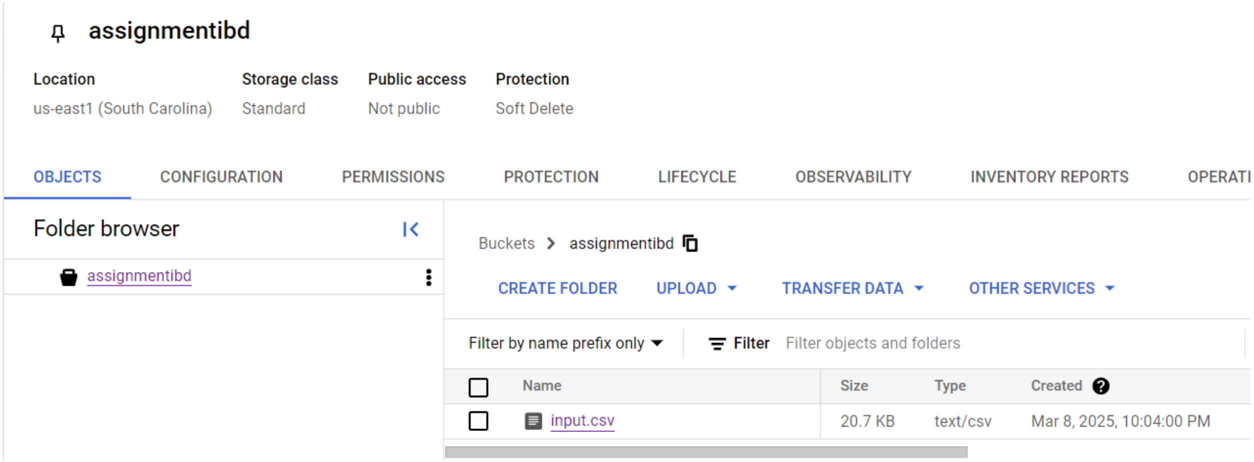
Question: Write a Producer that reads from that file, breaks the data into, and writes to Kafka and a Spark Streaming consumer that reads from the topic and emits count of rows.

Submitted by Shramana Sinha, 23f1002703

1. Implementation Steps

1.1 Google Cloud Storage Configuration

- Created a bucket named `assignmentibd`
- Uploaded an input file `input.csv` containing customer data with 1,000 records



assignmentibd

Location	Storage class	Public access	Protection
us-east1 (South Carolina)	Standard	Not public	Soft Delete

Name	Size	Type	Created
□ input.csv	20.7 KB	text/csv	Mar 8, 2025, 10:04:00 PM

Screenshot: The gcs bucket with the input.csv file

1.2 Network Configuration

- Created a firewall rule to allow Kafka traffic on port 9092:

```
Unset
gcloud compute firewall-rules create allow-kafka --allow tcp:9092
--description="Allow Kafka traffic" --direction=INGRESS
```

1.3 Kafka Server Setup

- Launched a VM instance with Ubuntu 20.04 LTS
- Installed Java and Python dependencies:

Unset

```
sudo apt update && sudo apt install -y openjdk-8-jdk wget python3-pip
pip install kafka-python google-cloud-storage
```

- Downloaded and extracted Apache Kafka 3.9.0:

Unset

```
wget https://dlcdn.apache.org/kafka/3.9.0/kafka_2.13-3.9.0.tgz
tar -xzf kafka_2.13-3.9.0.tgz
```

- Configured Kafka server to accept external connections:

Unset

```
# In kafka_2.13-3.9.0/config/server.properties
listeners=PLAINTEXT://0.0.0.0:9092
advertised.listeners=PLAINTEXT://<VM_EXTERNAL_IP>:9092
```

- Started Zookeeper and Kafka server as daemon processes:

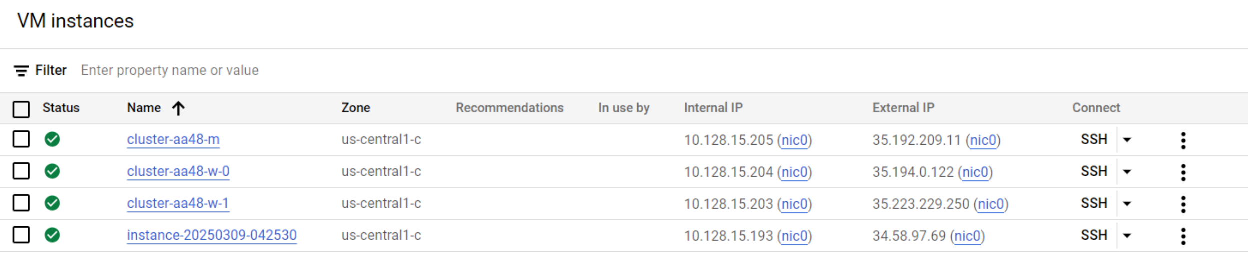
Unset

```
kafka_2.13-3.9.0/bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
kafka_2.13-3.9.0/bin/kafka-server-start.sh -daemon config/server.properties
```

- Uploaded the `producer.py` script and started its execution:

Unset

```
python3 producer.py
```



VM instances

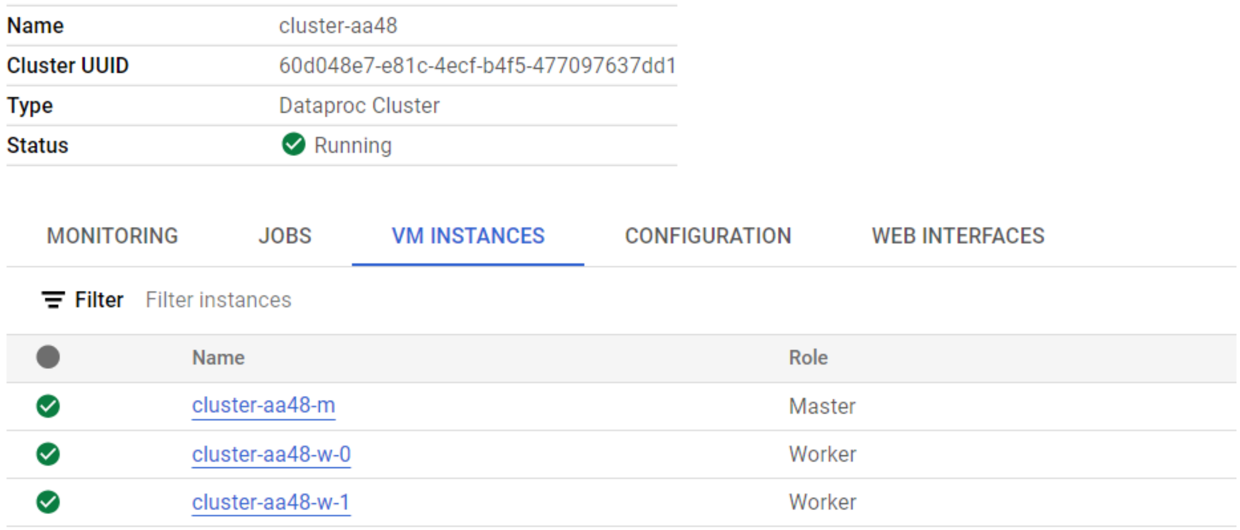
□	Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
□	✓	cluster-aa48-m	us-central1-c			10.128.15.205 (nic0)	35.192.209.11 (nic0)	SSH ▾ ⋮
□	✓	cluster-aa48-w-0	us-central1-c			10.128.15.204 (nic0)	35.194.0.122 (nic0)	SSH ▾ ⋮
□	✓	cluster-aa48-w-1	us-central1-c			10.128.15.203 (nic0)	35.223.229.250 (nic0)	SSH ▾ ⋮
□	✓	instance-20250309-042530	us-central1-c			10.128.15.193 (nic0)	34.58.97.69 (nic0)	SSH ▾ ⋮

Screenshot: The vm (instance-20250309-042530) for running the kafka and producer.py

1.4 Dataproc Cluster Setup

- Created a Dataproc cluster with external IP access enabled
- Uploaded the `consumer.py` script.
- Submitted the spark job

```
Unset
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.4
consumer.py
```



Name	cluster-aa48
Cluster UUID	60d048e7-e81c-4ecf-b4f5-477097637dd1
Type	Dataproc Cluster
Status	✓ Running

●	Name	Role
✓	cluster-aa48-m	Master
✓	cluster-aa48-w-0	Worker
✓	cluster-aa48-w-1	Worker

Screenshot: The dataproc cluster for running the consumer.py

2. Data Producer Implementation

The producer application performs the following tasks:

- Downloads the CSV file from Google Cloud Storage

```python
Python
def download_from_gcs(bucket_name, source_blob_name, destination_file_name):
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(source_blob_name)
    blob.download_to_filename(destination_file_name)
    print(
        f"Downloaded gs://{bucket_name}/{source_blob_name} to
{destination_file_name}"
    )
```

- Reads and parses the customer data

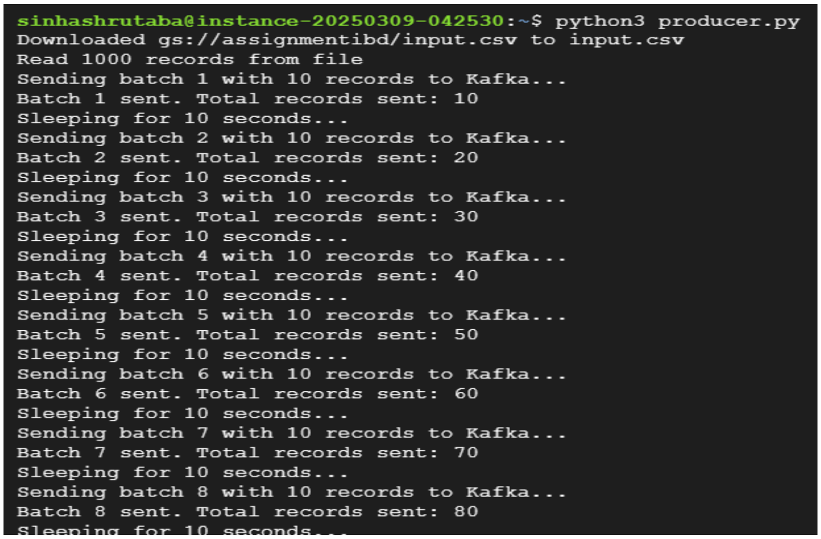
Python
```python
def read_local_file(file_path):
    rows = []
    with open(file_path, "r") as file:
        csv_reader = csv.DictReader(file, fieldnames=["customer_id", "name", "city"])
        for row in csv_reader:
            rows.append(row)
    return rows
```

- Streams the data to Kafka in batches of 10 records

Python
```python
records_to_send = len(batch)
batch = batch[:records_to_send]
print(f"Sending batch {i+1} with {len(batch)} records to Kafka...")
for record in batch:
       producer.send(KAFKA_TOPIC, value=record)
       sent_count += 1
producer.flush()
```

- Implements a controlled flow with 10-second pauses between batches

Python
```python
if i < len(batches) - 1:
       print(f"Sleeping for {sleep_seconds} seconds...")
       time.sleep(sleep_seconds)
```



```
sinhashrutaba@instance-20250309-042530:~$ python3 producer.py
Downloaded gs://assignmentibd/input.csv to input.csv
Read 1000 records from file
Sending batch 1 with 10 records to Kafka...
Batch 1 sent. Total records sent: 10
Sleeping for 10 seconds...
Sending batch 2 with 10 records to Kafka...
Batch 2 sent. Total records sent: 20
Sleeping for 10 seconds...
Sending batch 3 with 10 records to Kafka...
Batch 3 sent. Total records sent: 30
Sleeping for 10 seconds...
Sending batch 4 with 10 records to Kafka...
Batch 4 sent. Total records sent: 40
Sleeping for 10 seconds...
Sending batch 5 with 10 records to Kafka...
Batch 5 sent. Total records sent: 50
Sleeping for 10 seconds...
Sending batch 6 with 10 records to Kafka...
Batch 6 sent. Total records sent: 60
Sleeping for 10 seconds...
Sending batch 7 with 10 records to Kafka...
Batch 7 sent. Total records sent: 70
Sleeping for 10 seconds...
Sending batch 8 with 10 records to Kafka...
Batch 8 sent. Total records sent: 80
Sleeping for 10 seconds...
```

Screenshot: The producer sending batches of 10 records to kafka every 10 seconds

3. Data Consumer Implementation

The Spark Streaming consumer application performs the following functions:

- Establishes a connection to the Kafka topic and creates a streaming DataFrame from the Kafka source

```python
df = (
        spark.readStream.format("kafka")
        .option("kafka.bootstrap.servers", KAFKA_BOOTSTRAP_SERVERS)
        .option("subscribe", KAFKA_TOPIC)
        .option("startingOffsets", "earliest")
        .load()
    )
```

- Processes the incoming data by casting the binary values to string and getting the timestamp from the message.

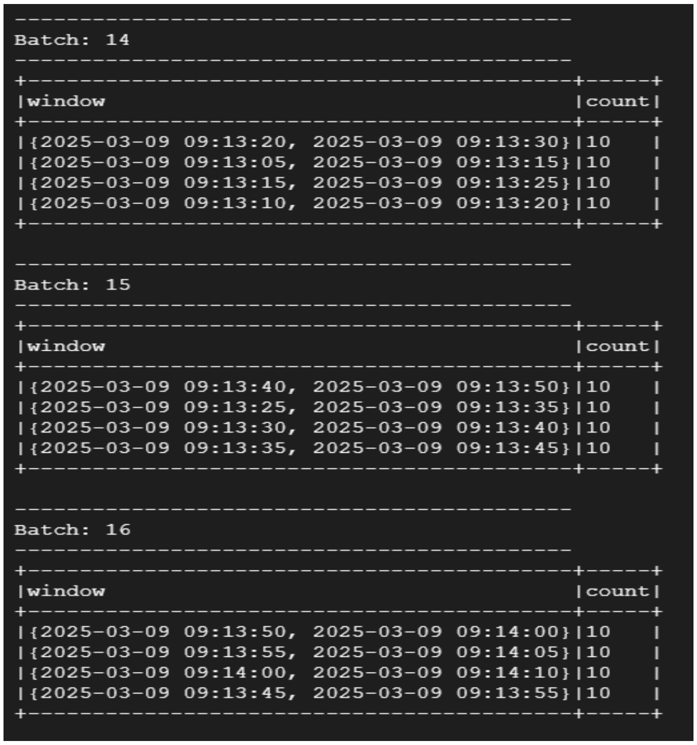
```python
value_df = df.selectExpr("CAST(value AS STRING)", "timestamp")
```

- Counts records in 10-second windows with 5-second sliding intervals

```python
windowedCounts = value_df.groupBy(
        window(col("timestamp"), "10 seconds", "5 seconds")
    ).count()
```

- Outputs results to the console in update mode

```python
query = (
        windowedCounts.writeStream.outputMode("update")
        .format("console")
        .option("truncate", "false")
        .start()
    )
```



```
-------------------------------------------
Batch: 14
-------------------------------------------
+------------------------------------------+-----+
|window                                    |count|
+------------------------------------------+-----+
|{2025-03-09 09:13:20, 2025-03-09 09:13:30}|10   |
|{2025-03-09 09:13:05, 2025-03-09 09:13:15}|10   |
|{2025-03-09 09:13:15, 2025-03-09 09:13:25}|10   |
|{2025-03-09 09:13:10, 2025-03-09 09:13:20}|10   |
+------------------------------------------+-----+

-------------------------------------------
Batch: 15
-------------------------------------------
+------------------------------------------+-----+
|window                                    |count|
+------------------------------------------+-----+
|{2025-03-09 09:13:40, 2025-03-09 09:13:50}|10   |
|{2025-03-09 09:13:25, 2025-03-09 09:13:35}|10   |
|{2025-03-09 09:13:30, 2025-03-09 09:13:40}|10   |
|{2025-03-09 09:13:35, 2025-03-09 09:13:45}|10   |
+------------------------------------------+-----+

-------------------------------------------
Batch: 16
-------------------------------------------
+------------------------------------------+-----+
|window                                    |count|
+------------------------------------------+-----+
|{2025-03-09 09:13:50, 2025-03-09 09:14:00}|10   |
|{2025-03-09 09:13:55, 2025-03-09 09:14:05}|10   |
|{2025-03-09 09:14:00, 2025-03-09 09:14:10}|10   |
|{2025-03-09 09:13:45, 2025-03-09 09:13:55}|10   |
+------------------------------------------+-----+
```

Screenshot: The output of the consumer.py showing the count of records for 10 seconds windows which are sliding every 5 seconds