Working with Json Array streaming from Kafka¶

Here, we have some instructions on how to handle array of json. Make sure your **Kafka Producer** is publishing an Array of objects.

Step 1 : Initialize Spark Session¶

In [1]:
```
import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-streaming-
kafka-0-10_2.12:3.0.0,org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.0 pyspark-shell'

from pyspark.sql import SparkSession
from pyspark.sql.functions import explode
from pyspark.sql.functions import split
from pyspark.sql import functions as F
from pyspark.sql.types import *

spark = SparkSession \
    .builder \
    .appName("Clickstream Analysis in Spark") \
    .getOrCreate()
```

Step 2 : Read Stream from the Kafka Topic¶

In [8]:
```
topic = "clickstream"
df = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "127.0.0.1:9092") \
  .option("subscribe", topic) \
  .load()
```


In [9]:
```
df = df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
```

Defining the Schema and Parsing the data¶

Here, since we are receiving data as an **array** compared with a single object in the previous example, we need to use `ArrayType` while defining our schema.

Before Running this, make sure that your that you created in **Week 9 , LT1-Producer.ipynb** is producing data in the following or similar format: `[{'Clicks': 0, 'Impressions': 3, 'ts': 1603072527}, {'Clicks': 0, 'Impressions': 3, 'ts': 1603072527}, {'Clicks': 0, 'Impressions': 3, 'ts': 1603072527}, {'Clicks': 0, 'Impressions': 3, 'ts': 1603072527}, {'Clicks': 0, 'Impressions': 11, 'ts': 1603072527}, {'Clicks': 1, 'Impressions': 11, 'ts': 1603072527}]`

In [10]:
```
#Define the schema for the structured datastream received
schema = ArrayType(StructType([
    StructField('Clicks', IntegerType(), True),
    StructField('Impressions', IntegerType(), True),
    StructField('ts', TimestampType(), True)
]))
```

In [11]:
```
df = df.select(F.from_json(F.col("value").cast("string"),
schema).alias('parsed_value'))
```

In [12]:
```
df.printSchema()
root
```

```
 |-- parsed_value: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- Clicks: integer (nullable = true)
 |    |    |-- Impressions: integer (nullable = true)
 |    |    |-- ts: timestamp (nullable = true)
```

You can notice the schema above, the **Columns** are nested. We can use the `explode` function to flatten it.

In [14]:
```
df = df.select(F.explode(F.col("parsed_value")).alias('unnested_value'))
```

In [15]:
```
df.printSchema()
```

```
root
 |-- unnested_value: struct (nullable = true)
 |    |-- Clicks: integer (nullable = true)
 |    |-- Impressions: integer (nullable = true)
 |    |-- ts: timestamp (nullable = true)
```

After using the **.explode()**, the schema looks normal again, we can now proceed with the rest of the operations.

In [ ]:
```
df_formatted = df.select(
                F.col("unnested_value.Clicks").alias("Clicks"),
                F.col("unnested_value.Impressions").alias("Impressions"),
                F.col("unnested_value.ts").alias("ts")
            )
```