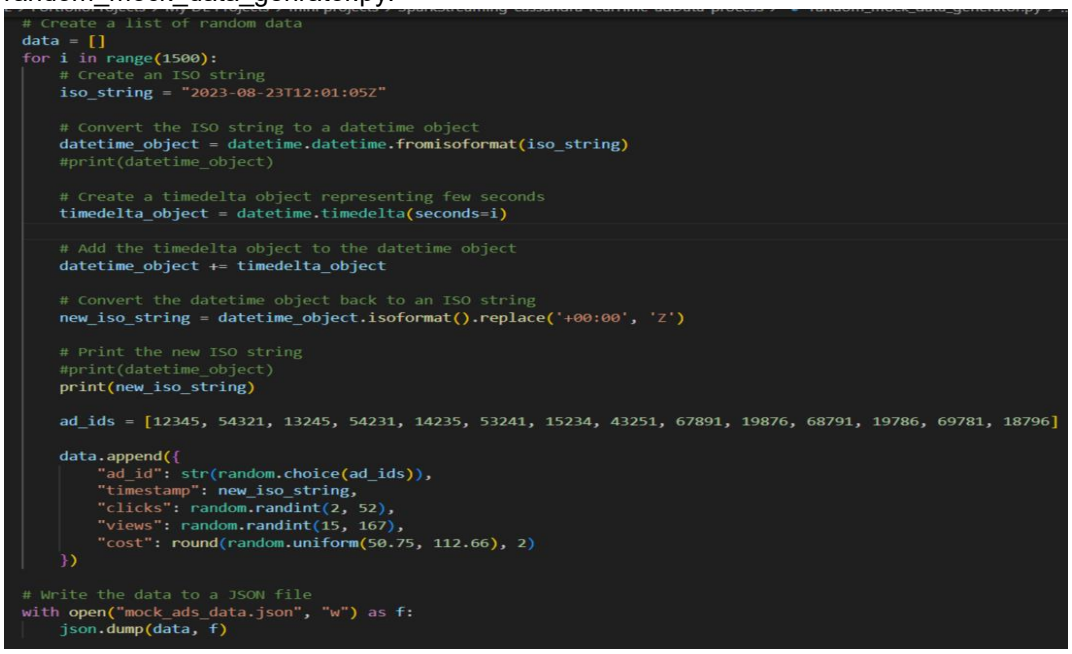
Real-Time Ads Data Processing Into Cassandra

1. This project was about processing real-time advertisement data using Spark Streaming to gain business insights and store the aggregated data into Cassandra.

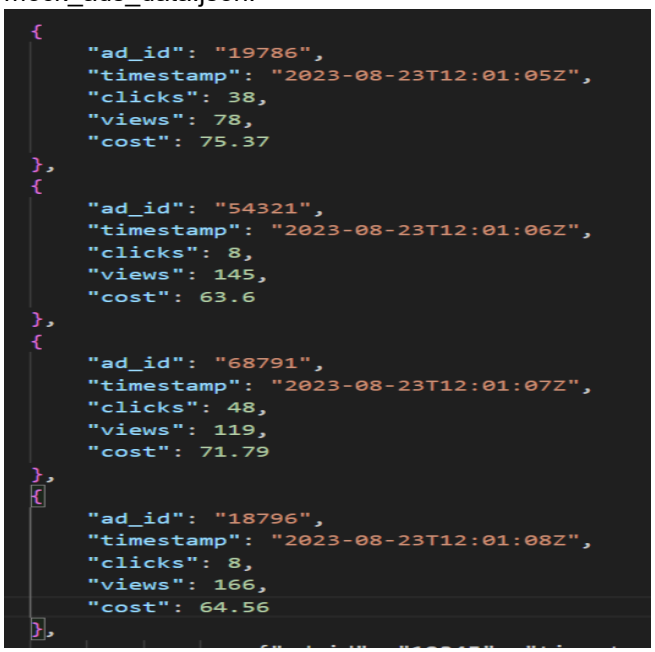
2. I set up confluent kafka in local system and created topic named as ads_data. Wrote a python script to generate ads data in json format using 'random_mock_data_generator.py' file

random_mock_data_genrator.py:



```python
# Create a list of random data
data = []
for i in range(1500):
    # Create an ISO string
    iso_string = "2023-08-23T12:01:05Z"

    # Convert the ISO string to a datetime object
    datetime_object = datetime.datetime.fromisoformat(iso_string)
    #print(datetime_object)

    # Create a timedelta object representing few seconds
    timedelta_object = datetime.timedelta(seconds=i)

    # Add the timedelta object to the datetime object
    datetime_object += timedelta_object

    # Convert the datetime object back to an ISO string
    new_iso_string = datetime_object.isoformat().replace('+00:00', 'Z')

    # Print the new ISO string
    #print(datetime_object)
    print(new_iso_string)

    ad_ids = [12345, 54321, 13245, 54231, 14235, 53241, 15234, 43251, 67891, 19876, 68791, 19786, 69781, 18796]

    data.append({
        "ad_id": str(random.choice(ad_ids)),
        "timestamp": new_iso_string,
        "clicks": random.randint(2, 52),
        "views": random.randint(15, 167),
        "cost": round(random.uniform(50.75, 112.66), 2)
    })

# Write the data to a JSON file
with open("mock_ads_data.json", "w") as f:
    json.dump(data, f)
```

