Introduction to Big Data
Graded Assignment : 9

Name - Avijeet Palit
Roll - 21f1005675
Date - 04/12/2024

Objective

Convert the batch image classification use case walked through in the class to a real-time execution model using Spark Streaming. The notebook is attached as a separate link in week 10. Run it and record your screen to capture the outputs so as to show proof of the streaming execution.

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 Image data set from tf flower dataset

- Setup the colab
- Download the images
- Save it in a zip folder
- Select some images and store it in a folder “images” in local machine

```
[ ]
import tensorflow as tf

# Download and extract the dataset
data_dir = tf.keras.utils.get_file(
    origin='https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
    fname='flower_photos', untar=True
)

print("Dataset downloaded and extracted at:", data_dir)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
228813984/228813984 ━━━━━━━━━━━━━━━━━━━━ 11s 0us/step
Dataset downloaded and extracted at: /root/.keras/datasets/flower_photos
```

```
import os

# Check extracted files
print("Extracted files:")
for root, dirs, files in os.walk(data_dir):
    print(root, "contains", len(files), "files.")

```

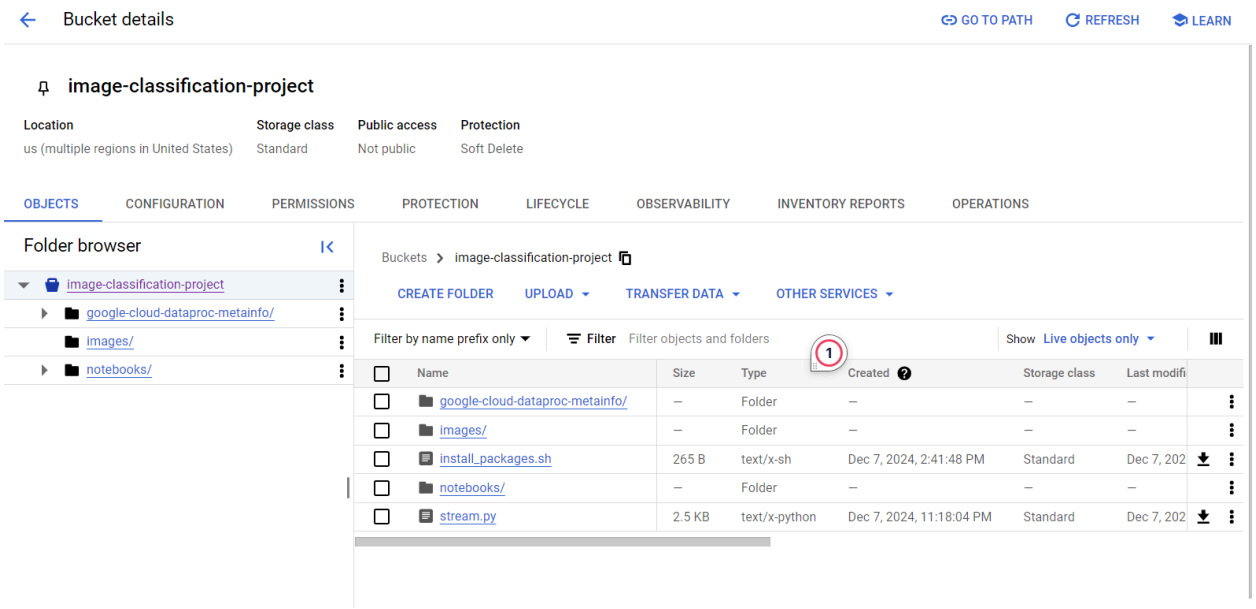
```
Extracted files:
/root/.keras/datasets/flower_photos contains 1 files.
/root/.keras/datasets/flower_photos/roses contains 641 files.
/root/.keras/datasets/flower_photos/tulips contains 799 files.
/root/.keras/datasets/flower_photos/daisy contains 633 files.
/root/.keras/datasets/flower_photos/dandelion contains 898 files.
/root/.keras/datasets/flower_photos/sunflowers contains 699 files.
```

```
[ ]
!zip -r flower_photos.zip /root/.keras/datasets/flower_photos

```

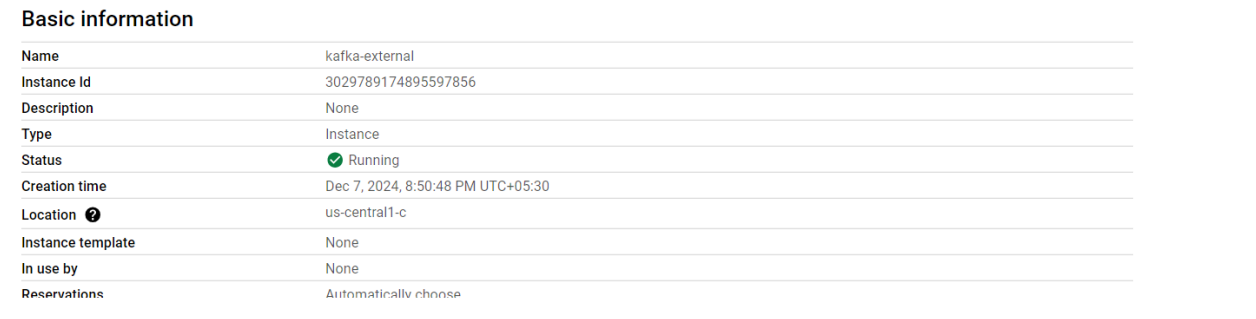
Step 3: Set Up Google Cloud Storage (GCS)

- Create a Cloud Storage bucket for storing images:
  - Go to **Cloud Storage** > **Buckets**.
  - Create a bucket (e.g., `image-classification-project`) with the appropriate region and permissions.
  - Upload some test images to the bucket (e.g., under `images/`).
  - Upload install_packages.sh



Step 4: Create a Virtual Machine (VM):

- Navigate to **Compute Engine > VM Instances**.
- Click **Create Instance** “kafka-external”
- 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".

