# Load data from Kafka to Hadoop

## Steps to run the python file to load data from Kafka

- Run the below command to download the Spark-SQL-Kafka jar file. This jar will be 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

```
[hadoop@ip-172-31-37-94 ~]$ wget https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar
--2022-03-15 17:44:25--  https://ds-spark-sql-kafka-jar.s3.amazonaws.com/spark-sql-kafka-0-10_2.11-2.3.0.jar
Resolving ds-spark-sql-kafka-jar.s3.amazonaws.com (ds-spark-sql-kafka-jar.s3.amazonaws.com)... 52.217.230.177
Connecting to ds-spark-sql-kafka-jar.s3.amazonaws.com (ds-spark-sql-kafka-jar.s3.amazonaws.com)|52.217.230.177|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 406313 (397K) [binary/octet-stream]
Saving to: 'spark-sql-kafka-0-10_2.11-2.3.0.jar'

100%[=======================================================================================================================>] 406,313     --.-K/s   in 0.009s

2022-03-15 17:44:25 (43.1 MB/s) - 'spark-sql-kafka-0-10_2.11-2.3.0.jar' saved [406313/406313]

[hadoop@ip-172-31-37-94 ~]$ ls
spark-sql-kafka-0-10_2.11-2.3.0.jar
[hadoop@ip-172-31-37-94 ~]$
```

- Below is the screenshot of the python file (**spark_kafka_to_local.py**) containing the code to ingest the relevant data from Kafka and load the data into hadoop file system.

```
[hadoop@ip-172-31-37-94 ~]$ cat spark_kafka_to_local.py
# Importing the SparkSession from Pyspark's SQL module
from pyspark.sql import SparkSession

# Initializing the Spark session
# Providing a meaningful name to the application relevant to the steps performed in the application
# Set the log level to error
spark = SparkSession  \
        .builder  \
        .appName("ClickstreamDatatoHadoop")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')

# Reading the data from Kafka topic
# startingOffsets used as earliest to load all the data from the begininng
# Provided the given Kafka server along with port
# Provided the Kafka topic name from which the data has to be read
rawData = spark  \
        .readStream  \
        .format("kafka")  \
        .option("multiline","true")  \
        .option("startingOffsets", "earliest")  \
        .option("kafka.bootstrap.servers","18.211.252.152:9092")  \
        .option("subscribe","de-capstone3")  \
        .load()

data = rawData.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")

# Writing the raw data to the Haddop location in JSON format
# Provided target directory in Hadoop where the data will be stored
# Provided target directory in Hadoop where the data checkpoint will be created
finalClickStream = data  \
    .writeStream  \
    .format("json")  \
    .outputMode("append")  \
    .option("truncate", "false")  \
    .option('path', "/user/hadoop/clickStreamData")  \
    .option("checkpointLocation", "/user/hadoop/clickStreamDataCP") \
    .start()  \
    .awaitTermination()
[hadoop@ip-172-31-37-94 ~]$
```

- Below is the spark-submit command to run the **spark_kafka_to_local.py** file.

**Command:** spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5 spark_kafka_to_local.py ##.###.###.### #### de-capstone3

