	0	48077.2594	48077.2594	0.00000	0.00000	48077.2594	48077.2594	48077.2594
2023-03-23	48474.1623	48474.1623	48474.1623	48474.1623	48474.1623	1000	0	48474.1623	48474.1623	0.00000	0.00000	48474.1623	48474.1623	48474.1623
2023-03-30	48743.0398	48743.0398	48743.0398	48743.0398	48743.0398	1000	0	48743.0398	48743.0398	0.00000	0.00000	48743.0398	48743.0398	48743.0398
2023-04-06	48740.1019	48740.1019	48740.1019	48740.1019	48740.1019	1000	0	48740.1019	48740.1019	0.00000	0.00000	48740.1019	48740.1019	48740.1019
2023-04-13	48974.3941	48974.3941	48974.3941	48974.3941	48974.3941	1000	0	48974.3941	48974.3941	0.00000	0.00000	48974.3941	48974.3941	48974.3941
2023-04-20	49494.1056	49494.1056	49494.1056	49494.1056	49494.1056	1000	0	49494.1056	49494.1056	0.00000	0.00000	49494.1056	49494.1056	49494.1056
2023-04-27	47483.7591	47483.7591	47483.7591	47483.7591	47483.7591	1000	0	47483.7591	47483.7591	0.00000	0.00000	47483.7591	47483.7591	47483.7591
2023-05-04	47176.764689	47176.764689	47176.764689	47176.764689	47176.764689	1000	0	47176.764689	47176.764689	0.00000	0.00000	47176.764689	47176.764689	47176.764689
2023-05-11	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-05-18	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-05-25	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-06-01	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-06-08	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-06-15	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-06-22	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-06-29	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-07-06	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-07-13	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-07-20	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-07-27	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-08-03	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-08-10	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-08-17	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-08-24	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-09-01	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-09-08	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-09-15	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-09-22	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-09-29	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-10-06	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-10-13	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-10-20	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-10-27	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-11-03	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-11-10	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-11-17	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-11-24	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-12-01	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-12-08	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-12-15	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-12-22	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2023-12-29	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2024-01-05	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2024-01-12	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2024-01-19	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2024-01-26	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2024-02-02	47176.7719	47176.7719	47176.7719	47176.7719	47176.7719	1000	0	47176.7719	47176.7719	0.00000	0.00000	47176.7719	47176.7719	47176.7719
2024-02-09	47176.7719	47176.7719	47176.7719	47176.7719	47176.77									

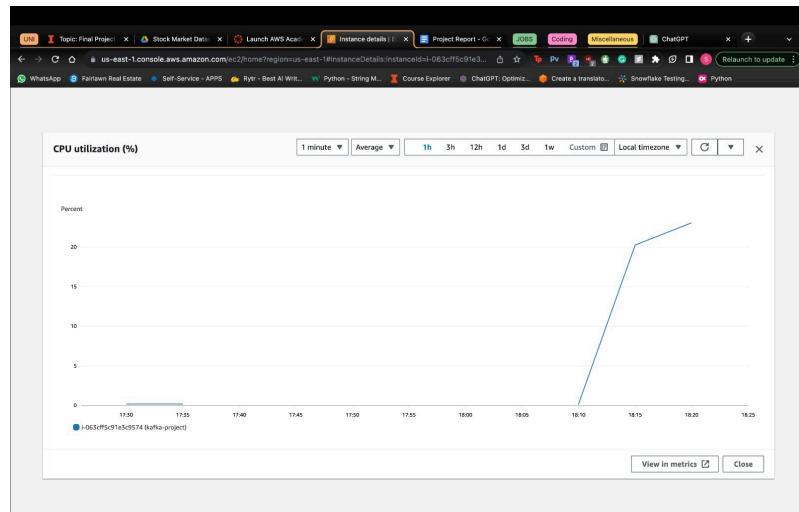
Querying on replica set

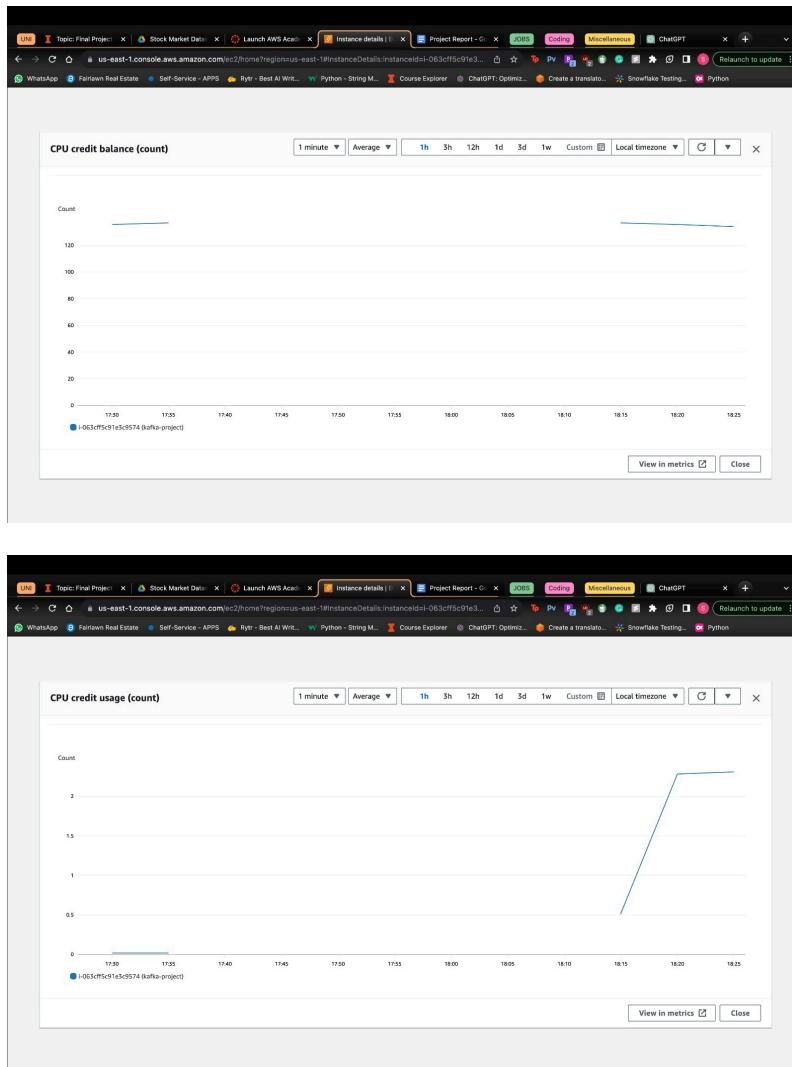


The same query is executed on the replica set as well so that we can understand the time difference, monitor performance and evaluate the efficiency of the system.

Monitoring

EC2 Monitoring



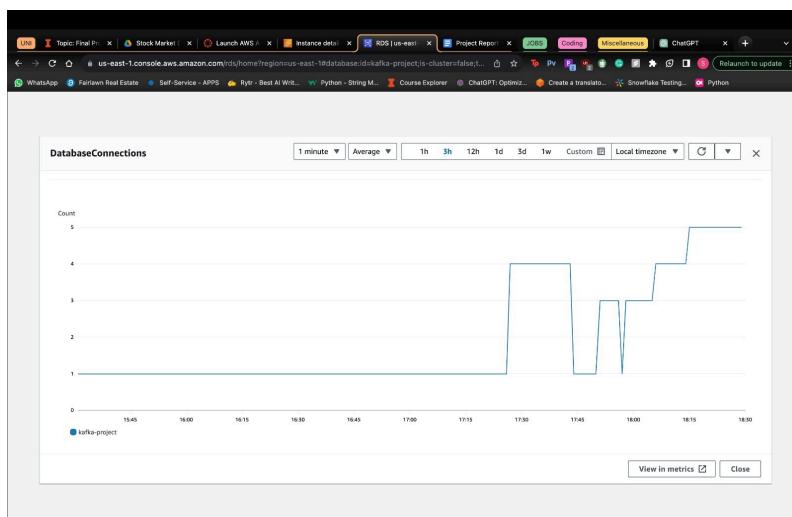
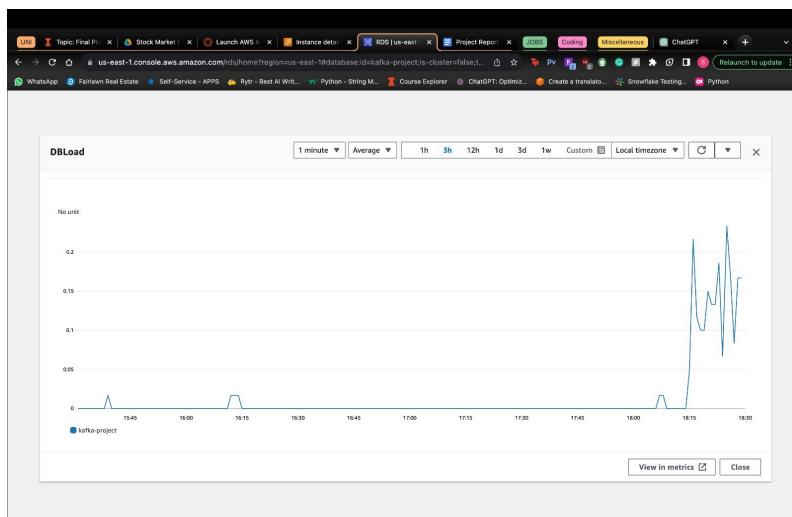
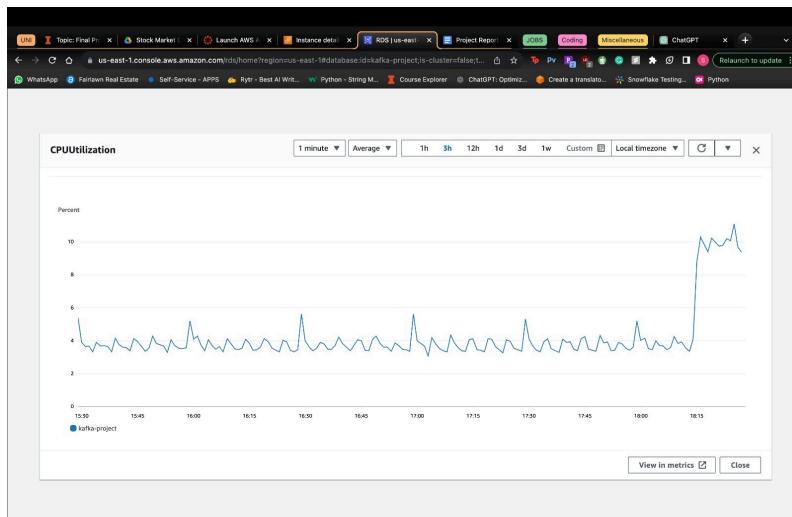


The above pictures monitor the performance of the EC2 instance that loads data in the RDS database.

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:

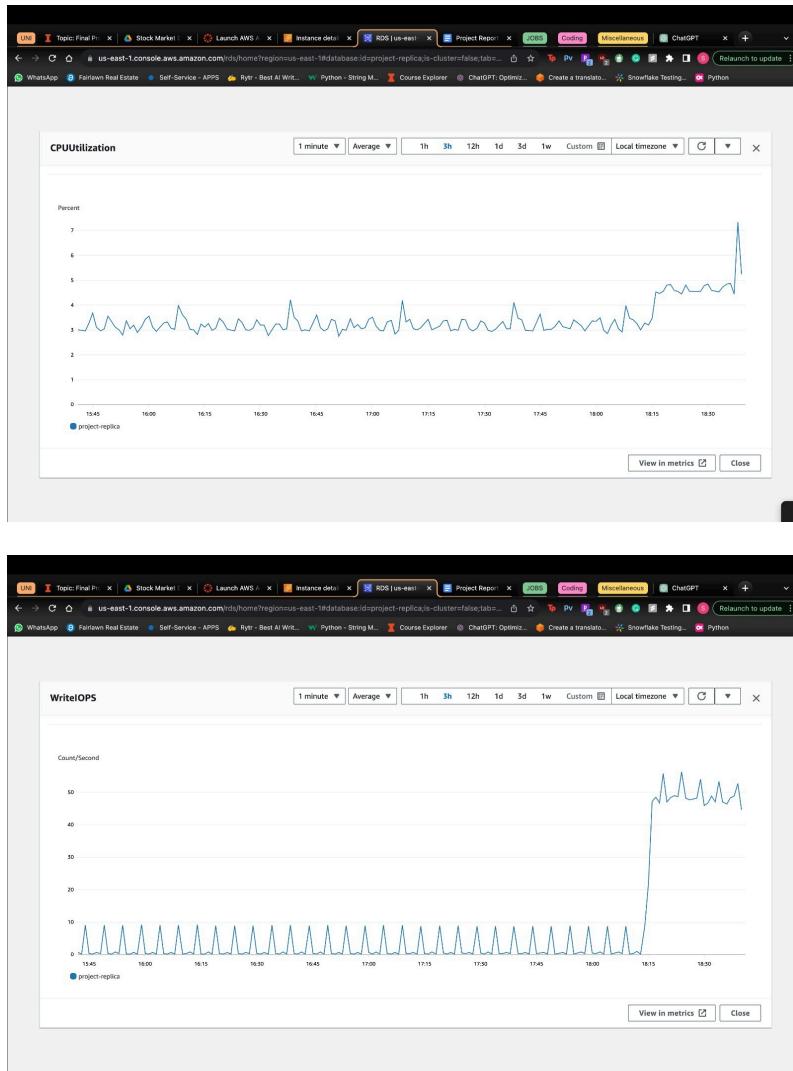


The above pictures monitor the performance of the primary RDS database when a query is executed.

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

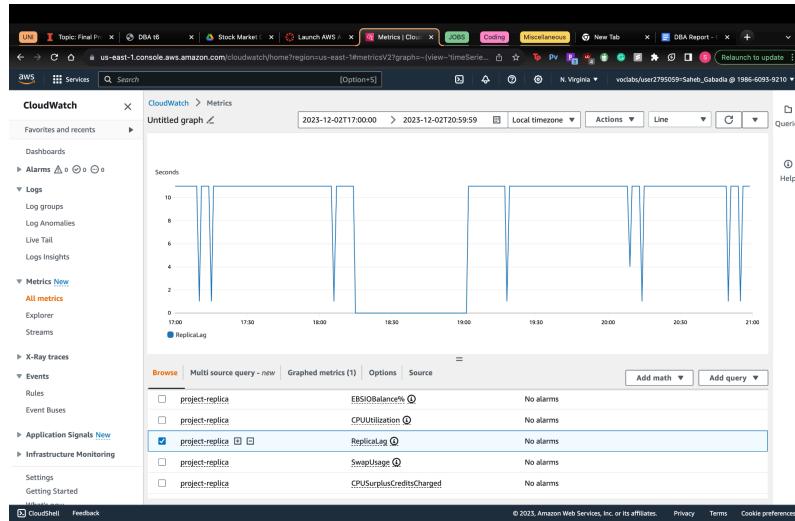


The above pictures monitor the performance of the replica RDS database when a query is executed. We fired the exact same query on both databases.

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.

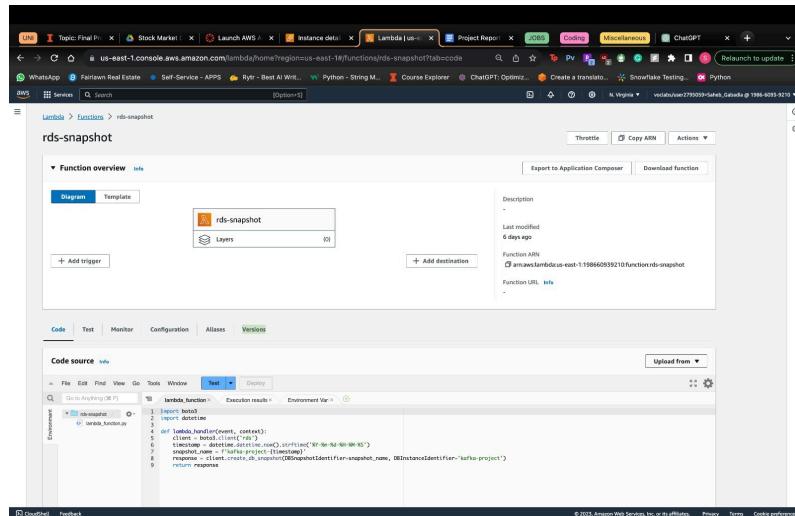
The CPU Utilization had an instant spike, increasing by about 1.5%.

Replica Set Lag:



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

Backup



Schedule detail

Schedule name	lambda-snapshot	Status	Enabled
Description	-	Schedule start time	-
Schedule ARN	arn:aws:scheduler:us-east-1:198660503910:schedule/default/lambda-snapshot	Schedule end time	-
Schedule group name	default	Execution time zone	America/Chicago
Action after completion	DELETE	Last modified date	Nov 26, 2023, 19:16:42 (UTC-06:00)

Schedule

Cron expression: `20 19 * * ? *`

Copy cron expression

Note: 10 trigger date
Date and time are displayed in the selected time zone for which this scheduled event is defined. See, [AWS Lambda Cron Expression Syntax](#).

