Introduction to Big Data
Final Assignment

Name - Avijeet Palit
Roll - 21f1005675
Date - 15/12/2024

Objective

The goal of the assignment is to identify and emit anomalies in historical stock trading data using Spark Streaming. The anomalies are defined as trades that significantly deviate in price or volume, helping a fraud control unit detect suspicious trading activities for further scrutiny.

Breakdown of the Assignment

1. Data Preparation and Loading

- Load minute-level stock trading data (price and volume) for a 3-year period (2017–2020).
- Stream data by loading into Kafka using Spark Batch.
- Ensure data quality checks to handle bad rows, unsorted timestamps, and data gaps.

2. Real-Time Stream Processing

- Write Spark Streaming code to consume data minute-by-minute from Kafka.
- Use window functions for anomaly detection:
    - **A1:** Detect price deviations exceeding ±0.5% compared to the previous minute.
    - **A2:** Detect volume spikes exceeding 2% of the average traded volume for the last 10 minutes.
- Emit detected anomalies with relevant details (trade data and anomaly type).

Detailed Steps

Step 1: Create a GCP Project

- Log in to the Google Cloud Console.
- Create a new project or use an existing one.
- Enable the required APIs:
    - **Cloud Storage API**
    - **Dataproc API**
    - **Compute Engine API**

Step 2: Download the data set from g-drive

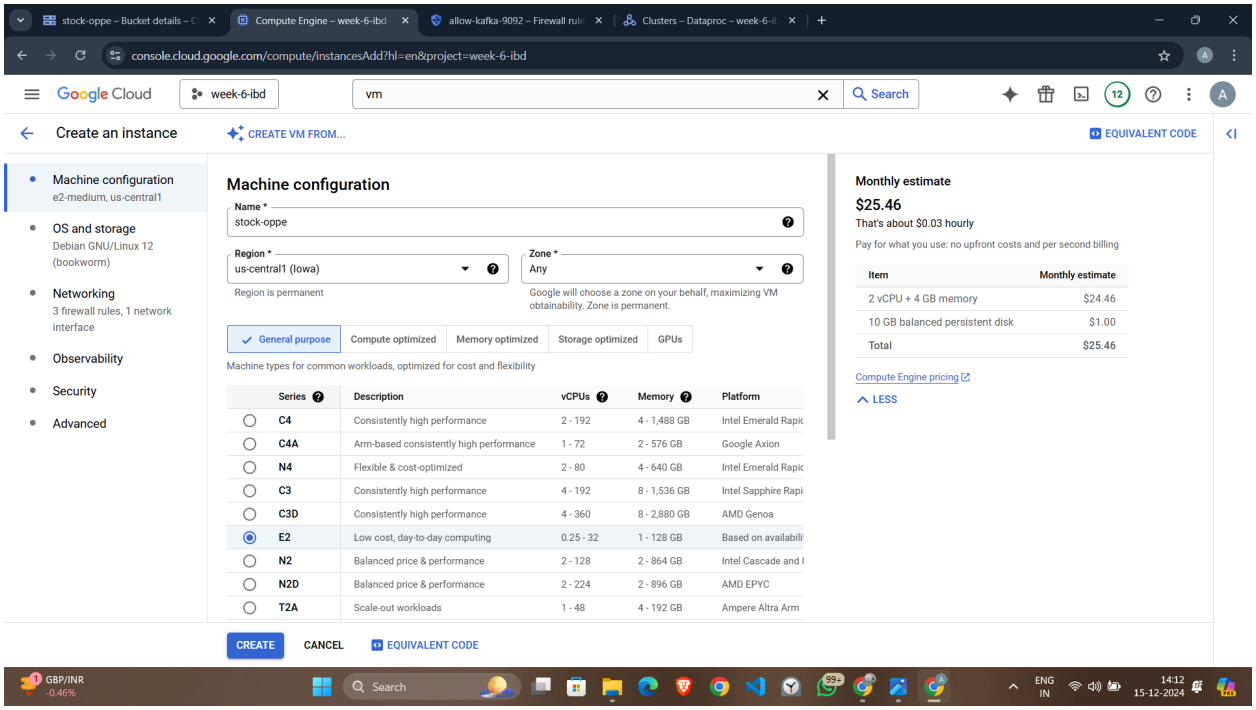
- Download the dataset
- Unzip the folder
- Manually analyze the dataset for better understanding.

