**Streaming Job Documentation**

**Overview**

This document outlines the configuration, implementation, and usage of a Spark Structured Streaming job for processing data from a Kafka topic and storing it in Hive tables

### Environment Setup

* **Spark Session Configuration:**
  + **Application Name:** KafkaStreamingExample
  + **Spark Packages:** org.apache.spark

.12:3.2.4

* + **Hadoop Configuration:** fs.defaultFS=hdfs://localhost:9000

### Kafka Connection Details

* **Bootstrap Servers:** pkc-56d1g.eastus.azure.confluent.cloud:9092
* **Topic:** salma\_topic
* **Security Settings:** SASL\_SSL with PLAIN authentication mechanism using provided credentials.

### Schema Definition

* **JSON Schema:**

schema = StructType() \

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

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

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

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

.add("metadata", MapType(StringType(), StringType())) \

.add("quantity", IntegerType()) \

.add("totalAmount", DoubleType()) \

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

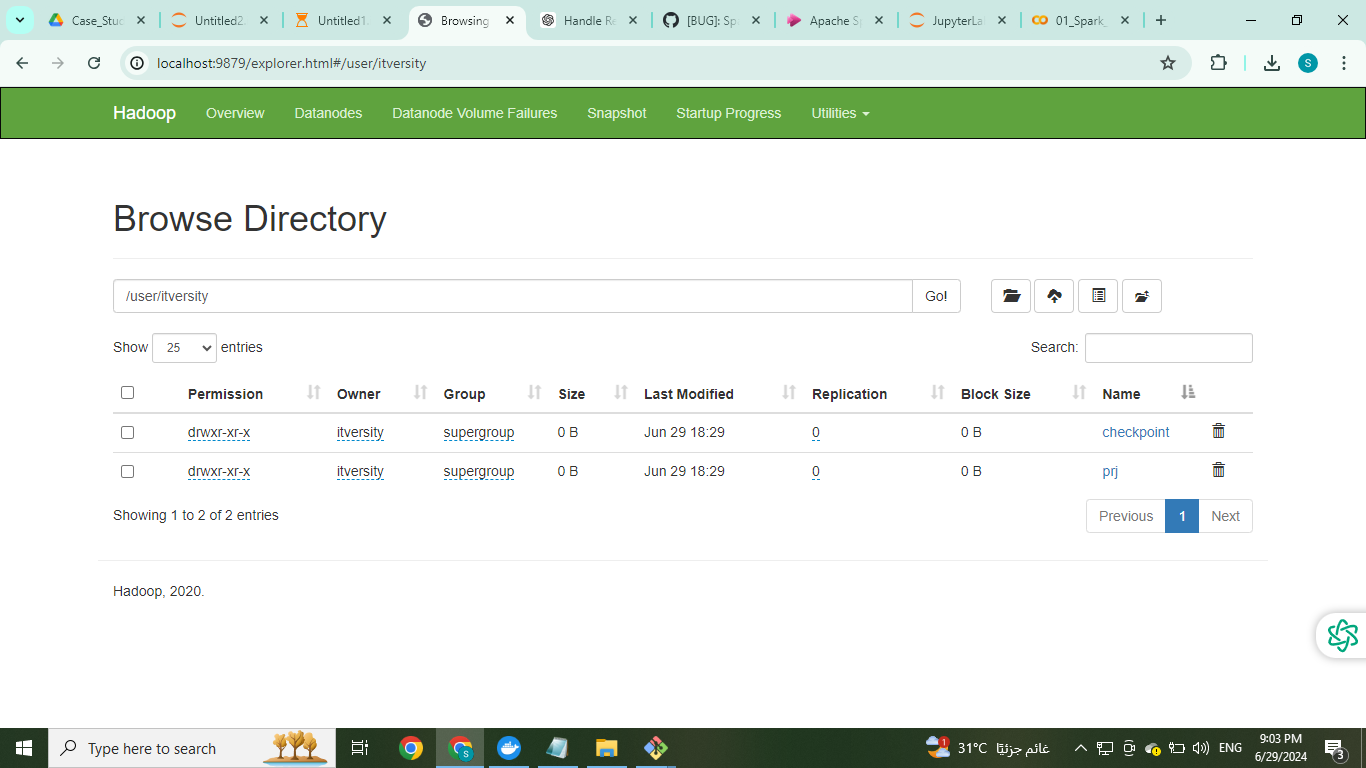
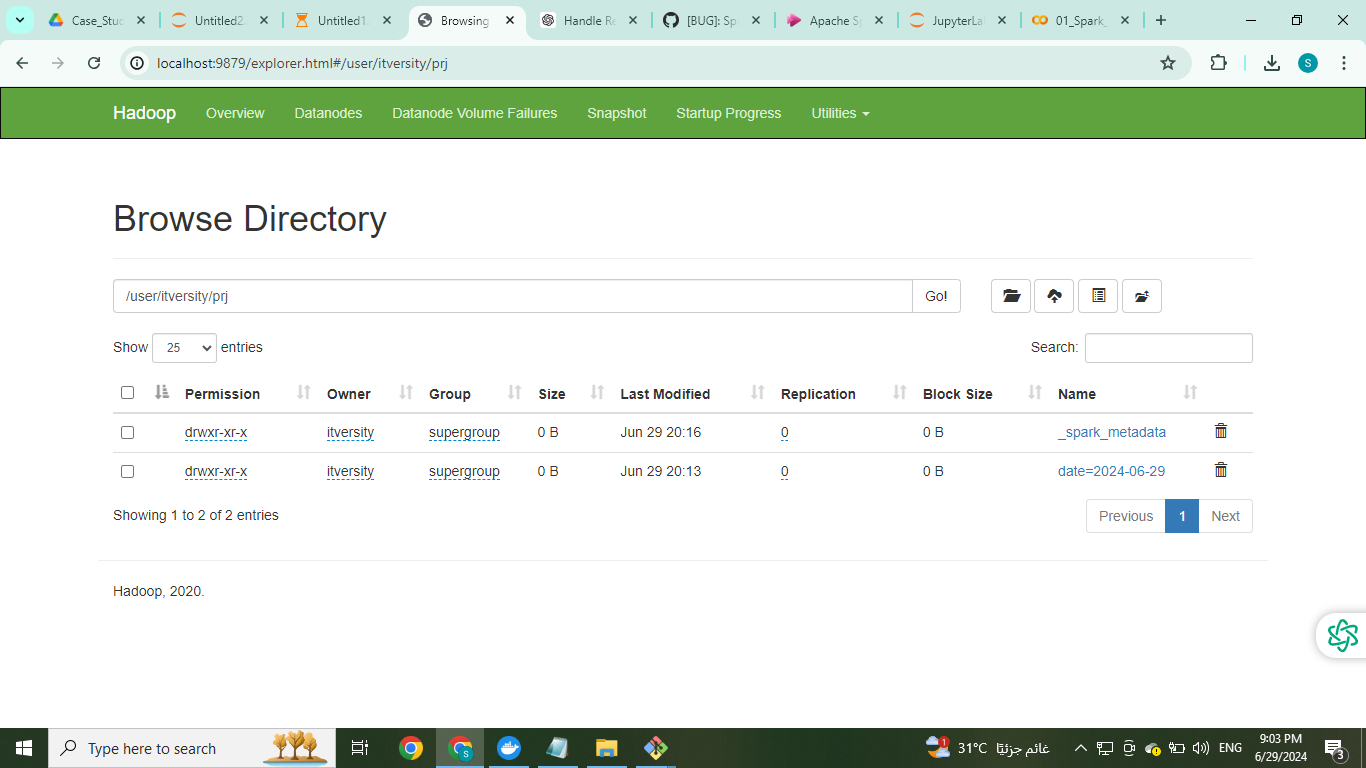
.add("recommendedProductId", StringType()) \

