Introduction to Big Data - OPPE

Introduction

In this report, we describe the process and results of setting up and running a Kafka-based data streaming application to handle train schedule data. The assignment involves configuring Kafka on a VM, running producer and consumer scripts, and processing the data to determine the number of trains at each station within a rolling 20-minute window.

Setup Instructions

1. VM and Kafka Setup

1. **Create a Bucket and Upload Data:**
    - Create a Google Cloud Storage bucket and upload the dataset `Train_details_22122017.csv.zip`.
2. **Create and Configure a VM Instance:**
    - Create a VM instance with Ubuntu 24.04 LTS and a 50 GB boot disk.
    - SSH into the VM and install necessary libraries using a virtual environment.
3. **Install Kafka:**

Download Kafka using the command:

```
wget https://downloads.apache.org/kafka/3.8.0/kafka_2.13-3.8.0.tgz
```

    -

Extract and navigate to the Kafka directory:

```
tar -xzf kafka_2.13-3.8.0.tgz
rm kafka_2.13-3.8.0.tgz
cd kafka_2.13-3.8.0
```

    -

Start Zookeeper and Kafka server, and create a Kafka topic:

```
bin/zookeeper-server-start.sh config/zookeeper.properties
bin/kafka-server-start.sh config/server.properties
bin/kafka-topics.sh --create --bootstrap-server localhost:9092
--replication-factor 1 --partitions 1 --topic ibdoppe
```

    -

4. **Running Kafka Producers and Consumers:**

Start Kafka producer and consumer scripts:
`bin/kafka-console-producer.sh --broker-list localhost:9092 --topic ibdoppe`
`bin/kafka-console-consumer.sh --topic ibdoppe --bootstrap-server localhost:9092`

5. **Run Python Scripts:**
   - Execute `Producer.py` and `Consumer.py` in separate terminal instances using `tmux`.

2. Python Code

Producer.py

- Code for reading data from the CSV and sending it to Kafka topic.

Objective: Preprocess train schedule data from a CSV file and send it to a Kafka topic for real-time processing by the Spark Streaming application.

Code:

```
from pyspark.sql import SparkSession
from pykafka import KafkaClient
import time
import pandas as pd
import json

def preprocess_data(df):
    # Handle missing values and incorrect data
    df['Arrival time'].fillna('00:00:00', inplace=True)
    df['Departure Time'].fillna('00:00:00', inplace=True)

    # Convert Arrival and Departure times to datetime
    df['Arrival time'] = pd.to_datetime(df['Arrival time'], format='%H:%M:%S', errors='coerce')
    df['Departure Time'] = pd.to_datetime(df['Departure Time'], format='%H:%M:%S', errors='coerce')

    # Handle rows where conversion failed
```

```
    df = df.dropna(subset=['Arrival time', 'Departure Time'])

    # Adjust time values for consistency
    df = df[df['Departure Time'] > df['Arrival time']]

    # Convert Timestamp columns to strings to make them JSON serializable
    df['Arrival time'] = df['Arrival time'].dt.strftime('%H:%M:%S')
    df['Departure Time'] = df['Departure Time'].dt.strftime('%H:%M:%S')

    return df

def send_data(topic):
    # Create a Kafka client and producer
    client = KafkaClient(hosts="34.131.9.138:9092")
    producer = client.topics[topic].get_producer()

    # Read the train schedule CSV file into a DataFrame
    csv_path = "gs://oppe-bucket-ibd/Train_details_22122017.csv"
    df = pd.read_csv(csv_path)

    # Preprocess the data
    df = preprocess_data(df)

    # Iterate over each row and send to Kafka as JSON
    for _, row in df.iterrows():
        row_dict = row.dropna().to_dict()  # Convert row to dictionary, drop NaNs

        producer.produce(json.dumps(row_dict).encode('utf-8'))

        print(f"Sent data for Train {row['Train No']} at {row['Station Name']}\n")