TATACHEM__EQ__NSE__NSE__MINUTE.csv

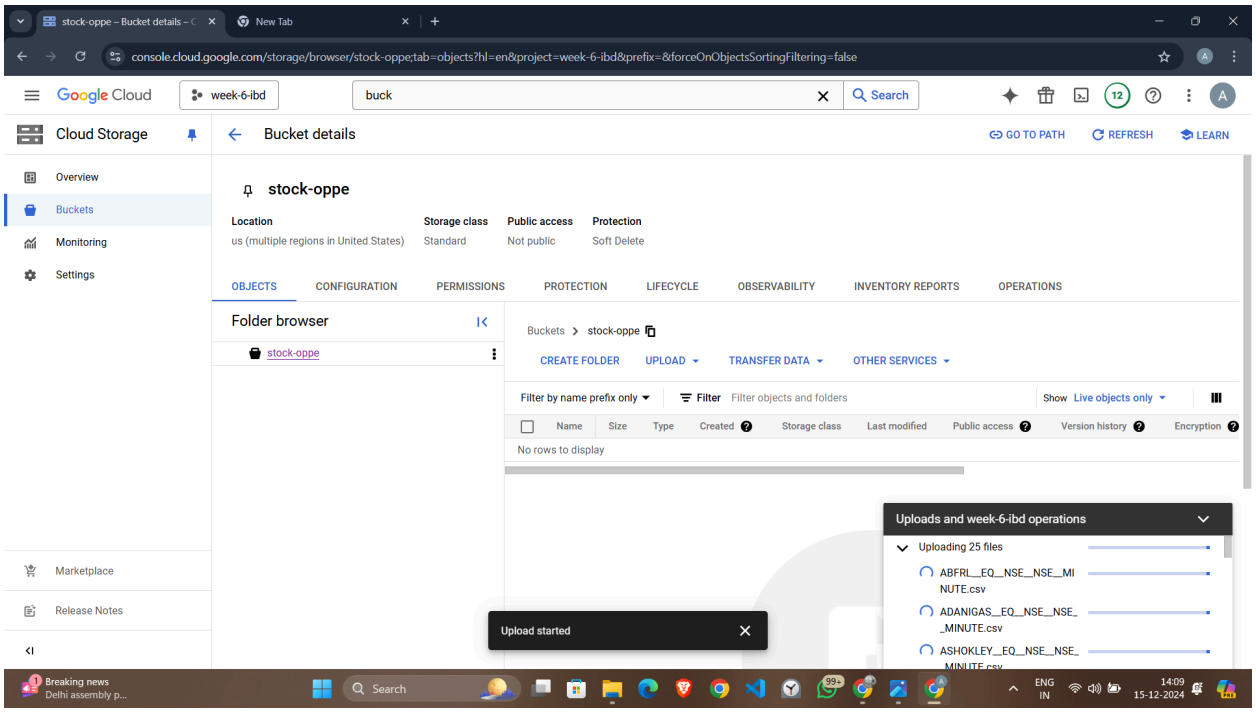
A	B	C	D	E	F
timestamp	open	high	low	close	volume
2017-01-02 09:15:00	506.55	509.65	505.45	508.85	12277
2017-01-02 09:16:00	508.35	509.95	508.35	509	8493
2017-01-02 09:17:00	509	509.5	508.75	509.4	4353
2017-01-02 09:18:00	509.4	509.4	508.35	508.4	6385
2017-01-02 09:19:00	508.65	509.45	508.25	508.5	9450
2017-01-02 09:20:00	508.3	509.2	508.3	508.5	10005
2017-01-02 09:21:00	508.5	509	507.9	508.25	5399
2017-01-02 09:22:00	508	508.75	507.9	508.75	4929
2017-01-02 09:23:00	508.75	510.65	508.75	510.25	12241
2017-01-02 09:24:00	509.8	511	509.55	510.85	5535
2017-01-02 09:25:00	511	511.65	510.4	510.4	9080
2017-01-02 09:26:00	509.95	510.8	509.25	510.65	10544
2017-01-02 09:27:00	510.6	510.6	509.9	510.05	13373
2017-01-02 09:28:00	510	510.4	509.7	509.7	4336
2017-01-02 09:29:00	509.7	510.3	509.4	510.3	5295
2017-01-02 09:30:00	510.3	510.45	509.2	509.6	5007
2017-01-02 09:31:00	509.6	509.9	509.05	509.25	3545
2017-01-02 09:32:00	509.25	509.85	509	509	5210
2017-01-02 09:33:00	509	509.85	509	509.45	1279
2017-01-02 09:34:00	509.45	511.6	509.45	511.55	7213
2017-01-02 09:35:00	511.25	511.5	510	510.65	3495
2017-01-02 09:36:00	510.65	511	510.2	510.4	3323
2017-01-02 09:37:00	510.65	510.7	510	510.45	3942
2017-01-02 09:38:00	510.4	511.45	510.4	511	2703
2017-01-02 09:39:00	511	511	510.5	511	2701
2017-01-02 09:40:00	511	511.95	510.5	511.35	4858
2017-01-02 09:41:00	511.35	512	510.6	511.95	3939
2017-01-02 09:42:00	512	513	512	512.6	8538

Step 3: Set Up Google Cloud Storage (GCS)

- Create a Cloud Storage bucket for storing images:
  - Go to **Cloud Storage** > **Buckets**.
  - Create a bucket (e.g., `stock-oppe`) with the appropriate region and permissions