- Below is the screenshot of the data imported from Kafka to Hadoop.

```
[hadoop@ip-172-31-37-94 ~]$ hadoop fs -ls /user/hadoop/clickStreamData
Found 2 items
drwxr-xr-x   - hadoop hadoop          0 2022-03-15 17:50 /user/hadoop/clickStreamData/_spark_metadata
-rw-r--r--   1 hadoop hadoop    1255605 2022-03-15 17:50 /user/hadoop/clickStreamData/part-00000-0d1fa81d-8fc9-4735-bd94-7bf9a96552f8-c000.json
[hadoop@ip-172-31-37-94 ~]$ hadoop fs -cat /user/hadoop/clickStreamData/part-00000-0d1fa81d-8fc9-4735-bd94-7bf9a96552f8-c000.json
{"value":"{\"customer_id\": \"26564820\", \"app_version\": \"3.2.35\", \"OS_version\": \"Android\", \"lat\": \"16.4454865\", \"lon\": \"99.902065\", \"page_id\": \"de54
5711-3914-4450-8c11-b17b8dabb5e1\", \"button_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up\": \
"No\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-09-14 09:59:07\\n\"}"}
{"value":"{\"customer_id\": \"31906387\", \"app_version\": \"2.4.7\", \"OS_version\": \"iOS\", \"lat\": \"-64.813749\", \"lon\": \"-133.527040\", \"page_id\": \"de54571
1-3914-4450-8c11-b17b8dabb5e1\", \"button_id\": \"a95dd57b-779f-49db-819d-b6960483e554\", \"is_button_click\": \"No\", \"is_page_view\": \"No\", \"is_scroll_up\": \"Yes
\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-05-16 16:30:21\\n\"}"}
{"value":"{\"customer_id\": \"25713677\", \"app_version\": \"3.4.12\", \"OS_version\": \"Android\", \"lat\": \"89.943435\", \"lon\": \"127.313415\", \"page_id\": \"b328
829e-17ae-11eb-adc1-0242ac120002\", \"button_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"No\", \"is_scroll_up\": \"
Yes\", \"is_scroll_down\": \"No\", \"timestamp\\n\": \"2020-02-09 00:52:13\\n\"}"}
{"value":"{\"customer_id\": \"83474293\", \"app_version\": \"3.1.8\", \"OS_version\": \"Android\", \"lat\": \"-69.939070\", \"lon\": \"-36.451670\", \"page_id\": \"e7bc
5fb2-1231-11eb-adc1-0242ac120002\", \"button_id\": \"e1e99492-17ae-11eb-adc1-0242ac120002\", \"is_button_click\": \"Yes\", \"is_page_view\": \"No\", \"is_scroll_up\": \
"Yes\", \"is_scroll_down\": \"No\", \"timestamp\\n\": \"2020-06-17 10:42:50\\n\"}"}
{"value":"{\"customer_id\": \"63727807\", \"app_version\": \"2.2.9\", \"OS_version\": \"iOS\", \"lat\": \"64.082108\", \"lon\": \"-81.822078\", \"page_id\": \"e7bc5fb2-
1231-11eb-adc1-0242ac120002\", \"button_id\": \"fcba68aa-1231-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up\": \"Yes\
", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-07-06 02:51:53\\n\"}"}
{"value":"{\"customer_id\": \"73737907\", \"app_version\": \"4.3.19\", \"OS_version\": \"Android\", \"lat\": \"-18.850508\", \"lon\": \"-116.358375\", \"page_id\": \"b3
28829e-17ae-11eb-adc1-0242ac120002\", \"button_id\": \"e1e99492-17ae-11eb-adc1-0242ac120002\", \"is_button_click\": \"No\", \"is_page_view\": \"Yes\", \"is_scroll_up\":
\"No\", \"is_scroll_down\": \"Yes\", \"timestamp\\n\": \"2020-04-26 06:18:16\\n\"}"}
```

Now we will run the **spark_local_flatten.py** python file to clean the loaded Kafka data to a more structured format.

- Below is the screenshot of the python file (**spark_local_flatten.py**) containing the code to clean the loaded Kafka data to a more structured format.

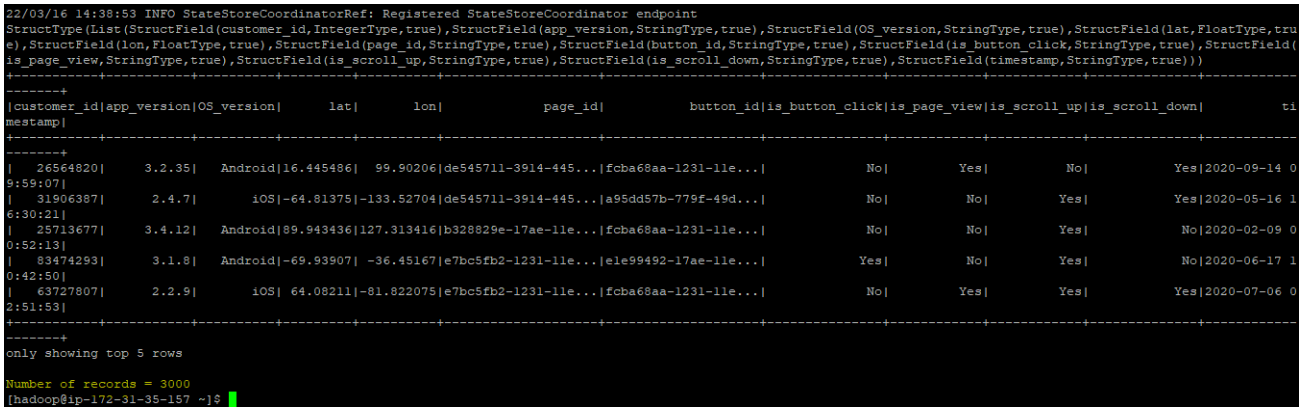
```
[hadoop@ip-172-31-35-157 ~]$ cat spark_local_flatten.py
# Importing the SparkSession, functions and types from Pyspark's SQL module
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import StringType, IntegerType, FloatType, TimestampType
# Initializing the Spark session
# Providing a meaningful name to the application relevant to the steps performed in the application
# Set the log level to error
spark = SparkSession  \
        .builder  \
        .appName("StructuredClickstreamDatatoHadoop")  \
        .getOrCreate()
spark.sparkContext.setLogLevel('ERROR')
# Reading the data from JSON file into a dataframe
df = spark.read.json('/user/hadoop/clickStreamData/*.json')
# Creating a new dataframe by extracting the columns from the JSON value
df1 = df.select(get_json_object(df['value'], "$.customer_id").alias("customer_id"),
                get_json_object(df['value'], "$.app_version").alias("app_version"),
                get_json_object(df['value'], "$.OS_version").alias("OS_version"),
                get_json_object(df['value'], "$.lat").alias("lat"),
                get_json_object(df['value'], "$.lon").alias("lon"),
                get_json_object(df['value'], "$.page_id").alias("page_id"),
                get_json_object(df['value'], "$.button_id").alias("button_id"),
                get_json_object(df['value'], "$.is_button_click").alias("is_button_click"),
                get_json_object(df['value'], "$.is_page_view").alias("is_page_view"),
                get_json_object(df['value'], "$.is_scroll_up").alias("is_scroll_up"),
                get_json_object(df['value'], "$.is_scroll_down").alias("is_scroll_down"),
                get_json_object(df['value'], "$.timestamp\n").alias("timestamp1"))

# Extracting only the required part from the timestamp1 column and saving in new column "timestamp"
df2 = df1.withColumn("timestamp", df1["timestamp1"][0:19])
# Creating a new dataframe by assigning the correct datatypes to the "customer_id", "lat" and "lon" columns
df3 = df2 \
  .withColumn("customer_id", df2["customer_id"].cast(IntegerType()))   \
  .withColumn("lat", df2["lat"].cast(FloatType()))    \
  .withColumn("lon", df2["lon"].cast(FloatType()))
# Selecting only the required columns as per the given schema
df4 = df3.select("customer_id","app_version","OS_version","lat","lon","page_id","button_id","is_button_click","is_page_view","is_scroll_up","is_scroll_down","timestamp"
)
# Printing the schema of the dataframe
print(df4.schema)
# Printing the first 10 records from the dataframe
df4.show(5)
# Printing the number of records
records = df4.count()
print("Number of records = " + str(records))
# Writing the structured clickstream data in CSV format to the hadoop location
df4.coalesce(1).write.format('csv').option('header','false').save('/user/hadoop/structuredclickStreamData', mode='overwrite')
[hadoop@ip-172-31-35-157 ~]$
```

- Below is the spark-submit command to run the **spark_local_flatten.py** file.

**Command:** spark-submit spark_local_flatten.py

- Below is the screenshot of the output after running the **spark_local_flatten.py file.**




## Steps to load the data into Hadoop

Data is loaded into HDFS at **"/user/hadoop/structuredclickStreamData"** directory using the below command which is included in the **spark_local_flatten.py file.**

**Command:**
**df4.coalesce(1).write.format('csv').option('header','false').save('/user/hadoop/structuredclickStreamData', mode='overwrite')**

## Screenshot of the data

- Below is the screenshot of the cleaned and structured data loaded in the Hadoop in CSV format. **(/user/hadoop/structuredclickStreamData)**