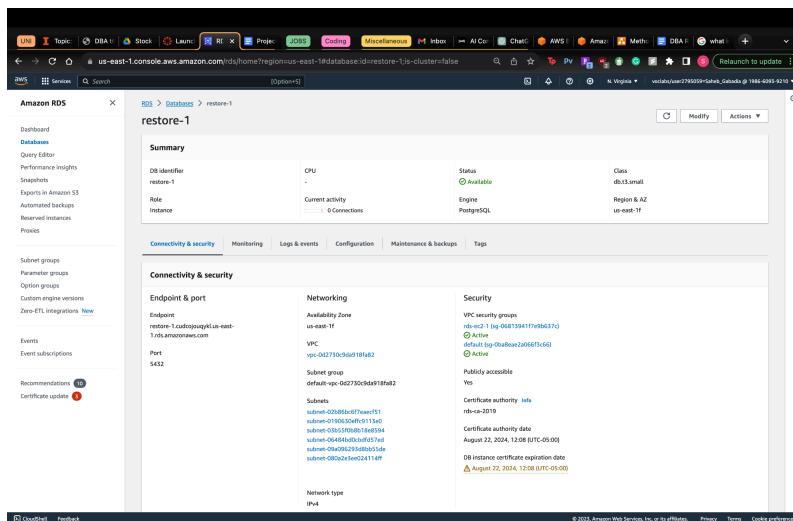
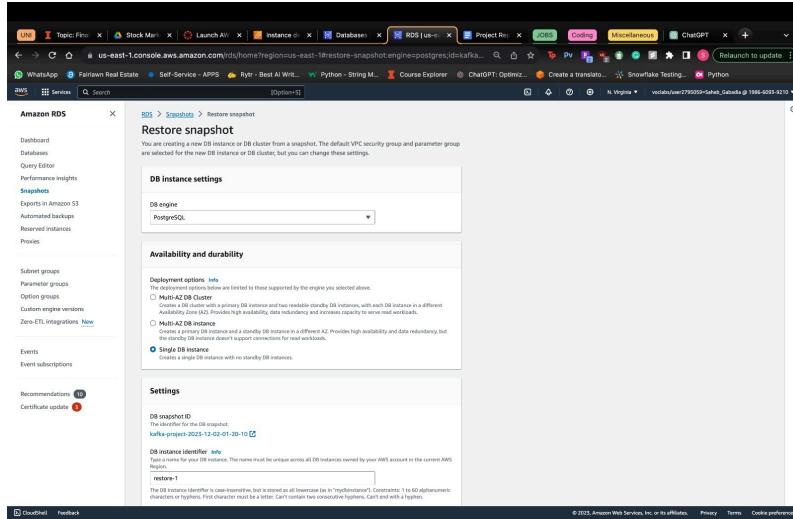
Sat, 02 Dec 2023 19:20:00 (UTC-06:00)
Sun, 03 Dec 2023 19:21:00 (UTC-06:00)
Mon, 04 Dec 2023 19:22:00 (UTC-06:00)
Tue, 05 Dec 2023 19:23:00 (UTC-06:00)
Wed, 06 Dec 2023 19:24:00 (UTC-06:00)
Thu, 07 Dec 2023 19:25:00 (UTC-06:00)

Snapshots

Snapshot name	DB instance or cluster	Snapshot creation time	DB Inst.
kafka-project-2023-12-02-22-47-30	kafka-project	December 02, 2023, 16:47 (UTC-06:00)	Never
kafka-project-2023-12-02-01-20-10	kafka-project	December 01, 2023, 19:20 (UTC-06:00)	Never
kafka-project-2023-12-01-01-20-10	kafka-project	November 30, 2023, 19:21 (UTC-06:00)	Never
kafka-project-2023-11-30-01-20-05	kafka-project	November 29, 2023, 19:20 (UTC-06:00)	Never
kafka-project-2023-11-29-01-20-05	kafka-project	November 28, 2023, 19:20 (UTC-06:00)	Never
kafka-project-2023-11-28-01-20-05	kafka-project	November 27, 2023, 19:21 (UTC-06:00)	Never
kafka-project-2023-11-27-01-23-19	kafka-project	November 26, 2023, 19:23 (UTC-06:00)	Never
kafka-project-2023-11-27-01-20-10	kafka-project	November 26, 2023, 19:20 (UTC-06:00)	Never

