Code Logic - Retail Data Analysis

Project : By Malathi Ashok

Problem Statement : In this project, we have been provided real-time sales data of a company across the globe. The data contains information related to the invoices of orders placed by customers all around the world. We are tasked with finding **Key Performance Indicators** **(KPIs)** for the e-commerce company, RetailCorp In using Spark Structured Streaming.

For this Spark Structured Streaming has used and all logic is implemented in python in the file

Spark-Streaming.py.

Spark-Streaming.py is the python code that needs to be run using the following command.

**spark2-submit --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.3.0 spark-streaming.py 18.211.252.152 9092 real-time-project >> console\_output**

This uses org.apache.spark:spark-sql-kafka-0-10\_2.11:2.3.0 library to connect to Kafka from the cloudera cluster to read real time order details data from the topic “real-time-project” on the Kafka Server “18.211.252.152” at port 9092. The output of this command is redirected to a file console\_output to see the results of the code displayed on the console.

Before running, we follow the below steps

1. we login to Cloudera using ssh
2. sudo -i to login as root user
3. export SPARK\_KAFKA\_VERSION=0.10 to set the version of the spark-kafka library
4. hdfs dfs -mkdir /user/root/kafkarealtime/country\_kpi
5. hdfs dfs -mkdir /user/root/kafkarealtime/time\_kpi
6. hdfs dfs -chown root:superuser /user/root/kafkarealtime/country\_kpi
7. hdfs dfs -chown root:superuser /user/root/kafkarealtime/time\_kpi
8. Create the above the directories on hdfs and change the ownership on them to be able to write our json files to them.
9. We are going to use the above directories for checkpointing and saving the output files

The entire code is implemented in the following steps

1. We read the Kafka Topic “real-time-project” from the Kafka Bootstrap Server “**18.211.252.152” at port 9092.**
2. We read the following json string

Which would like the following

{

"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**

}

]

}

Extract the key columns using jsonSchema StructType.

jsonSchema = StructType() \

.add("invoice\_no", LongType()) \

.add("country", StringType()) \

.add("timestamp", TimestampType()) \

.add("type", StringType()) \

.add("items", ArrayType(StructType([

StructField("SKU", StringType()),

StructField("title", StringType()),

StructField("unit\_price", FloatType()),

StructField("quantity", IntegerType()),

])))

Add additional Columns using udf’s (user defined functions) to the datastream . The Columns added are

1. total\_cost, total-items, is\_order, is\_return. The following udf’s are defined accordingly

add\_total\_item\_count = udf(get\_total\_item\_count, IntegerType())

add\_total\_order\_cost = udf(get\_total\_cost, FloatType())

add\_is\_order = udf(get\_is\_order, IntegerType())

add\_is\_return = udf(get\_is\_return, IntegerType())

Once we have all the columns added, we can now calculate the time based kpi’s using the following method

aggStreamByTimeForConsole = expandedOrderStream \

.withWatermark("timestamp", "1 minute") \

.groupBy(window("timestamp", "1 minute", "1 minute")) \

.agg(round(sum("total\_cost"), 2).alias("total\_sale\_volume"), \

count("invoice\_no").alias("OPM"), \

sum("is\_order").alias("total\_orders"), \

sum("is\_return").alias("total\_returned\_orders")) \

.orderBy(asc("window")) \

.select("window", "OPM", "total\_sale\_volume", "total\_orders", "total\_returned\_orders")

kpiStreamByTimeForConsole = aggStreamByTimeForConsole \

.withColumn("rate\_of\_return", round(add\_rate\_of\_return(aggStreamByTimeForConsole.total\_orders, aggStreamByTimeForConsole.total\_returned\_orders),2)) \

.withColumn("average\_transaction\_size", round(add\_avg\_trans\_size(aggStreamByTimeForConsole.total\_sale\_volume, aggStreamByTimeForConsole.total\_orders, aggStreamByTimeForConsole.total\_returned\_orders),2)).

We use 2 more udf’s to calculate “average\_transaction\_size” and “rate\_of\_return”. They are the following

add\_avg\_trans\_size = udf(get\_avg\_trans\_size, FloatType())

add\_rate\_of\_return = udf(get\_avg\_rate\_of\_return, FloatType())

Here we Transform the input stream by performing the group aggregations

using window function with a window interval of 1 minute and a tumbling window of 1 minute allowing for a late arrival of 1 minute. They are grouped based on a 1 minute interval to calculate the total\_sale\_volume, orders\_per\_minute, average\_transaction\_size and rate\_of \_return on a per minute basis.

1. This is then displayed on the console and also written to json file at /user/root/kafkarealtime/time\_kpi. Each json file is an aggregation of a 1 minute window

We then Calculate country based kpi’s using the following method

aggStreamByCountryForConsole = expandedOrderStream \

.withWatermark("timestamp", "1 minute") \

.groupBy(window("timestamp", "1 minute", "1 minute"), "country") \

.agg(round(sum("total\_cost"), 2).alias("total\_sale\_volume"), \

count("invoice\_no").alias("OPM"), \

sum("is\_order").alias("total\_orders"), \

sum("is\_return").alias("total\_returned\_orders")) \

.orderBy(asc("window"),"country") \

.select("window", "country", "OPM", "total\_sale\_volume", "total\_orders", "total\_returned\_orders")

kpiStreamByCountryForConsole = aggStreamByCountryForConsole \

.withColumn("rate\_of\_return", round(add\_rate\_of\_return(aggStreamByCountryForConsole.total\_orders, aggStreamByCountryForConsole.total\_returned\_orders),2))

Transform the input stream by performing the group aggregation using window and country with a window interval of 1 minute and a tumbling

window of 1 minute allowing for a late arrival of 1 minute. They are grouped based on a 1 minute interval to calculate the total\_sale\_volume, orders\_per\_minute, and rate\_of \_return on a per minute basis.

1. This is then displayed on the console and also written to json file at /user/root/kafkarealtime/country\_kpi. . Each json file is an aggregation of a 1 minute window

Then we run the transformation process as long as the stream is reading data from kafka and it is not interrupted by a keyboard interrupt command.