Code Logic - Retail Data Analysis

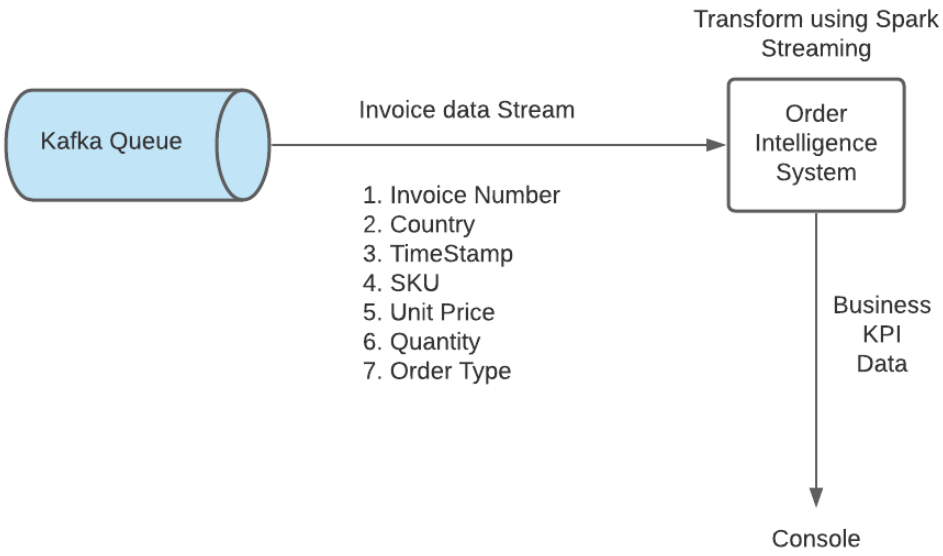
Problem Statement include below points

- Read input data from kafka and print in 1 min processing window
- Calculating additional columns and writing the summarized input table to the console
- Calculating UDF's and additionally calculate time based and time and country based KPI's and print output to hdfs

We are getting the Order Invoices data from a kafka topic given to us.
Topic name: real_time_project
host: port 18.211.252.152
port: 9092

Streaming is real time and as the data comes in, processing happens so we need to have Pre-Processed data ready.

The **spark-streaming.py** code logic is part of "**Order Intelligence Script**" which will **read the data from Kafka and write the data to hdfs and console.**



Once Order Intelligence produces the data which looks like below:

```
{
  "invoice_no": 154132541653705,
  "country": "United Kingdom",
  "timestamp": "2020-09-18 10:55:23",
  "type": "ORDER",
  "items": [
    {
      "SKU": "21485",
      "title": "RETROSPOT HEART HOT WATER BOTTLE",
      "unit_price": 4.95,
      "quantity": 6
    },
    {
```

```
      "SKU": "23499",
      "title": "SET 12 VINTAGE DOILY CHALK",
      "unit_price": 0.42,
      "quantity": 2
    }
  ]
}
```

The understanding of data is integral part of any coding as the structure should be known.
It has below columns:

1. Invoice Number
2. Country
3. Timestamp
4. SKU
5. Unit Price
6. Quantity
7. Order Type

In our Order Intelligence system (spark-streaming.py) code contains:

1. Preprocessor data
2. KPI data (Global and Country level)

As, I have run this code from CDH, first thing is to set the system dependencies for CDH

```
#set system dependencies for CDH
import os
import sys
os.environ["PYSPARK_PYTHON"] = "/opt/cloudera/parcels/Anaconda/bin/python"
os.environ["JAVA_HOME"] = "/usr/java/jdk1.8.0_232-cloudera/jre"
os.environ["SPARK_HOME"]="/opt/cloudera/parcels/SPARK2-2.3.0.cloudera2-1.cdh5.13.3.p0.316101/lib/spark2/"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
sys.path.insert(0, os.environ["PYLIB"] +"/py4j-0.10.6-src.zip")
sys.path.insert(0, os.environ["PYLIB"] +"/pyspark.zip")
```

Import necessary libraries and functions

```
# Importing required functions
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
import pyspark.sql.functions as Func
```

Initialize the Spark session and set the log level to 'ERROR' as a good practice

```
spark=SparkSession \
    .builder \
        .appName("OrderAnalyzer") \
            .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')
```

Define the User defined Functions to be later used in dataframe

```
# Utility function for checking if Order is of type "ORDER"
def get_is_order(order_type):
    if order_type == "ORDER":
        return(1)
    else:
        return(0)
# Utility function for checking if Order is of type "RETURN"
def get_is_return(order_type):
    if order_type == "RETURN":
        return(1)
    else:
        return(0)
# Utility function for calculating total cost of order positive values denotes Order type and negative values denotes Return Type
def get_total_cost(order_type,items):

    total_cost = 0
    for item in items:
            total_cost=total_cost+(item['quantity']*item['unit_price'])
    if order_type == "ORDER":

        return (total_cost)
    else:
         return ((total_cost) * -1)
# Utility function for calculating total number of items present in a single order
```

```
         return ((total_cost) * -1)
# Utility function for calculating total number of items present in a single order
def get_total_item_count(items):
    total_count = 0
    for item in items:
        total_count = total_count + item['quantity']
    return total_count
```

Read from kafka using readstream:

```
# Reading Input from kafka
orderRaw = spark \
    .readStream \
        .format("kafka") \
            .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
                .option("subscribe","real-time-project")  \
                    .option("failOnDataLoss","false") \
                      .option("startingOffsets", "latest")  \
                        .load()
```