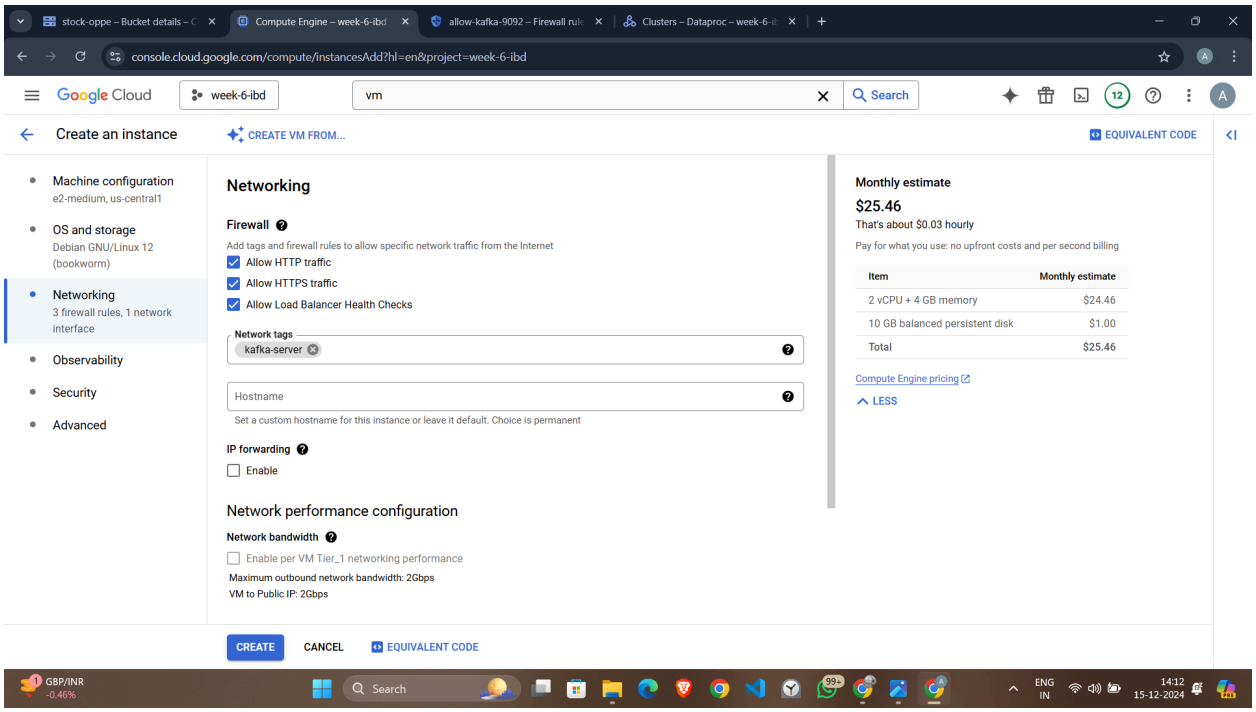


- Upload some test images to the bucket.
- Upload install_packages.sh



Step 4: Create a Virtual Machine (VM):

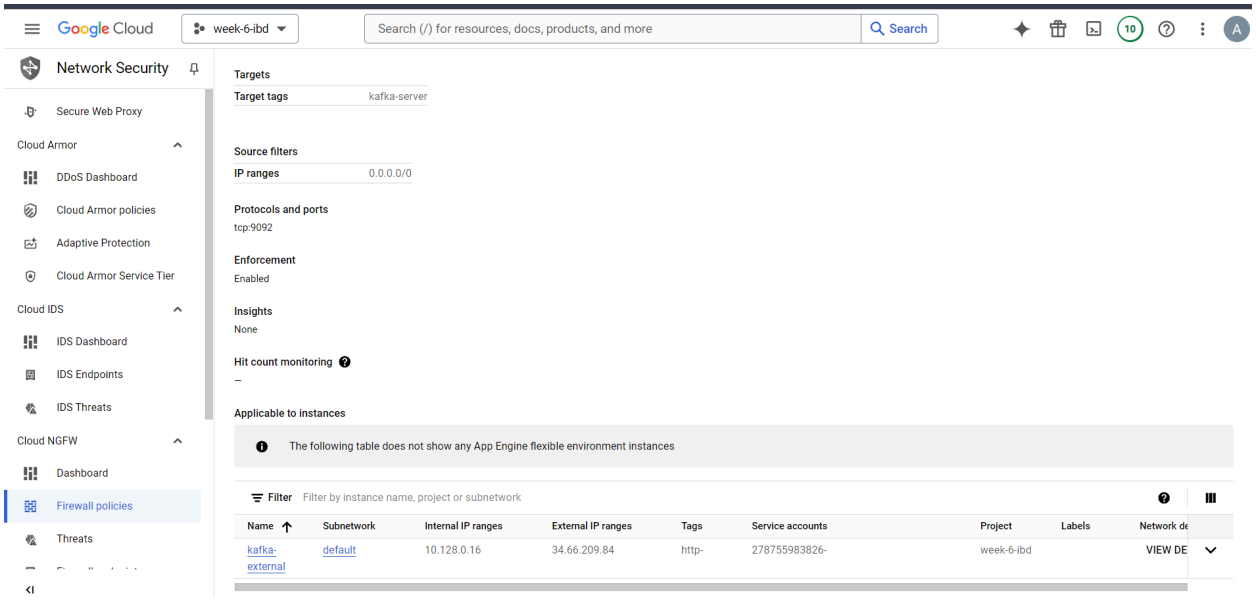
- Navigate to **Compute Engine > VM Instances**.
- Click **Create Instance** “stock-oppe”
- Choose:
    - Machine type (e.g., e2-medium for basic setups).
    - Operating system (e.g., Ubuntu 22.04 LTS).
- Configure networking to allow external access:
    - Under **Firewall**, check "Allow HTTP traffic" and "Allow HTTPS traffic".



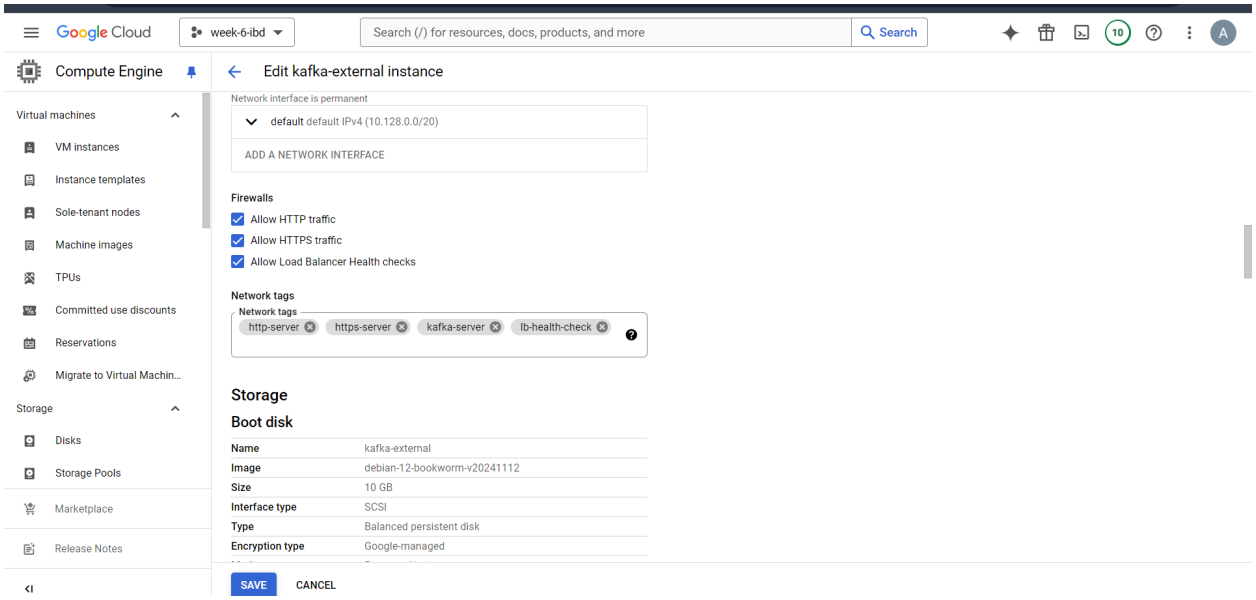
Step 5: Create a New Firewall Rule

- Click the **"Create Firewall Rule"** button at the top.
- Fill in the details for the new rule:
    - **Name**: Enter a descriptive name, such as `allow-kafka-9092`.
    - **Network**: Select the network your VM instance is part of (usually `default` unless you've set up a custom network).
    - **Priority**: Leave the default value (1000) unless you have specific needs.
    - **Direction of traffic**: Select **Ingress** (incoming traffic).
    - **Action on match**: Select **Allow**.
    - **Targets**:
        - Choose **Specified target tags**.
        - Add a **tag** (e.g., `kafka-server`) that you will assign to your VM later.
- **Source filter**:
    - Choose **IP ranges**.

    - Enter the IP range allowed to access your Kafka server:
        - For open access: `0.0.0.0/0` (not recommended for production environments due to security risks).
        - For restricted access: Use a specific IP or range, e.g., `192.168.1.0/24`.
- **Protocols and ports**:
    - Select **Specified protocols and ports**.
    - Check **tcp** and specify port `9092`.
- **Save the Firewall Rule**
    - Click the **Create** button to save the rule.



- **Tag Your VM with the Firewall Rule**
    - Go to **Compute Engine > VM Instances**.
    - Find the VM running Kafka.
    - Click the **Edit** button at the top of the VM details page.
    - In the **Network tags** section, add the same tag you used in the firewall rule (e.g., `kafka-server`).
    - Click **Save**.



Step 6: Install Kafka and Dependencies

- **Connect to the VM: SSH into the VM**
  - Install Java: Kafka requires Java. Install OpenJDK:

    ```
    sudo apt update
    sudo apt install -y default-jdk
    java -version
    ```
- **Download and Install Kafka:**
  - Navigate to Apache Kafka Downloads.
  - Download Kafka:

    ```
    Wget
    https://downloads.apache.org/kafka/3.5.1/kafka_2.13-3.5.1.tgz
    ```
  - Extract the Kafka archive:

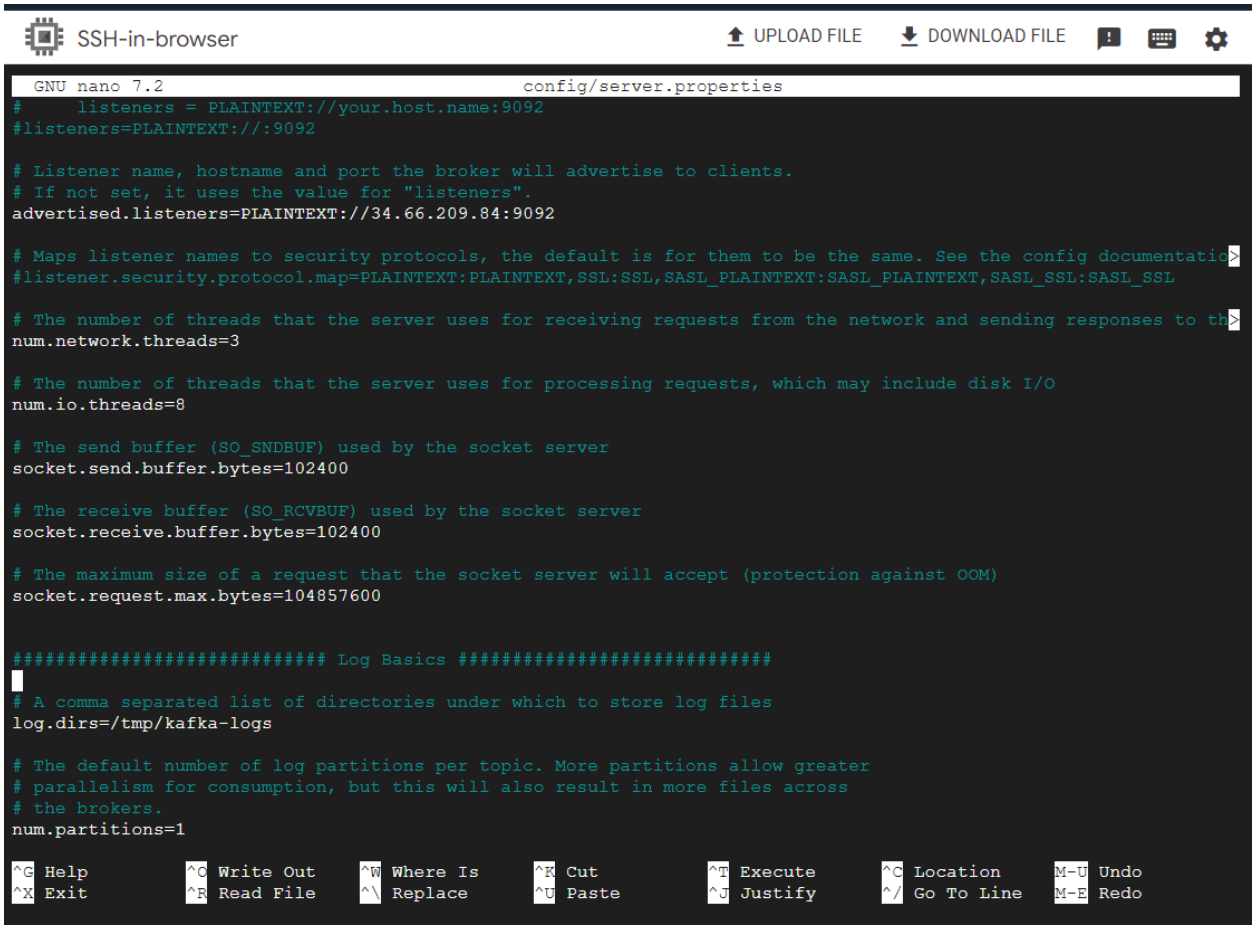
    ```
    tar -xvzf kafka_2.13-3.5.1.tgz
    mv kafka_2.13-3.5.1 kafka
    ```
- **Configure Kafka:**
  - Open Kafka configuration file:

    ```
    nano kafka/config/server.properties
    ```
  - Set the following:
  - Advertised Hostname:

    ```
    advertised.listeners=PLAINTEXT://34.28.48.166:9092
    ```
  - Zookeeper (default):

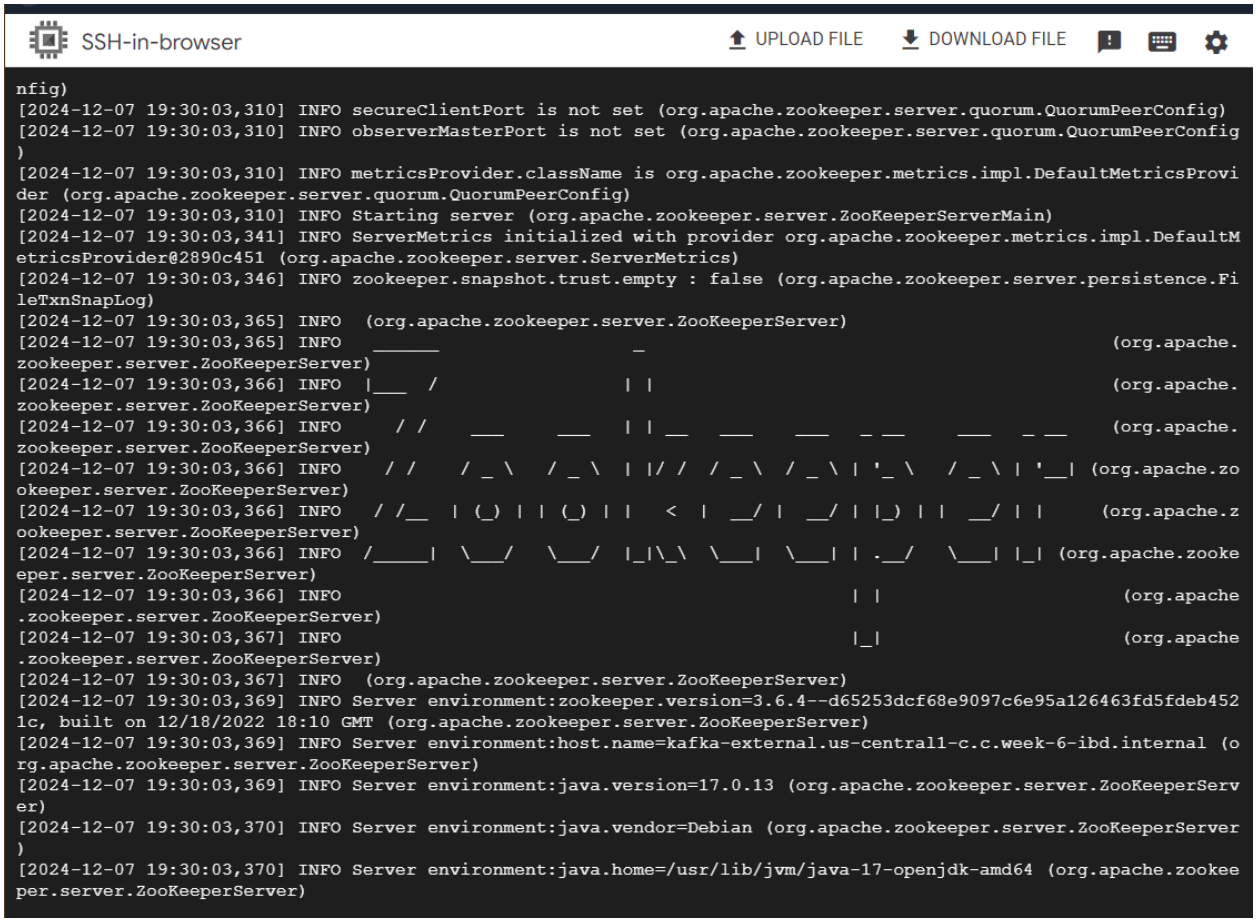
    ```
    zookeeper.connect=localhost:2181
    ```

- Save and exit.