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



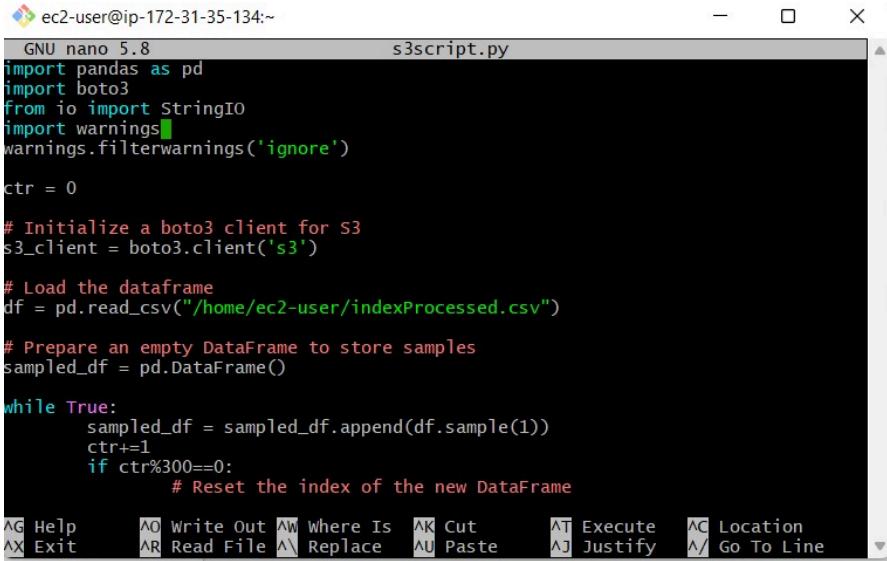
For restoring a backup in RDS, the snapshot has to be restored as a RDS instance. This is a new RDS instance that is created, so it uses the same space as the main database and levies the same cost again. The above pictures show the restoration of a snapshot and the restored snapshot.

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



```
GNU nano 5.8          s3script.py
import pandas as pd
import boto3
from io import StringIO
import warnings
warnings.filterwarnings('ignore')

ctr = 0

# Initialize a boto3 client for S3
s3_client = boto3.client('s3')

# Load the dataframe
df = pd.read_csv("/home/ec2-user/indexProcessed.csv")

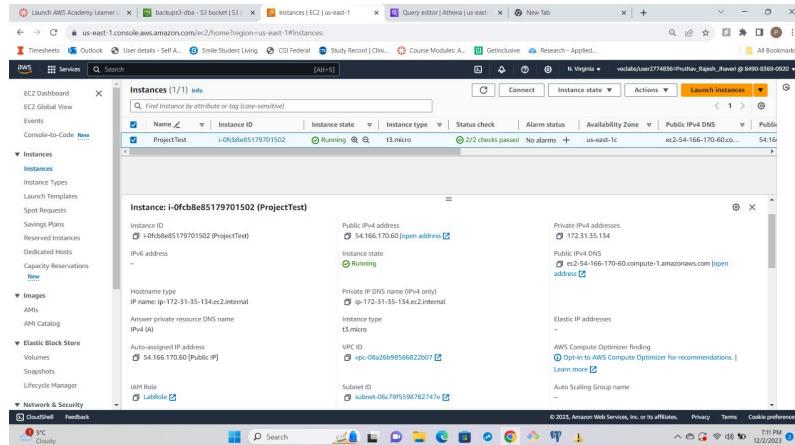
# Prepare an empty DataFrame to store samples
sampled_df = pd.DataFrame()

while True:
    sampled_df = sampled_df.append(df.sample(1))
    ctr+=1
    if ctr%300==0:
        # Reset the index of the new DataFrame
        sampled_df = sampled_df.reset_index(drop=True)

AG Help      AO Write Out  AW Where Is  AK Cut      AT Execute  AC Location
AX Exit      AR Read File  AX Replace  AU Paste     AJ Justify  AV Go To Line
```

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 screenshot shows the AWS S3 Buckets console. On the left, there's a sidebar with navigation links like 'Amazon S3', 'Buckets', 'Access Grants', etc. The main area displays bucket statistics: Total storage (1.8 GB), Object count (45), and Average object size (40.0 MB). A note says you can enable advanced metrics in the 'default-account-dashboard' configuration. Below this, there are tabs for 'General purpose buckets' and 'Directory buckets'. Under 'General purpose buckets', there's a table showing four buckets: 'backups3-dba', 'dbaprojectest', 'replica1-s3', and 'replica2-s3'. Each entry includes the AWS Region (us-east-1), Access (Objects can be public), and Creation date.

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.