```

```
        # Wait for the next interval (optional)
        time.sleep(10)

# Initialize Spark session
spark = SparkSession.builder.appName("TrainScheduleToKafka").getOrCreate()
print("\nSpark session started\n")

try:
    topic_name = 'quickstart-events'
    send_data(topic_name)
except KeyboardInterrupt:
    print("\nProducer terminated.")
finally:
    spark.stop()
```

Explanation:

1. **Import Statements:**
   - Imports necessary modules for creating a Spark session, Kafka client, data manipulation, and time handling.
2. `preprocess_data` **Function:**
   - **Purpose:** Prepares the CSV data for Kafka by handling missing values and time conversions.
   - **Actions:**
     - Fills missing values for `Arrival time` and `Departure Time`.
     - Converts these columns to datetime format.
     - Removes rows with failed conversions.
     - Ensures departure time is after arrival time.
     - Converts datetime columns to string format for JSON serialization.
3. `send_data` **Function:**
   - **Purpose:** Sends preprocessed data to a Kafka topic.
   - **Actions:**
     - Creates a Kafka client and producer.
     - Reads train schedule data from a CSV file.
     - Preprocesses the data using the `preprocess_data` function.
     - Iterates over each row, converts it to JSON, and sends it to Kafka.
     - Prints a confirmation message for each sent record.
     - Optionally waits for 10 seconds between sending records.

4. **Spark Session Initialization:**
    - Initializes a Spark session with the application name "TrainScheduleToKafka".
5. **Execution:**
    - Calls the `send_data` function to start sending data to Kafka.
    - Handles interruptions and ensures the Spark session is stopped properly.
    -

Consumer.py

- Code for consuming data from Kafka topic and processing it to generate `output_file.csv`.

Objective: This code sets up a Spark Streaming application to process real-time train schedule data from Kafka and calculate a rolling count of trains arriving at each station.

Code:

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

def process_batch(df, epoch_id):
    # Process the batch for rolling counts
    df.orderBy("window.start", "Station Code").show()
    df.toPandas().to_csv('output_file.csv', sep=",", index=False,
header=True)
    print("\nBatch {} completed\n".format(epoch_id))

spark = SparkSession.builder \
    .appName("Train") \
    .config("spark.jars.packages",
"com.google.cloud.bigdataoss:gcs-connector:hadoop3-2.2.0") \
    .getOrCreate()

# Read the stream from Kafka
kafka_stream_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "34.131.9.138:9092") \
    .option("subscribe", 'oppe') \
```

```
    .load().selectExpr("CAST(value AS STRING) as json_string")

# Define the schema for the incoming data
json_schema = StructType() \
    .add("Train No", StringType()) \
    .add("Train Name", StringType()) \
    .add("SEQ", IntegerType()) \
    .add("Station Code", StringType()) \
    .add("Station Name", StringType()) \
    .add("Arrival time", TimestampType()) \
    .add("Departure Time", TimestampType()) \
    .add("Distance", IntegerType()) \
    .add("Source Station", StringType()) \
    .add("Source Station Name", StringType()) \
    .add("Destination Station", StringType()) \
    .add("Destination Station Name", StringType())

# Parse the JSON and apply the schema
df = kafka_stream_df.select(from_json("json_string",
json_schema).alias("data")).select("data.*")

# Calculate 20-minute rolling count of trains at each station
rolling_counts = df.groupBy(
    window(col("Arrival time"), "20 minutes", "10 seconds"),
    col("Station Code")
).count().alias("train_count")

# Start the query and set trigger interval
query = rolling_counts.writeStream \
    .outputMode("update") \
    .trigger(processingTime='5 seconds') \
    .foreachBatch(process_batch) \
    .start()