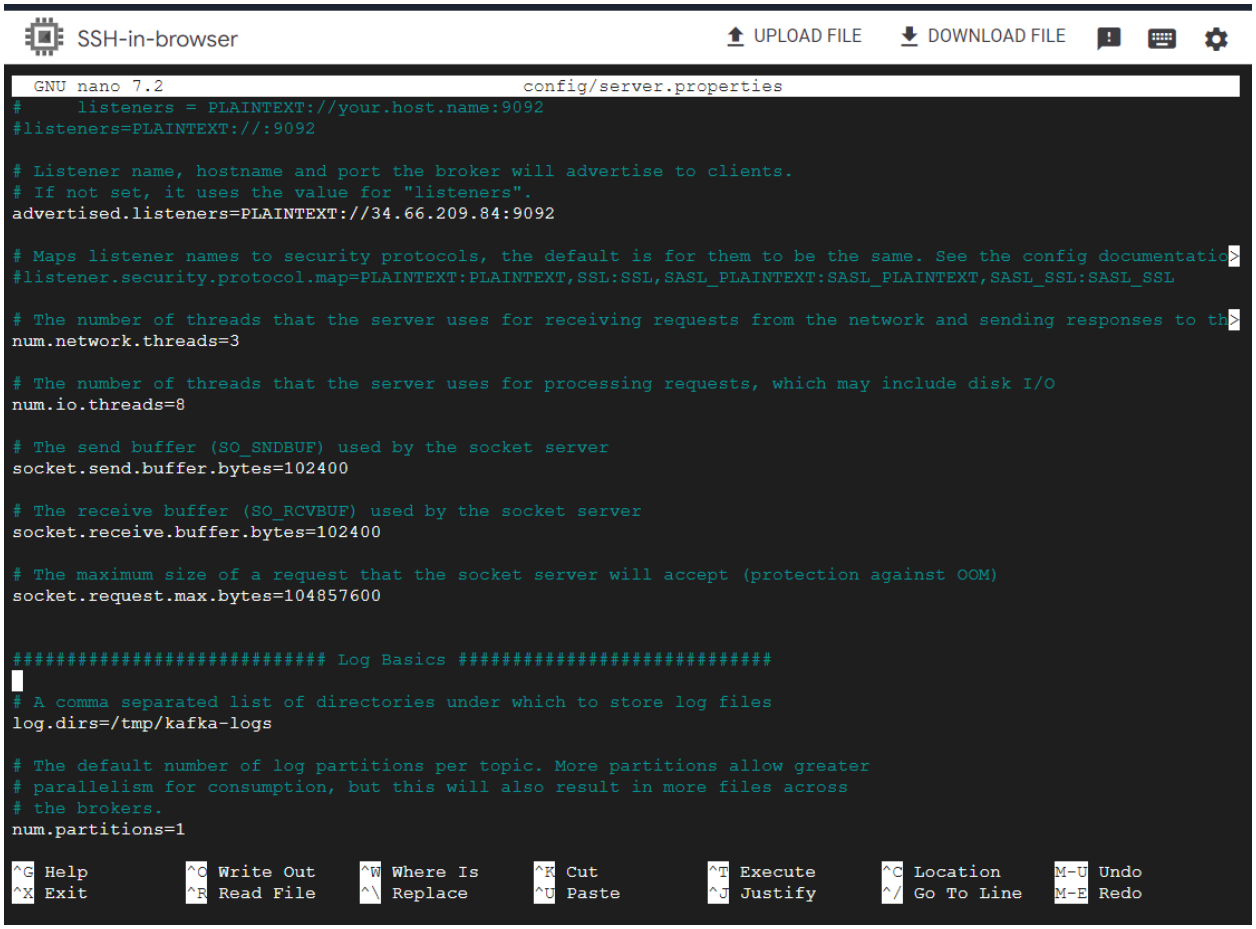


Step 3: Install Kafka and Dependencies

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

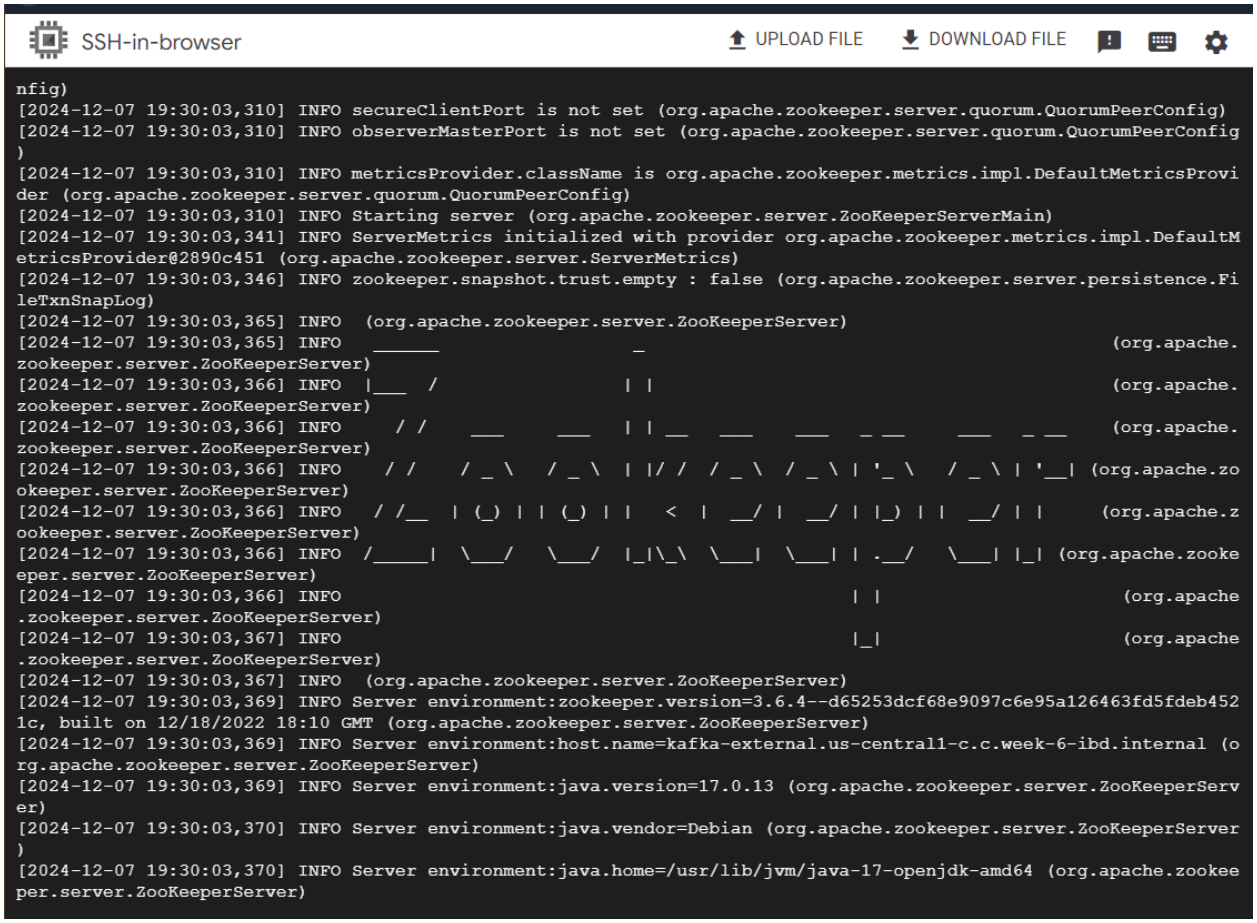
```
sudo apt update
sudo apt install -y default-jdk
java -version
```