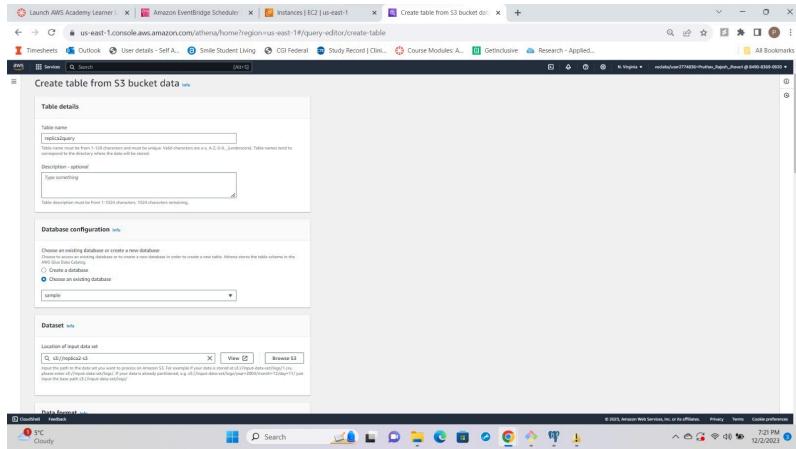
S3 replication sets

The screenshot shows the AWS S3 Replication Rules console. The sidebar has links for 'Amazon S3', 'Buckets', 'Access Grants', etc. The main area shows a message: 'There are no lifecycle rules for this bucket.' Below it is a 'Create lifecycle rule' button. Further down, there's a section for 'Replication rules (2)'. It says, 'Use replication rules to define options you want Amazon S3 to apply during replication such as server-side encryption, replica ownership, transitioning replicas to another storage class, and more.' There are two replication rules listed:

Replication rule name	Status	Destination bucket	Destination Region	Priority	Scope	Storage class	Replica owner	Replication Time Control	KMS-encrypted objects (SSE-KMS or DSSE-KMS)
toreplica1_s3	Enabled	s3://replica1-s3	US East (N. Virginia) us-east-1	0	Entire bucket	Same as source	Disabled	Replicate	
toreplica2s3	Enabled	s3://replica2-s3	US East (N. Virginia) us-east-1	1	Entire bucket	Same as source	Disabled	Replicate	

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

A screenshot of the Amazon Athena Query editor. A query named 'Query 15' is displayed, showing a complex SQL statement involving window functions and over-partitioning. The results pane shows 136,536 rows. The results table includes columns such as 'Index', 'Date', 'Close', 'moving_avg_close_5d', 'moving_avg_close_30d', 'stddevs_close_30d', 'max_high', 'min_low', and 'avg_value'. The time taken for the query is 102 ms and the run time is 7.65 sec.

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

The screenshot shows the AWS Athena Query Editor interface. A query is being run against a database named 'sample'. The query is:

```
1 select * from sample;"replicaquery"
```

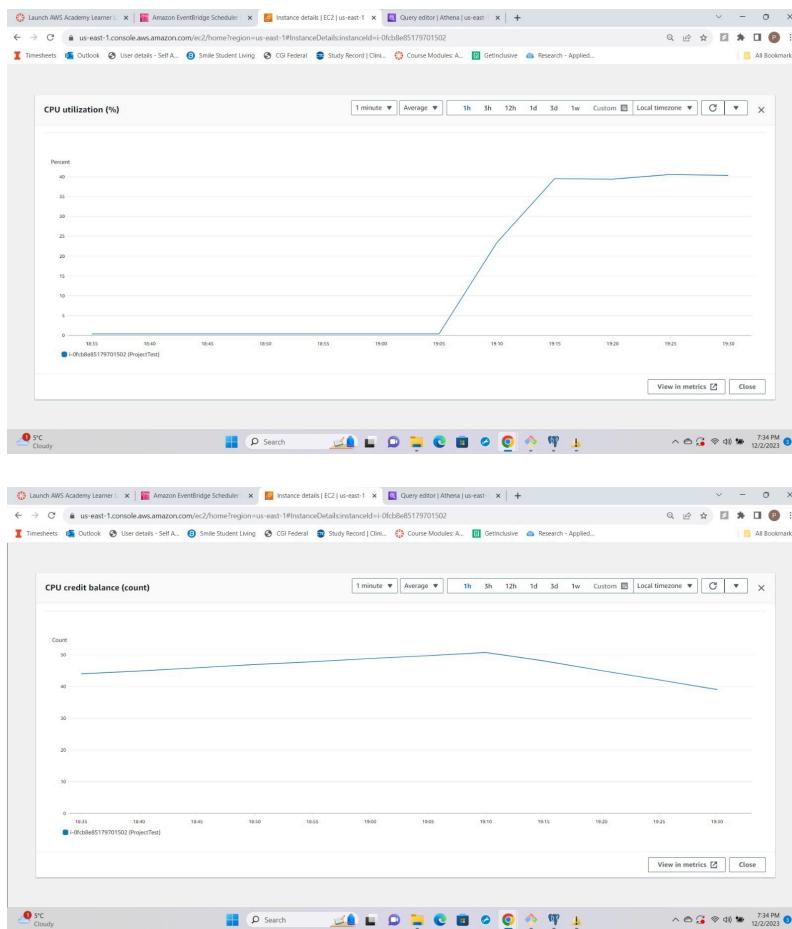
The results are displayed in a table titled 'Completed' with 8 rows of data. The columns are: index, date, open, high, low, close, adj_close, volume, and close_pct. The data shows price movements over time.