Step 7: Set Up and Start Zookeeper

- Kafka requires Zookeeper for coordination.
- Start Zookeeper:
  `kafka/bin/zookeeper-server-start.sh kafka/config/zookeeper.properties &`



Step 8: Set Up Kafka

- Start Kafka Broker:
  `kafka/bin/kafka-server-start.sh kafka/config/server.properties &`

```
[2024-12-07 19:31:03,066] INFO [GroupCoordinator 0]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2024-12-07 19:31:03,127] INFO [TransactionCoordinator id=0] Starting up. (kafka.coordinator.transaction.Transact
ionCoordinator)
[2024-12-07 19:31:03,145] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.Tra
nsactionCoordinator)
[2024-12-07 19:31:03,154] INFO [TxnMarkerSenderThread-0]: Starting (kafka.coordinator.transaction.TransactionMark
erChannelManager)
[2024-12-07 19:31:03,281] INFO [Controller id=0, targetBrokerId=0] Node 0 disconnected. (org.apache.kafka.clients
.NetworkClient)
[2024-12-07 19:31:03,284] WARN [Controller id=0, targetBrokerId=0] Connection to node 0 (/34.66.209.84:9092) coul
d not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-07 19:31:03,287] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$E
xpiredOperationReaper)
[2024-12-07 19:31:03,292] INFO [Controller id=0, targetBrokerId=0] Client requested connection close from node 0
(org.apache.kafka.clients.NetworkClient)
[2024-12-07 19:31:03,388] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotific
ationListener$ChangeEventProcessThread)
[2024-12-07 19:31:03,399] INFO [Controller id=0, targetBrokerId=0] Node 0 disconnected. (org.apache.kafka.clients
.NetworkClient)
[2024-12-07 19:31:03,401] WARN [Controller id=0, targetBrokerId=0] Connection to node 0 (/34.66.209.84:9092) coul
d not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-12-07 19:31:03,402] INFO [Controller id=0, targetBrokerId=0] Client requested connection close from node 0
(org.apache.kafka.clients.NetworkClient)
[2024-12-07 19:31:03,421] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Enabling request processing. (kafk
a.network.SocketServer)
[2024-12-07 19:31:03,426] INFO Awaiting socket connections on 0.0.0.0:9092. (kafka.network.DataPlaneAcceptor)
[2024-12-07 19:31:03,439] INFO Kafka version: 3.5.1 (org.apache.kafka.common.utils.AppInfoParser)
[2024-12-07 19:31:03,440] INFO Kafka commitId: 2c6fb6c54472e90a (org.apache.kafka.common.utils.AppInfoParser)
[2024-12-07 19:31:03,440] INFO Kafka startTimeMs: 1733599863433 (org.apache.kafka.common.utils.AppInfoParser)
[2024-12-07 19:31:03,442] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
[2024-12-07 19:31:03,598] INFO [zk-broker-0-to-controller-forwarding-channel-manager]: Recorded new controller, f
rom now on will use node 34.66.209.84:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
[2024-12-07 19:31:03,639] INFO [zk-broker-0-to-controller-alter-partition-channel-manager]: Recorded new controll
er, from now on will use node 34.66.209.84:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)
[2024-12-07 19:31:03,708] INFO [ReplicaFetcherManager on broker 0] Removed fetcher for partitions Set(image-topic
-0) (kafka.server.ReplicaFetcherManager)
[2024-12-07 19:31:03,752] INFO [Partition image-topic-0 broker=0] Log loaded for partition image-topic-0 with ini
tial high watermark 815 (kafka.cluster.Partition)
```

- Create a topic:
  `kafka/bin/kafka-topics.sh --create --topic stock-oppe --bootstrap-server 34.28.48.166:9092`
- List topics:
  `kafka/bin/kafka-topics.sh --list --bootstrap-server 34.28.48.166:9092`

Step 9: Kafka Producer

- Write a **Kafka producer** script and upload it to the VM ssh
- Install the python virtual environment and activate it
  - `sudo apt update`
  - `sudo apt install python3-venv`
  - `python3 -m venv env`
  - `source env/bin/activate`
- Install the Kafka Python client:
  `pip install kafka-python google-cloud-storage pandas`
- Run the producer.py
  - `Python3 producer.py`

```python
from google.cloud import storage
import pandas as pd
from kafka import KafkaProducer
import json
import time
from io import BytesIO


producer = KafkaProducer(
    bootstrap_servers='34.28.48.166:9092',
    acks='all',
    value_serializer=lambda v: json.dumps(v).encode('utf-8')  # Ensure it's properly serialized
)

client = storage.Client()
bucket = client.bucket("stock-oppe")
blobs = bucket.list_blobs(prefix="NSE_Stocks_Data/")

csv_files = {}
for blob in blobs:
    if blob.name.endswith(".csv"):
        print(f"Downloading: {blob.name}")
        content = blob.download_as_bytes()
        df = pd.read_csv(BytesIO(content))
        csv_files[blob.name] = df

dataframes = {}

for file_name, df in csv_files.items():
    company_name = file_name.split("/")[-1]
    company_name = company_name.split("_")[0]
    df['company'] = company_name
    dataframes[file_name] = df

# Initialize sliding windows for each CSV file
windows = {file_name: [] for file_name in csv_files}
indices = {file_name: 0 for file_name in csv_files}

batch_number = 0
while True:
    batch_number += 1
    print(f"Producing batch #{batch_number}...")

