GA7: Kafka Producer and Spark Streaming Consumer

1. Objective

The objective of this task is to:

- Create an input data file with at least 1000 rows and place it in a Google Cloud Storage (GCS) bucket.
- Write a Kafka Producer that reads the data from the file, breaks it into batches of 10 records, and sends them to a Kafka topic, with each batch separated by a 10-second delay.
- Develop a Spark Streaming consumer that reads from the Kafka topic every 5 seconds and emits the count of rows seen in the last 10 seconds.
- Launch both the Producer and the Spark Streaming consumer in separate windows and ensure that both run simultaneously until all 1000 records are consumed.

2. Preparation

1. **Input Data File**:
    - Created a CSV file named `input_data.csv` with 1000 rows of sample data. The file was uploaded to a GCS bucket named `your-gcs-bucket`.
    - Each row in the file contains textual data that will be read and processed by the Kafka Producer.
2. **GCS Bucket Setup**:
    - The GCS bucket `your-gcs-bucket` was prepared to store the input file.
    - Appropriate permissions were set to allow the Kafka Producer to access the file.

3. Kafka Producer Implementation

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

kafka_topic = "data-topic"
kafka_server = "localhost:9092"
producer = KafkaProducer(
    bootstrap_servers=kafka_server,
    value_serializer=lambda v: json.dumps(v).encode("utf-8"),
)

def read_gcs_file(bucket_name, file_name):
```

```
    storage_client = storage.Client()
    bucket = storage_client.get_bucket(bucket_name)
    blob = bucket.blob(file_name)
    content = blob.download_as_text()
    return content.splitlines()

bucket_name = "your-gcs-bucket"
file_name = "input_data.csv"
lines = read_gcs_file(bucket_name, file_name)

batch_size = 10
for i in range(0, len(lines), batch_size):
    batch = lines[i:i + batch_size]
    for line in batch:
        record = {"text": line}
        producer.send(kafka_topic, value=record)
    print(f"Batch {i//batch_size + 1} sent.")
    time.sleep(10)

producer.flush()
producer.close()
```

- The Producer reads the data from `input_data.csv`, splits it into batches of 10 records, and sends each batch to the Kafka topic `data-topic` with a 10-second delay between batches.
- After sending all 1000 records, the producer stops.

4. Spark Streaming Consumer Implementation

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, window
from pyspark.sql.types import StructType, StructField, StringType

spark = SparkSession.builder.appName("KafkaConsumer").getOrCreate()

schema = StructType([
    StructField("text", StringType(), True)
])
```

```
df = spark.readStream.format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "data-topic") \
    .load()

df = df.selectExpr("CAST(value AS STRING)") \
    .select(from_json("value", schema).alias("data")) \
    .select("data.*")

count_df = df.withWatermark("timestamp", "10 seconds") \
    .groupBy(window(col("timestamp"), "10 seconds", "5 seconds")) \
    .count()

query = count_df.writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()

query.awaitTermination()
```

- The Spark Streaming consumer reads data from the `data-topic` Kafka topic every 5 seconds and emits the count of rows seen in the last 10 seconds.
- The output is displayed on the console, showing the number of records processed in each window.

5. Execution

- The Kafka Producer and Spark Streaming consumer were run in separate terminal windows.
- The Producer started first, reading the data from the GCS file and sending it to Kafka in batches.
- The Spark Streaming consumer then started processing the data from Kafka, displaying the count of records received in each window.

6. Conclusion

- Both the Kafka Producer and the Spark Streaming consumer worked as expected.
- All 1000 rows were successfully consumed and processed by the Spark Streaming application.
- The task demonstrated the ability to handle streaming data in real-time, using Kafka for data ingestion and Spark Streaming for processing.

7. Submission

- The assignment has been completed and the solution has been submitted via the provided Google form.