index	date	open	high	low	close	adj_close	volume	close_pct
1	2010-06-13	3219.4	3219.4	3219.4	3219.4	3219.4	0.0	42.79
2	2010-06-15	31701.39	31880.15	31641.05	31702.71	31702.71	1.96705689	203.05
3	2010-06-18	4177.2	4177.2	4121.5	4166.6	4166.6	0.0	12.84
4	2010-06-21	8212.8	8212.8	8190.0	8210.0	8210.0	0.0	12.01
5	2010-06-22	21028.39	21041.54	21030.20	21031.96	21031.96	1.37772463	270.45
6	2010-06-23	50796.14	50817.43	50784.63	50805.66	50805.66	0.0	142.54
7	2010-06-24	21050.02	20514.76	20511.7	20517.24	20517.24	1.95116919	17.54

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.

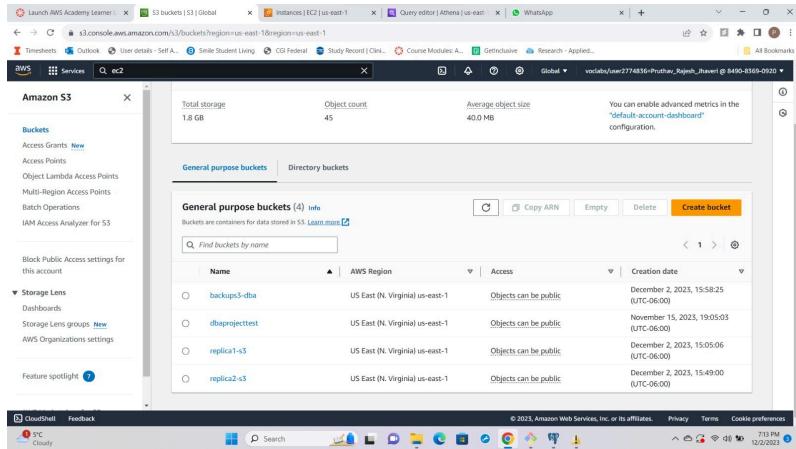
The CPU credit balance starts to decrease from 19:10, indicating the amount of credit being used.

Backup

The image contains two screenshots of the AWS console. The top screenshot shows the Lambda function overview for 'dbas3backup'. It displays the function name, code source (a file named 'index.js' containing a CRON-based Lambda function), and execution results. The bottom screenshot shows the Amazon EventBridge schedule for the same function. It details a scheduled event named 'dbas3backup' with a CRON expression of '0 17 * * *' (每天凌晨17:00执行) and a target of the Lambda function.

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



The screenshot shows the AWS S3 console interface. At the top, there are tabs for Launch AWS Academy Learner, S3 buckets (Global), Instances | EC2 (us-east-1), Query editor | Athena (us-east-1), and WhatsApp. Below the tabs, the main header says "Amazon S3" and "Buckets". On the left sidebar, there are sections for Access Grants (New), Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Block Public Access settings for this account, Storage Lens, Dashboards, Storage Lens groups (New), and AWS Organizations settings. A "Feature spotlight" section is also present. The main content area displays "General purpose buckets (4)" with the following details:

Name	AWS Region	Access	Creation date
backupsS3-dba	US East (N. Virginia) us-east-1	Objects can be public	December 2, 2023, 15:58:25 (UTC-06:00)
dbaprojecttest	US East (N. Virginia) us-east-1	Objects can be public	November 15, 2023, 19:05:03 (UTC-06:00)
replica1-s3	US East (N. Virginia) us-east-1	Objects can be public	December 2, 2023, 15:05:06 (UTC-06:00)
replica2-s3	US East (N. Virginia) us-east-1	Objects can be public	December 2, 2023, 15:49:00 (UTC-06:00)

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-sched
ule.html](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/>