Lab6-Task2: Guided solution for Producing Data From Kafka To S3

Objective:
To develop a Spark Streaming application that reads JSON data from a Kafka topic, processes the data, and writes the results into the S3 in Parquet format.

Overview:
This Spark Streaming application is designed to integrate with Apache Kafka for data ingestion. It reads JSON-formatted review data associated with applications, parses this data, and subsequently stores it on S3 for further analysis or processing. The primary workflow includes initializing a Spark session, defining the data schema, streaming data from Kafka, parsing the JSON data, and persisting the processed data to S3.

Guided Solution:
Spark Environment Setup: Import Libraries:

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql import types as T
```

Define the JSON Schema for Incoming Data:
In Apache Spark, when dealing with structured data (especially data that comes in structured formats like JSON), it is often necessary to define the schema to inform Spark about the structure of the data. This helps in efficient processing and in ensuring data integrity.
In the given code, we're defining a schema for the expected JSON data. The schema is defined using Spark's **StructType** and **StructField** classes from the **pyspark.sql.types** module.

```
json_schema = T.StructType([
    T.StructField('application_name', T.StringType()),
    T.StructField('num_of_positive_sentiments', T.LongType()),
    T.StructField('num_of_neutral_sentiments', T.LongType()),
    T.StructField('num_of_negative_sentiments', T.LongType()),
    T.StructField('avg_sentiment_polarity', T.DoubleType()),
    T.StructField('avg_sentiment_subjectivity', T.DoubleType()),
    T.StructField('category', T.StringType()),
    T.StructField('rating', T.StringType()),
    T.StructField('reviews', T.StringType()),
    T.StructField('size', T.StringType()),
    T.StructField('num_of_installs', T.DoubleType()),
```

```
    T.StructField('price', T.DoubleType()),
    T.StructField('age_limit', T.LongType()),
    T.StructField('genres', T.StringType()),
    T.StructField('version', T.StringType())])
```

Initialize Spark Session: Establish a local Spark session utilizing all available cores and adding a Spark-Kafka integration package, which allows Spark to interact with Kafka.

```
spark = SparkSession \
    .builder \
    .master("local") \
    .appName('ex6_store_results') \
    .config('spark.jars.packages', 'org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0') \
    .getOrCreate()
```

Read Streaming Data from Kafka

```
stream_df = spark \
    .readStream \
    .format('kafka') \
    .option("kafka.bootstrap.servers", "course-kafka:9092") \
    .option("subscribe", "gps-with-reviews") \
    .option('startingOffsets', 'earliest') \
    .load() \
    .select(F.col('value').cast(T.StringType()))
```

initializes a streaming DataFrame (stream_df) from a Kafka source. Here's a brief explanation:

- sets up a streaming read from Kafka.
- Specifies the Kafka bootstrap server address as "course-kafka:9092".
- Subscribes to the Kafka topic "gps-with-reviews".
- Sets the starting offsets to the earliest, meaning it will process data from the beginning of the topic.
- Finally, after loading the data, it selects the 'value' column and casts it to a string type.

Today's tip

In Kafka, each message consists of a key and a value. When you consume messages from Kafka using Spark Structured Streaming, these messages are read into a DataFrame with multiple columns, two of which are key and value.

The value column in this context contains the actual content of the Kafka message.

In the provided code, the value column is being selected and cast to a string type. This implies that the actual message content resides in the value column, and it's being treated as a string (which is often the case when dealing with JSON-formatted messages in Kafka).

parse the JSON content

```
#First Part:
parsed_df = stream_df \
   .withColumn('parsed_json', F.from_json(F.col('value'), json_schema))
```

- F.from_json(...): is a PySpark SQL function that parses a column of JSON strings into a structured format based on the provided schema.
- It fetches the column named **value** from the DataFrame. This column contains the JSON strings and apply the provided schema
- After this operation, the DataFrame will contain a new column named parsed_json. This new column holds the structured data that results from parsing the JSON strings in the value column.

For visualization:

Assuming **stream_df** looks like:

```
+--------------------------+
|value                     |
+--------------------------+
|{"name": "John", "age": 30}|
+--------------------------+
```

After the operation, **parsed_df** will look like:

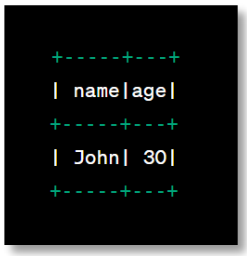
```
+--------------------------+-------------+
|value                     |parsed_json  |
+--------------------------+-------------+
|{"name": "John", "age": 30}|[John, 30]   |
+--------------------------+-------------+
```

#Second part