- **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://34.66.209.84:9092
    ```
  - Zookeeper (default):
    ```
    zookeeper.connect=localhost:2181
    ```
  - Save and exit.



Step 4: Set Up and Start Zookeeper

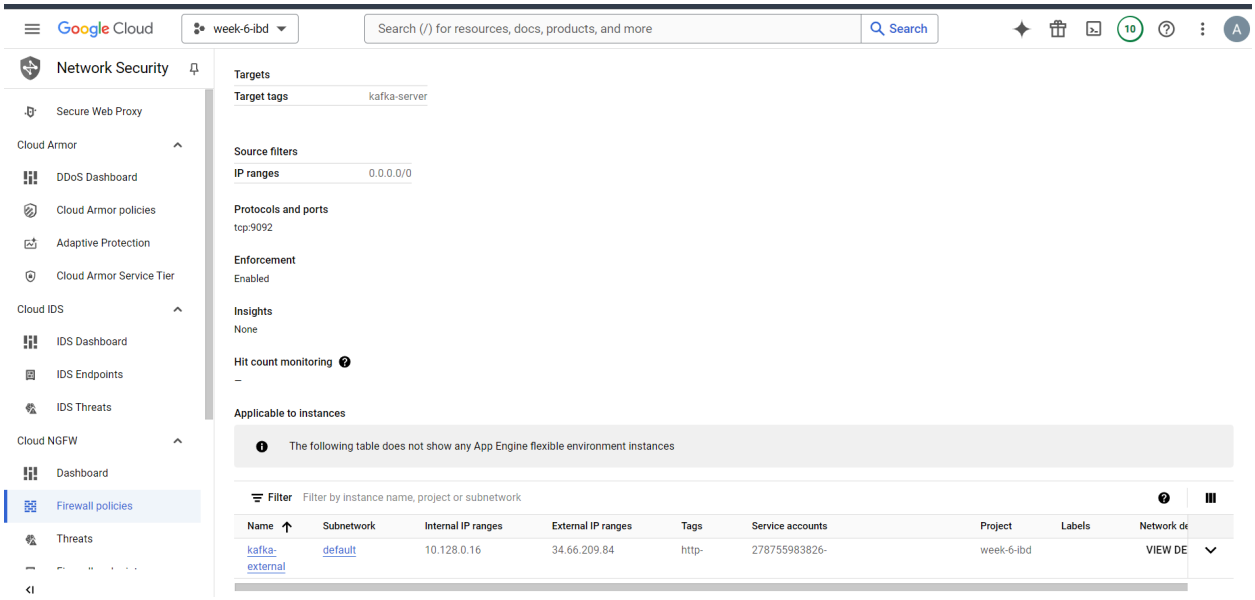
- Kafka requires Zookeeper for coordination.
- Start Zookeeper:
  `kafka/bin/zookeeper-server-start.sh kafka/config/zookeeper.properties &`



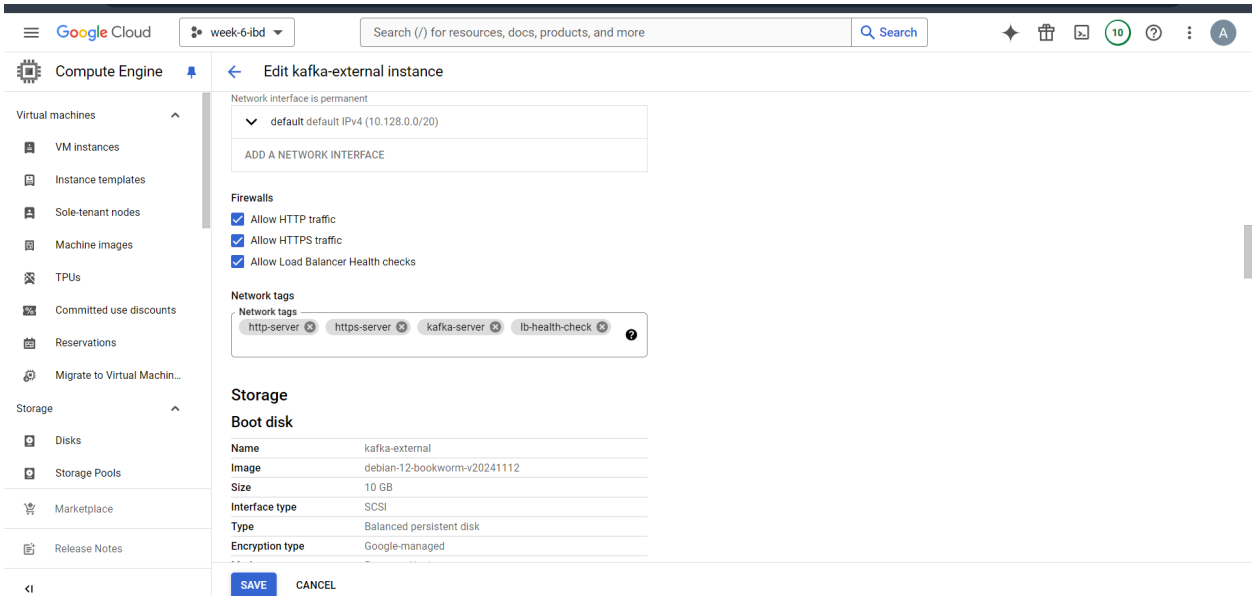
Step 5: Create a New Firewall Rule

- 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: 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 test-topic --bootstrap-server 34.66.209.84:9092`
- List topics:
  `kafka/bin/kafka-topics.sh --list --bootstrap-server 34.66.209.84:9092`

```
avijeetcloud@kafka-external:~/kafka_2.13-3.5.1$ bin/kafka-topics.sh --list --bootstrap-server 34.66.209.84:9092
image-topic
avijeetcloud@kafka-external:~/kafka_2.13-3.5.1$
```

Step 7: Kafka Producer for Image Metadata

- Write a **Kafka producer** script and upload it to the VM ssh.

```python
from kafka import KafkaProducer
from google.cloud import storage
import time

# Initialize Kafka Producer
producer = KafkaProducer(
    bootstrap_servers='34.66.209.84:9092',
    acks='all',  # Wait for acknowledgment from all replicas
    batch_size=1  # Send each record as a separate batch
)

# Function to list and send files from Google Cloud Storage
def list_files(bucket_name):
    client = storage.Client()
    bucket = client.get_bucket(bucket_name)
    blobs = bucket.list_blobs(prefix="images/")  # Adjust prefix if ne
    for blob in blobs:
        # Send each image name as a message to the Kafka topic
        producer.send('image-topic', value=blob.name.encode())
        print(f"Image sent to topic: {blob.name}")
        time.sleep(7)  # Pause between sending images

# Specify GCS Bucket Name
list_files('image-classification-project')
print("Done!")
```

- 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`
- Run the `producer.py`
    - `Python3 producer.py`

Step 8: 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://image-classification-project/install_packages.sh`

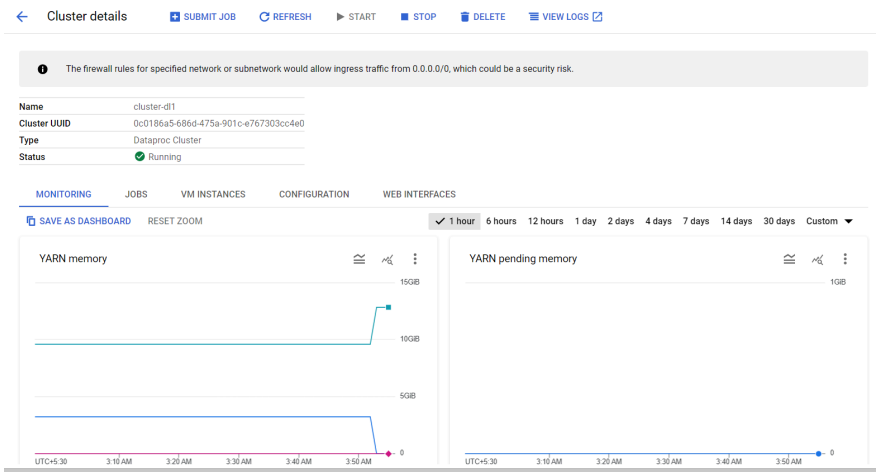
install_packages.sh

```
#!/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
pip3 install torch torchvision tensorflow pillow google-cloud-storage

```



- Check the connection
    - telnet 34.66.209.84 9092

```
avijeetcloud@cluster-dl1-m:~$ telnet 34.66.209.84 9092
Trying 34.66.209.84...
Connected to 34.66.209.84.
Escape character is '^]'.
```

- Write a **Kafka consumer "Stream.py**" script with the modified version on **Notebook given in Week 10 which leverages TensorFlow** and uploads it to the VM ssh.
- Run `the stream.py`
    - `Python3 stream.py`

stream.py > ...

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf
from pyspark.sql.types import StringType
from kafka import KafkaConsumer
import io
import json
from PIL import Image
import torch
import requests
from torchvision import models, transforms
from google.cloud import storage

# Initialize Spark Session
spark = SparkSession.builder \
    .appName("ImageClassificationStreaming") \
    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.1") \
    .config("spark.executorEnv.TORCH_HOME", "/tmp/torch_cache") \
    .getOrCreate()

# Kafka Source
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "34.66.209.84:9092") \
    .option("subscribe", "image-topic") \
    .option("failOnDataLoss", "false") \
    .option("maxOffsetsPerTrigger", "10") \
    .load()

df = df.selectExpr("CAST(value AS STRING) as path")

# ImageNet Labels Loader
Tabnine | Edit | Test | Explain | Document | Ask
def load_imagenet_labels():
    url = "https://raw.githubusercontent.com/anishathalye/imagenet-simple-labels/master/imagenet-simple-labels.json"
    response = requests.get(url)
    return json.loads(response.text)

imagenet_labels = load_imagenet_labels()
```

```
# UDF for Image Classification
Tabnine | Edit | Test | Explain | Document | Ask
def classify_udf(path):
    try:
        client = storage.Client()
        bucket = client.get_bucket('image-classification-project')
        blob = bucket.get_blob(path)
        content = blob.download_as_bytes()

        transform = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ])
        image = Image.open(io.BytesIO(content)).convert("RGB")
        image_tensor = transform(image)

        model = models.mobilenet_v2(pretrained=True)
        model.eval()
        with torch.no_grad():
            predictions = model(image_tensor.unsqueeze(0))
        class_index = predictions.argmax().item()
        return json.dumps({"index": class_index, "label": imagenet_labels[class_index]})
    except Exception as e:
        return json.dumps({"error": str(e)})