mock_ads_data.json:




```json
{
    "ad_id": "19786",
    "timestamp": "2023-08-23T12:01:05Z",
    "clicks": 38,
    "views": 78,
    "cost": 75.37
},
{
    "ad_id": "54321",
    "timestamp": "2023-08-23T12:01:06Z",
    "clicks": 8,
    "views": 145,
    "cost": 63.6
},
{
    "ad_id": "68791",
    "timestamp": "2023-08-23T12:01:07Z",
    "clicks": 48,
    "views": 119,
    "cost": 71.79
},
{
    "ad_id": "18796",
    "timestamp": "2023-08-23T12:01:08Z",
    "clicks": 8,
    "views": 166,
    "cost": 64.56
},
```

3. Created a producer script which was publishing those json ads data into kafka topic.

```python
from confluent_kafka import Producer
import json
import time

def delivery_report(err, msg):
    if err is not None:
        print(f"Message delivery failed: {err}")
    else:
        print(f"Message delivered to {msg.topic()}")

with open('mock_ads_data.json') as f:
    data = json.load(f)

p = Producer({'bootstrap.servers': 'localhost:9092'})

for record in data:
    p.poll(0)
    record_str = json.dumps(record)
    p.produce('ads_data', record_str, callback=delivery_report)
    print("Message Published -> ",record_str)
    time.sleep(0.5) # wait for half second before sending the next record

p.flush()
```

4. Next I set up a spark streaming application where I used the kafka connector to read data from ads_data topic. Then parsed the json data into proper structure using my defined schema.

```python
# Create a SparkSession
spark = SparkSession.builder \
    .appName("Ads Stream") \
    .master("local[3]") \
    .config("spark.sql.shuffle.partitions", "2") \
    .config("spark.streaming.stopGracefullyOnShutdown", "true") \
    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0") \
    .config("spark.cassandra.connection.host", "localhost") \
    .config("spark.cassandra.connection.port", "9042") \
    .getOrCreate()

spark.sparkContext.setLogLevel("ERROR")
load_dotenv()

# Define the schema of the JSON data
schema = StructType([
    StructField("ad_id", StringType()),
    StructField("timestamp", TimestampType()),
    StructField("clicks", IntegerType()),
    StructField("views", IntegerType()),
    StructField("cost", FloatType()),
])

# Read the stream from Kafka
df = spark.readStream.format("kafka")\
    .option("kafka.bootstrap.servers", "localhost:9092")\
    .option("subscribe", "ads_data")\
    .option("startingOffsets", "earliest")\
    .load()


# Parse the JSON data and select the fields
df = df.select(from_json(col("value").cast("string"), schema).alias("data")).select("data.*")
```

5. Further transformed data by performed windowed based aggregation and included cassandra connection script into spark stream app so that I can write aggregated data into cassandra table.

```
# Perform the aggregation in windows of 1 minute and sliding interval of 30 secs
df = df.groupBy("ad_id", window(df.timestamp, "1 minute", "30 seconds")).agg(sum("clicks").alias("total_clicks"),
                                                                              sum("views").alias("total_views"),
                                                                              sum("cost").alias("total_cost"),
                                                                              sum("cost")/sum("views").alias("avg_cost_per_view"))


########## connect cassandra db ##########

# This secure connect bundle is autogenerated when you download your SCB,
# if yours is different update the file name below
cloud_config= {
  'secure_connect_bundle': 'secure-connect-ads-data-db.zip'
}

# This token JSON file is autogenerated when you download your token,
# if yours is different update the file name below
with open("ads_data_db-token.json") as f:
    secrets = json.load(f)

CLIENT_ID = secrets["clientId"]
CLIENT_SECRET = secrets["secret"]

auth_provider = PlainTextAuthProvider(CLIENT_ID, CLIENT_SECRET)
cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
session = cluster.connect()

row = session.execute("select release_version from system.local").one()
if row:
  print(row[0])
else:
  print("An error occurred.")

keyspace="ads_data"
################## end ###################
```

6. But writing data with the logic was like if entry already existed with 'ad_id' then updated that record or else inserting new entry in table.

```
#### create table ########
# Table does not exist, create it
create_table_query = f"""
CREATE TABLE IF NOT EXISTS ads_data.agg_ads_data (
    ad_id TEXT,
    total_clicks INT,
    total_views INT,
    total_cost FLOAT,
    avg_cost_per_view FLOAT,
    PRIMARY KEY (ad_id)
)
"""
session.execute(create_table_query)

# Convert Spark DataFrame to Pandas DataFrame
# pandas_df = df.toPandas()

# Write the output to the console in complete mode
write_query = df.writeStream \
.outputMode("update") \
.format("org.apache.spark.sql.cassandra") \
.option("keyspace", "ads_data") \
.option("table", "agg_ads_data") \
.option("truncate", "false") \
.trigger(processingTime="3 second") \
.start() \
.awaitTermination()
```