



Load data from Kafka to Hadoop

Steps to load data from Kafka

1. Establishing Spark Session:

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

2. Reading data from Kafka Server & Topic given:

3. Casting raw data as string and aliasing:

```
kafkaDF = lines.selectExpr("cast(key as string)","cast(value as string)")
```

4. Writing data in json file:

```
output = kafkaDF \
    .writeStream \
    .outputMode("append") \
    .format("json") \
    .option("truncate", "false") \
    .option("path","/user/hadoop/clickStreamData/") \
    .option("checkpointLocation", "/user/hadoop/clickstream_checkpoint/") \
    .start()

output.awaitTermination()
```





Steps to run spark_kafka_to_local.py file:

1. Logged in to the EMR instance and below command is executed to download Spark-SQL-Kafka jar file. This jar is used to run the Spark Streaming-Kafka codes

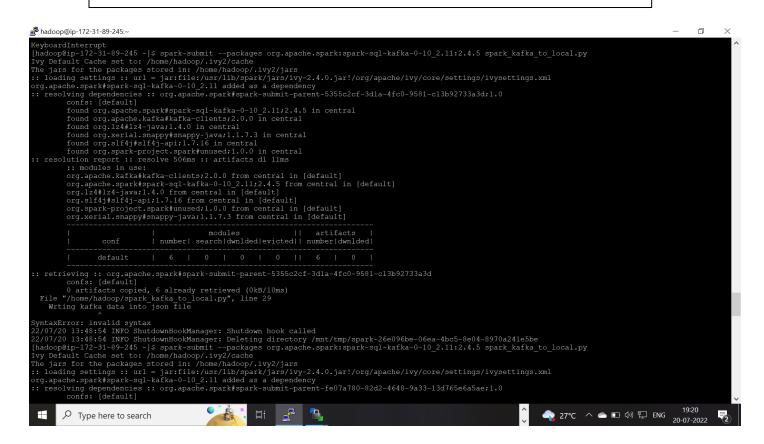
```
wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0- 10_2.11-2.3.0.jar
```

2. Kafka version is set using the following command:

```
export SPARK_KAFKA_VERSION=0.10
```

3. Submitted the spark job using command below:

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-
10_2.11:2.4.5 spark_kafka_to_local.py
```







```
| April | April | Dicobtanapertaters | Registering | Blocktanapertaters | Registering | Registering
```

Steps to flatten the data and load into Hadoop:

1. Establishing spark connection

```
spark=SparkSession \
    .builder \
    .appName('transformKafkaData') \
    .master('yarn') \
    .getOrCreate()
```

2. Reading extracted data:

```
df=spark.read.json('/user/hadoop/clickStreamData/')
```





3. Flattening of raw data and store into respective columns in a dataframe:

```
flatten_df=df.withColumn("value",
F.split(F.regexp_replace(F.regexp_replace("Value",'\{|}',"")
),'\:',','),'\"|"',"").cast("string"),','))\
.withColumn("customer_id", F.element_at("value",2))\
withColumn("app_version", F.element_at("value",4))\
.withColumn("OS_version",F.element_at("value",6))\
.withColumn("lat",F.element_at("value",8))\
.withColumn("lon", F.element_at("value",10))\
.withColumn("page id", F.element at("value",12))\
.withColumn("button_id",F.element_at("value",14))\
.withColumn("is_button_click",F.element_at("value",16))\
.withColumn("is_page_view",F.element_at("value",18))\
.withColumn("is_scroll_up",F.element_at("value",20))\
.withColumn("is_scroll_down",F.element_at("value",22))\
.withColumn("date_hour",F.element_at("value",24))\
.withColumn("minutes",F.element at("value",25))\
.withColumn("seconds",F.element at("value",26))\
.drop("value")
```

4. To concatenate date_hour, minutes and seconds column to make it into timestamp format:

```
flatten_df=flatten_df.select("*",F.concat(col("date_hour"),F.lit(":")
,col("minutes"),F.lit(":"),col("seconds")).alias("timestamp"))
```

5. To remove extra characters \n from timestamp column:

```
flatten_df =
flatten_df.select("*").withColumn("timestamp",F.expr("substring(timestamp, 1,
length(timestamp)-2)")).drop("date_hour").drop("minutes").drop("seconds")
```

6. To write the flattened dataframe in csv file:

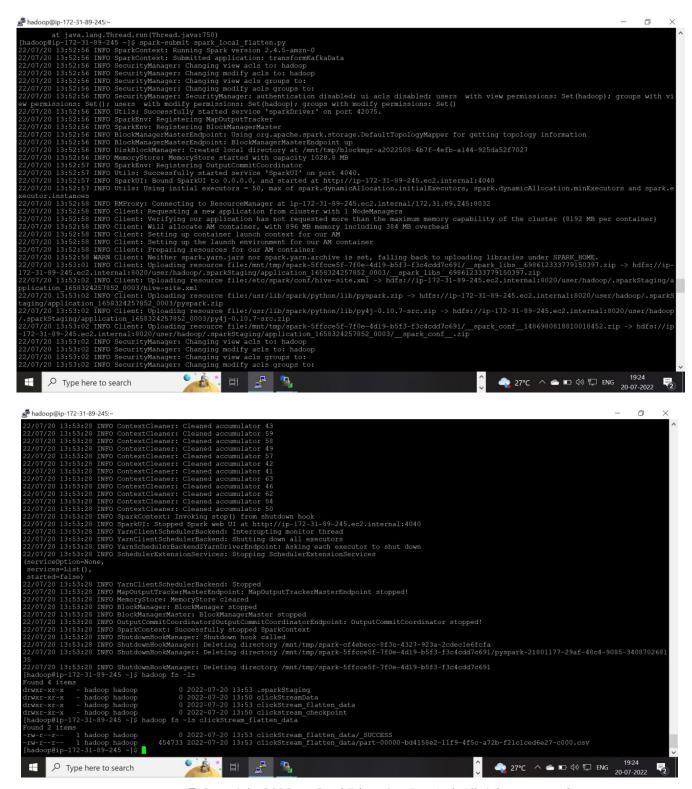
```
flatten_df.write.option("header","true").csv('/user/hadoop/
clickStream_flatten_data/')
```





7. Command to run spark_local_flatten.py file:

spark-submit spark_local_flatten.py







8. Screenshot of the data:

