Design of data pipeline for processing bookings and click stream data for YourOwnCabs Inc.

Atharv Darekar

10th September 2022

Contents

1 Read Data From Apache Kafka Topic

1.1 Analyze Incoming Data

The data read from the Kafka topic is in JSON format. A sample data looks as given below:

```
{
  "customer_id": "62605529",
  "app_version": "2.1.36",
  "OS_version": "Android",
  "lat": "44.196239",
  "lon": "-15.354515",
  "page_id": "e7bc5fb2-1231-11eb-adc1-0242ac120002",
  "button_id": "a95dd57b-779f-49db-819d-b6960483e554",
  "is_button_click": "Yes",
  "is_page_view": "Yes",
  "is_scroll_up": "No",
  "is_scroll_down": "No",
  "timestamp\n": "2020-06-24 01:55:32\n"
}
```

1.2 Define Schema

From the sample data, it can be inferred that the data needs to be clean before imposing a proper schema. Hence the data is read as string type:

```
schema = StructType(
    [
        StructField(name="customer_id", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="app_version", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="OS_version", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="lat", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="lon", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="page_id", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="button_id", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="is_button_click",
        ↪  dataType=StringType(), nullable=True),
        StructField(name="is_page_view", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="is_scroll_up", dataType=StringType(),
        ↪  nullable=True),
        StructField(name="is_scroll_down", dataType=StringType(),
        ↪  nullable=True),
```

```
        StructField(name="timestamp\n", dataType=StringType(),
        ↪  nullable=True)
    ]
)

```

1.3 Creating A Streaming DataFrame

The data is flatten, cleaned and stored in a dataframe.

```
click_stream_data = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "18.211.252.152:9092") \
    .option("subscribe", "de-capstone3") \
    .option("startingOffsets", "earliest") \
    .load() \
    .select(from_json(col("value").cast("string"),
    ↪  schema).alias("click_stream_data")) \
    .select("click_stream_data.*") \
    .withColumnRenamed("OS_version", "os_version") \
    .withColumnRenamed("timestamp\n", "timestamp")

```

1.4 Store The Streaming DataFrame

The streaming dataframe is written to the Apache Hadoop Distributed File System (HDFS) in CSV format.

```
write_click_stream_data = click_stream_data \
    .writeStream \
    .outputMode("append") \
    .format("csv") \
    .option("truncate", "false") \
    .option("path", "/user/livy/click_stream_data") \
    .option("checkpointLocation",
    ↪  "hdfs:///user/livy/checkpoints/click_stream_data") \
    .trigger(once=True) \
    .start()

```

2 Import Table From Amazon Relational Database Service MySQL Server

2.1 Table Description

The Amazon RDS MySQL server stores the bookings data of the YourOwnCabs. The schema of the table is given below:

```
BOOKING_ID                  STRING
CUSTOMER_ID                 INT
DRIVER_ID                   INT
CUSTOMER_APP_VERSION        STRING
CUSTOMER_PHONE_OS_VERSION   STRING
PICKUP_LAT                  FLOAT
PICKUP_LON                  FLOAT
DROP_LAT                    FLOAT
DROP_LON                    FLOAT
PICKUP_TIMESTAMP            TIMESTAMP
DROP_TIMESTAMP              TIMESTAMP
TRIP_FARE                   FLOAT
TIP_AMOUNT                  FLOAT
CURRENCY_CODE               STRING
CAB_COLOR                   STRING
CAB_REGISTRATION_NO         STRING
CUSTOMER_RATING_BY_DRIVER   INT
RATING_BY_CUSTOMER          INT
PASSENGER_COUNT             INT
```

2.2 Import The Data Into HDFS

The bookings data hosted by the MySQL server is imported into the HDFS and stored in CSV format using the Apache Sqoop.

To import data from MySQL server using the Sqoop, MySQL connector needs to be setup on the Amazon Elastic Map Reduce (EMR) instance.

```
12 echo ">>> Download and install MySQL connector in Amazon Elastic
   ↪ Map Reduce (EMR) master node"
13
14 wget
   ↪ "https://de-mysql-connector.s3.amazonaws.com/mysql-connector-java-8.0.25.tar.gz"
15 sudo rm /usr/lib/sqoop/lib/mysql-connector-java-8.0.25.jar
16 sudo tar -xvf mysql-connector-java-8.0.25.tar.gz
   ↪ --strip-components=1 --directory /usr/lib/sqoop/lib/
   ↪ mysql-connector-java-8.0.25/mysql-connector-java-8.0.25.jar
17
18
19 echo ">>> Run the Apache Sqoop job to import bookings data from
   ↪ Amazon Relational Database Service (RDS) into HDFS"
20
21 sqoop import \
22 --connect
   ↪ jdbc:mysql://upgraddetest.cyaielc9bmnf.us-east-1.rds.amazonaws.com/testdatabase
   ↪ \
23 --table bookings \
24 --username student --password STUDENT123 \
25 --target-dir /user/livy/bookings \
26 -m 1
27
```

3 Ingest The Data Into Apache Hive

3.1 Ingest The Streaming Data

The streaming data stored in the HDFS is ingested into Hive table using the serializing and de-serializing ensuring proper consumption of the data into the table.

```
CREATE TABLE IF NOT EXISTS TBL_CLICK_STREAMS
(
  CUSTOMER_ID       INT,
  APP_VERSION       STRING,
  OS_VERSION        STRING,
  LAT               FLOAT,
  LON               FLOAT,
  PAGE_ID           STRING,
  BUTTON_ID         STRING,
  IS_BUTTON_CLICK   STRING,
  IS_PAGE_VIEW      STRING,
  IS_SCROLL_UP      STRING,
  IS_SCROLL_DOWN    STRING,
  `TIMESTAMP`       TIMESTAMP
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
STORED AS TEXTFILE;

LOAD DATA INPATH "hdfs:///user/livy/feeds_path/click_stream_data"
OVERWRITE INTO TABLE TBL_CLICK_STREAMS;
```

3.2 Ingest The Batch Data

The batch bookings data is ingested in a similar fashion as that of the streaming data.

```
CREATE TABLE IF NOT EXISTS TBL_BOOKINGS
(
    BOOKING_ID                  STRING,
    CUSTOMER_ID                 INT,
    DRIVER_ID                   INT,
    CUSTOMER_APP_VERSION        STRING,
    CUSTOMER_PHONE_OS_VERSION   STRING,
    PICKUP_LAT                  FLOAT,
    PICKUP_LON                  FLOAT,
    DROP_LAT                    FLOAT,
    DROP_LON                    FLOAT,
    PICKUP_TIMESTAMP            TIMESTAMP,
    DROP_TIMESTAMP              TIMESTAMP,
    TRIP_FARE                   FLOAT,
    TIP_AMOUNT                  FLOAT,
    CURRENCY_CODE               STRING,
    CAB_COLOR                   STRING,
```

```
    CAB_REGISTRATION_NO         STRING,
    CUSTOMER_RATING_BY_DRIVER   INT,
    RATING_BY_CUSTOMER          INT,
    PASSENGER_COUNT             INT
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

LOAD DATA INPATH "hdfs:///user/livy/feeds_path/bookings"
OVERWRITE INTO TABLE TBL_BOOKINGS;
```

4 Aggregation Over The Hive Tables Using Apache Spark

The bookings data is aggregated date-wise using the PySpark, the aggregated data is stored as a Hive table as well as in CSV format in HDFS.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, to_date, count, sum as
↪  pyspark_sum

spark = SparkSession \
    .builder \
    .appName('aggregate_bookings_data') \
    .enableHiveSupport() \
    .getOrCreate()

bookings = spark.sql("SELECT * FROM TBL_BOOKINGS")

bookings_aggregated = bookings.withColumn("trip_date",
↪  to_date(col("pickup_timestamp"), "yyyy-MM-dd")) \
    .groupBy(col("trip_date")) \
    .agg(
        count("booking_id").alias("number_of_trips"),
        pyspark_sum("trip_fare").alias("total_trip_amount"),
        pyspark_sum("tip_amount").alias("total_tip_amount"),

        ↪  pyspark_sum("passenger_count").alias("total_passenger_counts")
        ) \
    .orderBy("trip_date")

bookings_aggregated.createOrReplaceTempView("VW_AGGREGATED_BOOKINGS")
spark.sql("CREATE TABLE IF NOT EXISTS TBL_AGGREGATED_BOOKINGS \
            AS SELECT * FROM VW_AGGREGATED_BOOKINGS")

bookings_aggregated \
    .write \
    .csv("/user/livy/aggregate_batch_data", header="false")
```

5 Run The Data Pipeline

The complete process from data ingestion, transformation and loading is automated using a simple bash script.

To execute the script, change the working directory to the root of the repository. [1]

5.1 Setup Directories To Stage Data

```
#!/usr/bin/env bash

echo ">>> Setup Apache Hadoop Distributed File System (HDFS)
↪  directories to import click stream data"

hdfs dfs -rm -r /user/livy
hdfs dfs -mkdir -p /user/livy/click_stream_data

```

5.2 Execute Spark Structured Streaming Application

```
echo ">>> Run the Apache Spark Structured Streaming application
↪  to read data from Apache Kafka topic and store as CSV files
↪  in HDFS"

spark-submit --packages
↪  org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.5
↪  data_sourcing/import_streaming_data.py

```

5.3 Setup Feeds Path To Ingest The Data Into Hive

```
echo ">>> Setup feeds path and copy data into the feeds path for
↪  ingestion of the imported data into Apache Hive"

hdfs dfs -mkdir -p
↪  {/user/livy/feeds_path/click_stream_data,/user/livy/feeds_path/bookings}
hdfs dfs -cp /user/livy/click_stream_data/*.csv
↪  /user/livy/feeds_path/click_stream_data
hdfs dfs -cp /user/livy/bookings/part*
↪  /user/livy/feeds_path/bookings

```

5.4 Ingest The Data Into Hive

```
echo  ">>> Ingest the click stream data and bookings data into
↪  Hive tables"

hive -f data_sourcing/ingest_streaming_data.sql
```

[1] Refer to pipeline run video.

```
hive -f data_sourcing/ingest_batch_data.sql

```

5.5 Aggregate The Bookings Data

```
echo ">>> Run Apache Spark application to aggregate bookings data, load the aggregated data into a Hive table and store as CSV files in HDFS"

spark-submit data_transformation/aggregate_batch_data.py
```