```

```
    for file_name, df in dataframes.items():
        current_index = indices[file_name]
        if current_index >= len(df):
            continue
        windows[file_name].append(df.iloc[current_index].to_dict())
        if len(windows[file_name]) > 10:
            windows[file_name].pop(0)

        indices[file_name] += 1
        windowed_df = pd.DataFrame(windows[file_name])
        windowed_data = windowed_df.groupby('company').agg(
            avg_volume=('volume', 'mean'),
            last_close=('close', 'max')
        ).reset_index()

        merged_df = windowed_df.merge(windowed_data, on='company', how='left')

        merged_df['A1'] = abs((merged_df['close'] - merged_df['last_close']) / merged_df['last_close']) > 0.005
        merged_df['A2'] = merged_df['volume'] > merged_df['avg_volume'] * 1.02
        merged_df['is_anomaly'] = merged_df['A1'] | merged_df['A2']

        anomalies = merged_df[merged_df['is_anomaly']]

        for index, row in anomalies.iterrows():
            record = {
                "company": row["company"],
                "open": row["open"],
                "high": row["high"],
                "low": row["low"],
                "close": row["close"],
                "volume": row["volume"],
                "avg_volume": row["avg_volume"],
                "last_close": row["last_close"],
                "A1": row["A1"],
                "A2": row["A2"],
                "is_anomaly": row["is_anomaly"]
            }
            print(f"Sending record: {record}")
            producer.send("stock-oppe", record)

    if all(indices[file_name] >= len(dataframes[file_name]) for file_name in csv_files):
        print("All data processed. Exiting.")
        break
```

```
    if all(indices[file_name] >= len(dataframes[file_name]) for file_name in csv_files):
        print("All data processed. Exiting.")
        break

    producer.flush()
    time.sleep(20)

producer.close()
```

Step 10: Create a Dataproc Cluster

- Go to **Dataproc** > **Clusters** > **Create Cluster**.
- Choose:
    - Cluster type: **Standard (1 master, N workers)**.
    - Machine type: **e2-standard-4** (adjust based on workload).
    - Enable components like **Jupyter Notebook** if needed.
    - Add **Initialization actions** with `gs://stock-oppe/install_packages.sh`

```bash
#!/bin/bash
# install_packages.sh

sudo apt-get update
# Install pip (if not already installed)
sudo apt-get install python3-pip -y

# Install required Python packages
pip3 install kafka-python
```

- Check the connection
    - telnet 34.28.48.166 9092

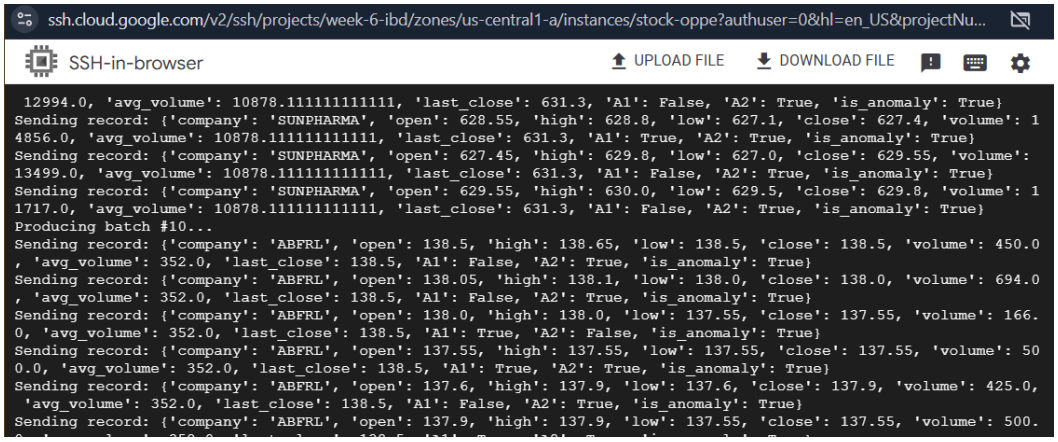
```
avijeetcloud@stock-oppe-m:~$ telnet 34.28.48.166 9092
Trying 34.28.48.166...
Connected to 34.28.48.166.
Escape character is '^]'.
```