# Register UDF
classify_spark_udf = udf(classify_udf, StringType())

# Apply UDF and Write Stream
predictions = df.withColumn("prediction", classify_spark_udf(col("path")))

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

query.awaitTermination()
```

Step 9: Result and Testing the Pipeline

- **Kafka Testing:**
  - Use the Kafka producer script to push test messages containing image paths to the Kafka topic.
- **Upload Test Images:**
  - Add images to the GCS bucket under the appropriate path (e.g., `images/`).
- **Monitor Outputs:**
  - Check the cluster ssh terminal for real-time predictions.
  - Verify the console output for predictions.

SSH-in-browser UPLOAD FILE DOWNLOAD FILE

```
avijeetcloud@kafka-external:~$ source env/bin/activate
(env) avijeetcloud@kafka-external:~$ python3 producer.py
Image sent to topic: images/10172379554_b296050f82_n.jpg
Image sent to topic: images/10486992895_20b344ce2d_n.jpg
Image sent to topic: images/12240577184_b0de0e53ea_n.jpg
Image sent to topic: images/12471290635_1f9e3aae16_n.jpg
Image sent to topic: images/1306119996_ab8ae14d72_n.jpg
Image sent to topic: images/134409839_71069a95d1_m.jpg
Image sent to topic: images/13807932364_673b7f1c1c_n.jpg
Image sent to topic: images/14368895004_c486a29c1e_n.jpg
Image sent to topic: images/1441939151_b271408c8d_n.jpg
Image sent to topic: images/1469726748_f359f4a8c5.jpg
Image sent to topic: images/14698531521_0c2f0c6539.jpg
Image sent to topic: images/14741866338_bdc8bfc8d5_n.jpg
Image sent to topic: images/15319767030_e6c5602a77_m.jpg
Image sent to topic: images/15424480096_45bb574b33.jpg
Image sent to topic: images/16484100863_979beacb08.jpg
Image sent to topic: images/17280886635_e384d91300_n.jpg
Image sent to topic: images/18684594849_7dd3634f5e_n.jpg
Image sent to topic: images/1879567877_8ed2a5faa7_n.jpg
Image sent to topic: images/19813618946_93818db7aa_m.jpg
```

```
-------------------------------------------
+--------------------------------+--------------------------------+
|path                            |prediction                      |
+--------------------------------+--------------------------------+
|images/16484100863_979beacb08.jpg|{"index": 562, "label": "fountain"}|
+--------------------------------+--------------------------------+

-------------------------------------------
Batch: 9
-------------------------------------------
+--------------------------------+--------------------------------+
|path                             |prediction                      |
+--------------------------------+--------------------------------+
|images/17280886635_e384d91300_n.jpg|{"index": 228, "label": "Komondor"}|
|images/18684594849_7dd3634f5e_n.jpg|{"index": 985, "label": "daisy"}   |
+--------------------------------+--------------------------------+

-------------------------------------------
Batch: 10
-------------------------------------------
+--------------------------------+--------------------------------+
|path                            |prediction                      |
+--------------------------------+--------------------------------+
|images/1879567877_8ed2a5faa7_n.jpg|{"index": 985, "label": "daisy"}|
+--------------------------------+--------------------------------+

-------------------------------------------
Batch: 11
-------------------------------------------
+--------------------------------+--------------------------------+
|path                             |prediction                      |
+--------------------------------+--------------------------------+
|images/19813618946_93818db7aa_m.jpg|{"index": 985, "label": "daisy"}|
|images/2019064575_7656b9340f_m.jpg |{"index": 985, "label": "daisy"}|
+--------------------------------+--------------------------------+

-------------------------------------------
Batch: 12
-------------------------------------------
+--------------------------------+--------------------------------------------------+
|path                            |prediction                                        |
+--------------------------------+--------------------------------------------------+
|images/2045022175_ad087f5f60_n.jpg|{"index": 89, "label": "sulphur-crested cockatoo"}|
+--------------------------------+--------------------------------------------------+
```

Conclusion

This project successfully transformed a batch image classification use case into a real-time streaming model using Spark Streaming and Kafka on GCP. By setting up GCS, VMs, and Dataproc clusters, we created a real-time image processing pipeline that streams image metadata and provides predictions using TensorFlow. Despite challenges in configuring Kafka and firewall rules, the pipeline was successfully tested, producing real-time predictions for uploaded images. This solution demonstrates the potential of cloud computing and streaming technologies for scalable real-time analytics.