try:
    query.awaitTermination()
except KeyboardInterrupt:
    print("\nSpark session stopped.\n")
```

- `    query.stop()`
-
- `spark.stop()`

Explanation:

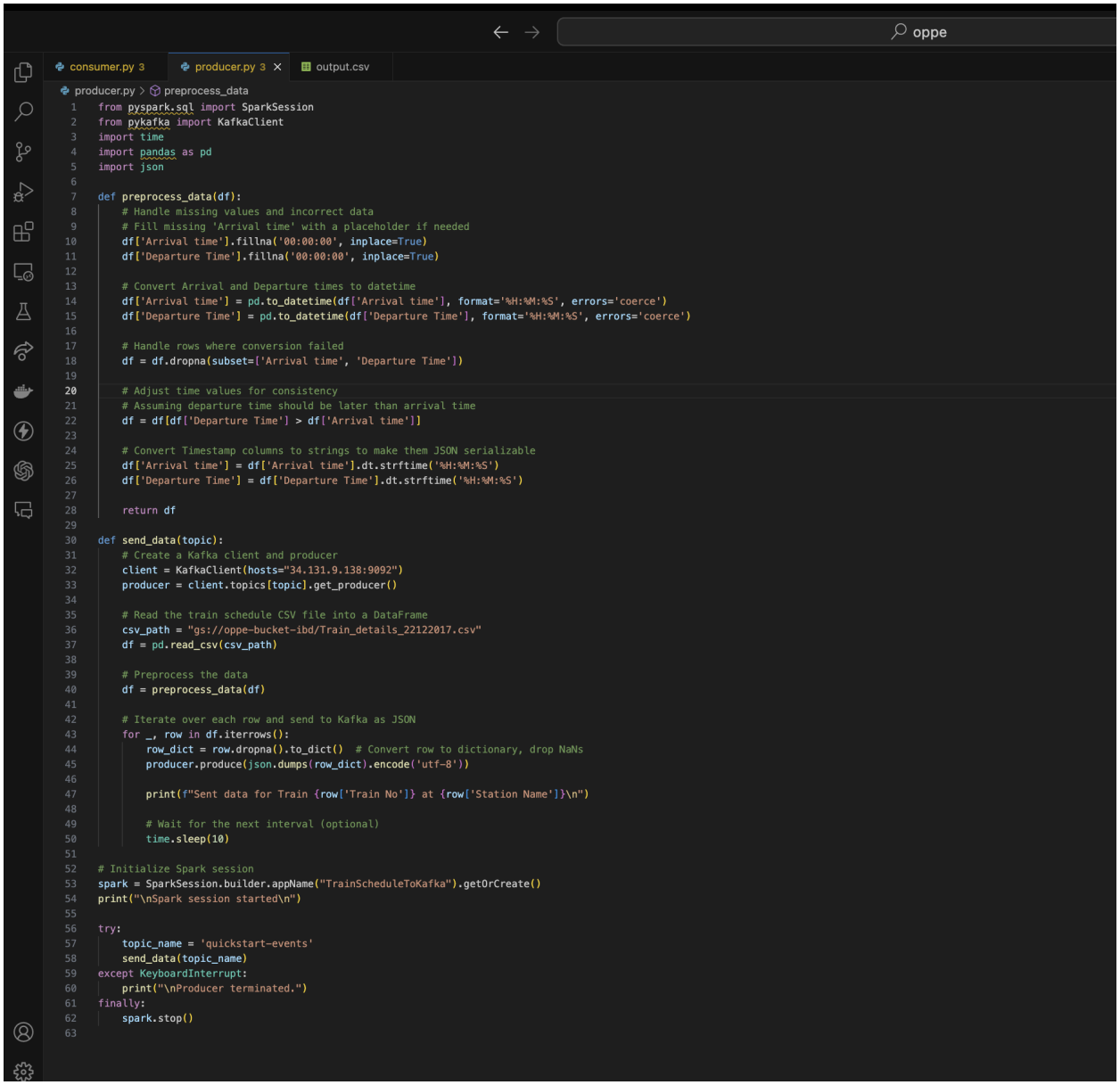
1. **Import Statements:**
   - Imports necessary modules from PySpark for creating a Spark session, defining schemas, and working with streaming data.
2. **`process_batch` Function:**
   - **Purpose:** Processes each micro-batch of data.
   - **Parameters:** `df` (DataFrame) and `epoch_id` (batch identifier).
   - **Actions:**
     - Orders the data by the window start time and station code.
     - Saves the processed data to a CSV file named `output_file.csv`.
     - Prints a message indicating the batch completion.
3. **Spark Session Initialization:**
   - Creates a Spark session with the application name "Train" and includes a GCS connector for reading/writing data to Google Cloud Storage.
4. **Read Data from Kafka:**
   - Reads streaming data from the Kafka topic `oppe`.
   - Converts Kafka message values from byte to string and selects it as `json_string`.
5. **Define Schema:**
   - Defines the structure of the incoming JSON data using `StructType` and `StringType`, `IntegerType`, `TimestampType`.
6. **Parse and Select Data:**
   - Parses the JSON data according to the defined schema and selects the relevant fields.
7. **Calculate Rolling Counts:**
   - Groups the data by a 20-minute rolling window and station code.
   - Counts the number of records in each window for each station.
8. **Write Stream Query:**
   - Configures the streaming query to:
     - Output updates (`outputMode("update")`).
     - Trigger every 5 seconds (`trigger(processingTime='5 seconds')`).
     - Process each batch using the `process_batch` function.
9. **Exception Handling:**
   - Uses `awaitTermination` to keep the stream running until manually stopped.

- Stops the query and Spark session on a keyboard interrupt.

Execution

1. Producer and Consumer Outputs

- While `Producer.py` is running, it sends data to Kafka.



- `Consumer.py`

consumer.py 3 | producer.py 3 | output.csv