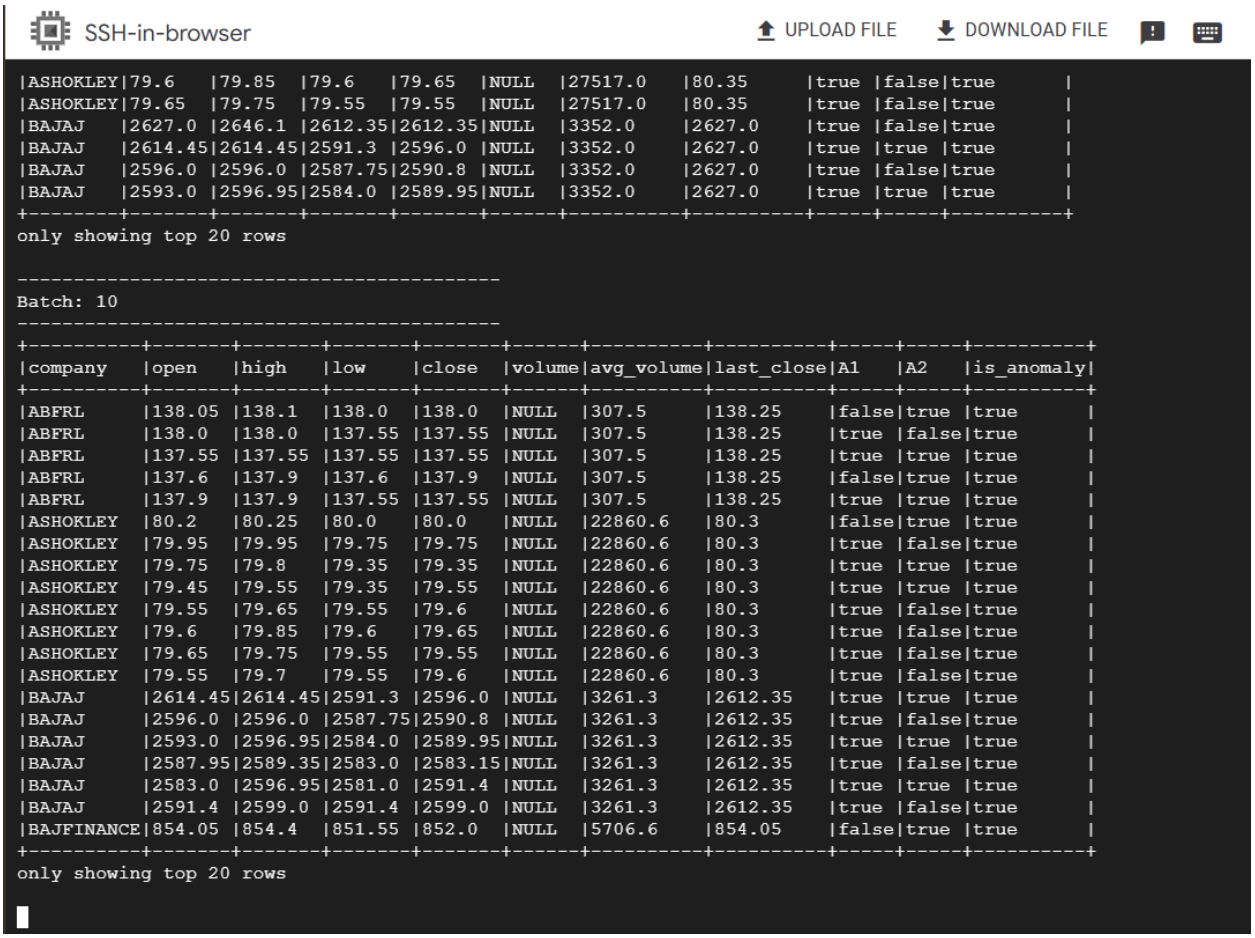
- Write a **Kafka consumer "Stream.py**" script and uploads it to the VM ssh.
- Run the stream.py
    - Python3 stream.py

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, IntegerType, BooleanType

spark = SparkSession.builder \
    .appName("StockAnomalyDetection") \
    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1") \
    .getOrCreate()

json_schema = StructType([
    StructField("company", StringType(), True),
    StructField("open", DoubleType(), True),
    StructField("high", DoubleType(), True),
    StructField("low", DoubleType(), True),
    StructField("close", DoubleType(), True),
    StructField("volume", IntegerType(), True),
    StructField("avg_volume", DoubleType(), True),
    StructField("last_close", DoubleType(), True),
    StructField("A1", BooleanType(), True),
    StructField("A2", BooleanType(), True),
    StructField("is_anomaly", BooleanType(), True)
])

stock_data_stream = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "34.28.48.166:9092") \
    .option("subscribe", "stock-oppe") \
    .option("startingOffsets", "latest") \
    .load() \
    .selectExpr("CAST(value AS STRING) as value")

anomalies = stock_data_stream \
    .select(from_json(col("value"), json_schema).alias("data")) \
    .select("data.*") \
    .filter(col("data").isNotNull())

query = anomalies.writeStream \
    .outputMode("append") \
    .format("console") \
    .option("truncate", "false") \
    .trigger(processingTime="10 seconds") \
    .start()

query.awaitTermination()
```

Step 11: Result and Testing the Pipeline

- **Kafka Testing:**
    - Use the Kafka producer script to push test messages to the Kafka topic.
- **Monitor Outputs:**
    - Check the cluster ssh terminal for real-time predictions.
    - Verify the console output manually checking the dataset





Conclusion

The successful implementation of real-time anomaly detection in historical stock trading data using Spark Streaming and Kafka demonstrates the practical integration of big data technologies to address financial irregularities. By creating a robust data pipeline that ingests, processes, and analyzes minute-level trading data, we effectively identified significant price deviations and volume spikes that could signal suspicious trading activities.

This project not only highlights the capabilities of distributed systems in managing large-scale data but also showcases their potential to enable proactive fraud detection mechanisms. The detailed steps, from setting up infrastructure in Google Cloud Platform to testing and validating the pipeline, ensure a scalable and reliable solution.

Future enhancements could include incorporating machine learning models for anomaly classification, expanding the scope of detection parameters, and automating the pipeline for broader financial applications. This assignment underscores the importance of big data analytics in safeguarding the integrity of financial markets.