```
parsed_df = parsed_df.select(F.col('parsed_json.*'))
```

F.col('parsed_json.*'): This syntax is used to select all the individual fields from the structured column **parsed_json** and elevate them to the top level. Essentially, it "flattens" the nested or structured data in **parsed_json**

Continuing from our previous example:
After the **select** operation, **parsed_df** will look like:



write the data as a stream to S3

```
query = parsed_df \
    .writeStream \
    .trigger(processingTime='1 minute') \
    .format('parquet') \
    .outputMode('append') \
    .option("path", "s3a://spark/data/target/google_reviews_calc") \
    .option('checkpointLocation', 's3a://spark/checkpoints/ex6/store_result') \
    .start()
```

- **parsed_df.writeStream**:

This code is preparing to write the data contained in **parsed_df** as a stream. PySpark provides stream processing capabilities, and this is the starting point for setting up the streaming write.

- **.trigger(processingTime='1 minute')**:

This sets the trigger for the streaming query to process data once every minute.

- **.format('parquet')**:

This specifies that the output data format should be **parquet**, which is a popular columnar storage format.

- **.outputMode('append')**:

This indicates that new data should be appended to the result table/directory.

- **.option("path", "s3a://spark/data/target/google_reviews_calc")**:

This sets the destination where the processed streaming data should be written. Here, it is specifying a location in the S3.

- **.option('checkpointLocation', 's3a://spark/checkpoints/ex6/store_result')**:

Streaming queries in PySpark need a location to store recovery information, which is used in case of a failure to restart the stream from where it left off. This is called a checkpoint location, and it's set to a directory in S3 in this code.

- **.start()**:

This method actually starts the streaming query. Until this method is called, no data is processed or written. Once started, the data will be processed according to the configurations set up above and written to the specified location in S3.

streaming query termination

```
query.awaitTermination()

spark.stop()
```

After starting the streaming query, this line of code will make the application block and wait for the streaming query to terminate, either due to a failure or a manual termination. It's essentially saying, "Hold on here and keep running until the streaming process finishes for some reason."

Finally, after the streaming query terminates (or in some setups, potentially never if the **awaitTermination()** doesn't detect a termination), `spark.stop()` line stops the SparkSession, freeing up any resources and ending the application.

Full Code Solution

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql import types as T

json_schema = T.StructType([
    T.StructField('application_name', T.StringType()),
    T.StructField('num_of_positive_sentiments', T.LongType()),
    T.StructField('num_of_neutral_sentiments', T.LongType()),
    T.StructField('num_of_negative_sentiments', T.LongType()),
    T.StructField('avg_sentiment_polarity', T.DoubleType()),
    T.StructField('avg_sentiment_subjectivity', T.DoubleType()),
    T.StructField('category', T.StringType()),
    T.StructField('rating', T.StringType()),
    T.StructField('reviews', T.StringType()),
    T.StructField('size', T.StringType()),
```

```
    T.StructField('num_of_installs', T.DoubleType()),
    T.StructField('price', T.DoubleType()),
    T.StructField('age_limit', T.LongType()),
    T.StructField('genres', T.StringType()),
    T.StructField('version', T.StringType())])

spark = SparkSession \
    .builder \
    .master("local[*]") \
    .appName('ex6_store_results') \
    .config('spark.jars.packages', 'org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0') \
    .getOrCreate()

stream_df = spark \
    .readStream \
    .format('kafka') \
    .option("kafka.bootstrap.servers", "course-kafka:9092") \
    .option("subscribe", "gps-with-reviews") \
    .option('startingOffsets', 'earliest') \
    .load() \
    .select(F.col('value').cast(T.StringType()))

parsed_df = stream_df \
    .withColumn('parsed_json', F.from_json(F.col('value'), json_schema)) \
    .select(F.col('parsed_json.*'))

query = parsed_df \
    .writeStream \
    .trigger(processingTime='1 minute') \
    .format('parquet') \
    .outputMode('append') \
    .option("path", "s3a://spark/data/target/google_reviews_calc") \
    .option('checkpointLocation', 's3a://spark/checkpoints/ex6/store_result') \
    .start()

query.awaitTermination()

spark.stop()
```