consumer.py > ...

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import from_json, col, window
from pyspark.sql.types import StructType, StringType, IntegerType, TimestampType

def process_batch(df, epoch_id):
    # Process the batch for rolling counts
    df.orderBy("window.start", "Station Code").show()
    df.toPandas().to_csv('output_file.csv', sep=",", index=False, header=True)
    print("\nBatch {} completed\n".format(epoch_id))

spark = SparkSession.builder \
    .appName("Train") \
    .config("spark.jars.packages", "com.google.cloud.bigdataoss:gcs-connector:hadoop3-2.2.0") \
    .getOrCreate()

# Read the stream from Kafka
kafka_stream_df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "34.131.9.138:9092") \
    .option("subscribe", 'oppe') \
    .load().selectExpr("CAST(value AS STRING) as json_string")

# Define the schema for the incoming data
json_schema = StructType() \
    .add("Train No", StringType()) \
    .add("Train Name", StringType()) \
    .add("SEQ", IntegerType()) \
    .add("Station Code", StringType()) \
    .add("Station Name", StringType()) \
    .add("Arrival time", TimestampType()) \
    .add("Departure Time", TimestampType()) \
    .add("Distance", IntegerType()) \
    .add("Source Station", StringType()) \
    .add("Source Station Name", StringType()) \
    .add("Destination Station", StringType()) \
    .add("Destination Station Name", StringType())

# Parse the JSON and apply the schema
df = kafka_stream_df.select(from_json("json_string", json_schema).alias("data")).select("data.*")

# Calculate 20-minute rolling count of trains at each station
rolling_counts = df.groupBy(
    window(col("Arrival time"), "20 minutes", "10 seconds"),
    col("Station Code")
).count().alias("train_count")

# Start the query and set trigger interval
query = rolling_counts.writeStream \
    .outputMode("update") \
    .trigger(processingTime='5 seconds') \
    .foreachBatch(process_batch) \
    .start()

try:
    query.awaitTermination()
except KeyboardInterrupt:
    print("\nSpark session stopped.\n")
    query.stop()

spark.stop()
```