Define the Schema and read as “data”

```
jsonSchema =  StructType([
                StructField("country", StringType()),
                StructField("invoice_no", LongType()) ,
                StructField("items", ArrayType(
                    StructType([
                        StructField("SKU", StringType()),
                        StructField("title", StringType()),
                        StructField("unit_price", FloatType()),
                        StructField("quantity", IntegerType())
                              ])
                        )),
                StructField("timestamp", TimestampType()),
                StructField("type", StringType()),
            ])

CastedStream = orderRaw.select(from_json(col("value").cast("string"), jsonSchema).alias("data"))

orderStream = CastedStream.select("data.*")
```

Using the UDF's for calculation of additional columns and adding them to dataframe columns

```
# Defining the UDFs with the utility functions and Calculate additional columns
Total_cost = udf(get_total_cost, FloatType())

orderStream = orderStream \
               .withColumn("total_Cost", Total_cost(orderStream.type,orderStream.items))

isOrderType = udf(get_is_order, IntegerType())

orderStream = orderStream \
              .withColumn("is_Order", isOrderType(orderStream.type))

isReturnType = udf(get_is_return, IntegerType())

orderStream = orderStream \
               .withColumn("is_Return", isReturnType(orderStream.type))

add_total_item_count = udf(get_total_item_count, IntegerType())

orderStream = orderStream \
                    .withColumn("total_items", add_total_item_count(orderStream.items)) \
```

```
# expandedOrderStream contains all the columns for printing to console
expandedOrderStream = orderStream.select("invoice_no","country","timestamp","total_Cost","total_items","is_Order","is_Return")
```

Calculating KPI's

Orders per minute (OPM):
This is nothing but the count of distinct invoices

```
Func.approx_count_distinct("invoice_no").alias("OPM")
```

Total Volume of Sales:

Order cost for complete order
Using udf "get_total_cost" , get the sum of all orders for window period to get total volume of sales

```
.agg(sum("total_Cost").alias("Total_sales_volume"),
```

Rate of return:

For this utilzed the "get_is_order" and "get_is_return" UDF's
Summation of these values respectively will give total_order and total_return

```
sum("is_Order").alias("total_Order"), \
sum("is_Return").alias("total_return"),
```

```
aggStreamByTimeCountry = aggStreamByTimeCountry.withColumn("rate_of_return",
aggStreamByTimeCountry.total_return/(aggStreamByTimeCountry.total_Order+aggStreamByTimeCountry.total_return))
```

Avg Transaction Size:

For this KPI used “get_is_order” and “get_is_return” KPI’s and sum of them stored as total_order and total_return and then used below formula.

```
aggStreamByTime =
aggStreamByTime.withColumn("Average_Transaction_size",aggStreamByTime.Total_sales_volume/(aggStreamByTime.total_Order+aggStreamByTime.total
_return))
```

As the calculation is done, need to print the output in console as well as well as to hdfs file system

Print output to console using expandedOrderStream with Output mode “append” and Truncate as “false” within 1 min processing time.

```
queryConsole = expandedOrderStream \
       .select("invoice_no", "country", "timestamp","total_Cost", "total_items","is_Order","is_Return") \
       .writeStream \
       .outputMode("append") \
       .format("console") \
       .option("truncate", "false") \
       .trigger(processingTime="1 minute") \
       .start()
```

Print time based Output to hdfs by providing path and checkpoint location

```
queryByTime = aggStreamByTime \
              .writeStream \
              .format("json") \
              .outputMode("append") \
              .option("truncate", "false") \
              .option("path","/user/ec2-user/time_KPI") \
              .option("checkpointLocation", "/user/ec2-user/time_KPI") \
              .trigger(processingTime="1 minute") \
              .start()
```

Print time and Country based Output to hdfs by providing path and checkpoint location

```
queryFinal = aggStreamFinal \
             .writeStream \
             .format("json") \
             .outputMode("append") \
             .option("truncate", "false") \
             .option("path","/user/ec2-user/time_country_KPI") \
             .option("checkpointLocation", "/user/ec2-user/time_country_KPI") \
             .trigger(processingTime="1 minute") \
             .start()
```

Running the code for printing the output to console

Steps followed:

1. Go to the folder where code is present **ec2-user/real_time_assignment**
2. Download the jar file using below command (spark-sql-kafka jar file)

wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar

3. Created checkpoint location same as provided in code.(from hdfs)
4. Logout and ran below from ec2-user/real_time_assignment folder.
5. Export the spark_kafka_version **export SPARK_KAFKA_VERSION=0.10**
6. Run the spark2-submit code

spark2-submit --jars spark-sql-kafka-0-10_2.11-2.3.0.jar spark-streaming.py>console_output

For printing the output file in hdfs, we need to create path for output and checkpoint

1. Used below commands from hdfs to create the respective output location

hadoop fs -mkdir /user/ec2-user/time_KPI
hadoop fs -mkdir /user/ec2-user/time_country_KPI

2. Provided respective permissions to read write data using chmod

3. After the output is printed to above path, need to move to local (ec2-user path)

hadoop fs -get /user/ec2-user/time_country_KPI /home/ec2-user/real_time_assignment/Output/time_country_KPI
hadoop fs -get /user/ec2-user/time_KPI /home/ec2-user/real_time_assignment/Output/time_KPI

4. zip the output file and copy to local machine using winscp