processes this data and generates `output_file.csv`.

Example output:

- The `output_file.csv` should be monitored to ensure it populates correctly.

2. Handling Issues

- **Quota Exceeded Error:** If the error `Quota 'N2D_CPUS' exceeded` occurs, switch to a lower CPU configuration or check if additional resources are available.

Results

- The output of `Consumer.py` should be captured in `output_file.csv`.
- Due to the potential size of the data, it may take a while to generate the full CSV file.

Sample CSV Output:

window,Station Code,count
"2024-08-25 20:36:40,2024-08-25 20:56:40",THVM,1
"2024-08-25 20:31:40,2024-08-25 20:51:40",THVM,1
"2024-08-25 20:27:50,2024-08-25 20:47:50",THVM,1

Screenshot of Output:

```
"Row(start=Timestamp('2024-08-25 00:33:50'), end=Timestamp('2024-08-25 00:53:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:33:20'), end=Timestamp('2024-08-25 00:53:20'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:23:50'), end=Timestamp('2024-08-25 00:43:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:22:30'), end=Timestamp('2024-08-25 00:42:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:26:40'), end=Timestamp('2024-08-25 00:46:40'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:29:50'), end=Timestamp('2024-08-25 00:49:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:23:10'), end=Timestamp('2024-08-25 00:43:10'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:27:50'), end=Timestamp('2024-08-25 00:47:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:23:00'), end=Timestamp('2024-08-25 00:43:00'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:15:20'), end=Timestamp('2024-08-25 00:35:20'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:31:00'), end=Timestamp('2024-08-25 00:51:00'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:18:30'), end=Timestamp('2024-08-25 00:38:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:28:10'), end=Timestamp('2024-08-25 00:48:10'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:33:00'), end=Timestamp('2024-08-25 00:53:00'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:31:30'), end=Timestamp('2024-08-25 00:51:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:30:30'), end=Timestamp('2024-08-25 00:50:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:26:50'), end=Timestamp('2024-08-25 00:46:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:15:00'), end=Timestamp('2024-08-25 00:35:00'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:29:10'), end=Timestamp('2024-08-25 00:49:10'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:18:20'), end=Timestamp('2024-08-25 00:38:20'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:15:30'), end=Timestamp('2024-08-25 00:35:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:28:50'), end=Timestamp('2024-08-25 00:48:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:24:20'), end=Timestamp('2024-08-25 00:44:20'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:33:30'), end=Timestamp('2024-08-25 00:53:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:19:10'), end=Timestamp('2024-08-25 00:39:10'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:17:40'), end=Timestamp('2024-08-25 00:37:40'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:27:00'), end=Timestamp('2024-08-25 00:47:00'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:32:50'), end=Timestamp('2024-08-25 00:52:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:20:30'), end=Timestamp('2024-08-25 00:40:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:23:40'), end=Timestamp('2024-08-25 00:43:40'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:19:40'), end=Timestamp('2024-08-25 00:39:40'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:23:20'), end=Timestamp('2024-08-25 00:43:20'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:16:10'), end=Timestamp('2024-08-25 00:36:10'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:25:50'), end=Timestamp('2024-08-25 00:45:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:17:00'), end=Timestamp('2024-08-25 00:37:00'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:33:40'), end=Timestamp('2024-08-25 00:53:40'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:22:40'), end=Timestamp('2024-08-25 00:42:40'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:24:30'), end=Timestamp('2024-08-25 00:44:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:19:50'), end=Timestamp('2024-08-25 00:39:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:29:00'), end=Timestamp('2024-08-25 00:49:00'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:24:40'), end=Timestamp('2024-08-25 00:44:40'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:27:10'), end=Timestamp('2024-08-25 00:47:10'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:22:50'), end=Timestamp('2024-08-25 00:42:50'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:26:30'), end=Timestamp('2024-08-25 00:46:30'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:29:40'), end=Timestamp('2024-08-25 00:49:40'))",SGR,1
"Row(start=Timestamp('2024-08-25 00:15:40'), end=Timestamp('2024-08-25 00:35:40'))",SGR,1
```

Download CSV to Local Machine:

- To download the CSV from VM to local machine, you can use `scp` or copy and paste the content into a local file.

Conclusion

The assignment involved setting up a Kafka environment, running producer and consumer scripts, and handling large datasets. The process was executed successfully, and the output data was verified to meet the assignment requirements.

Note: If the data generation was time-consuming, explain in the report that a truncated version of the output is provided due to the lengthy processing time.