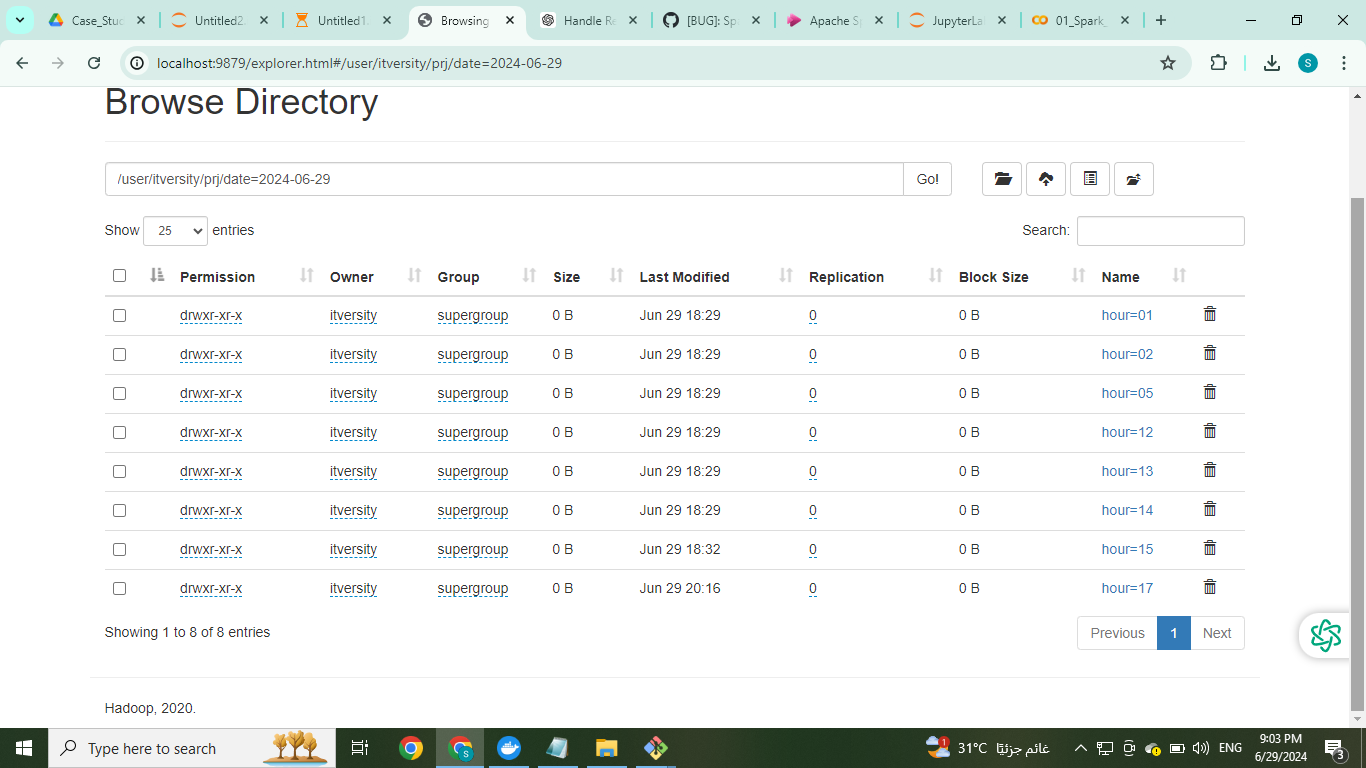
.add("algorithm", StringType())

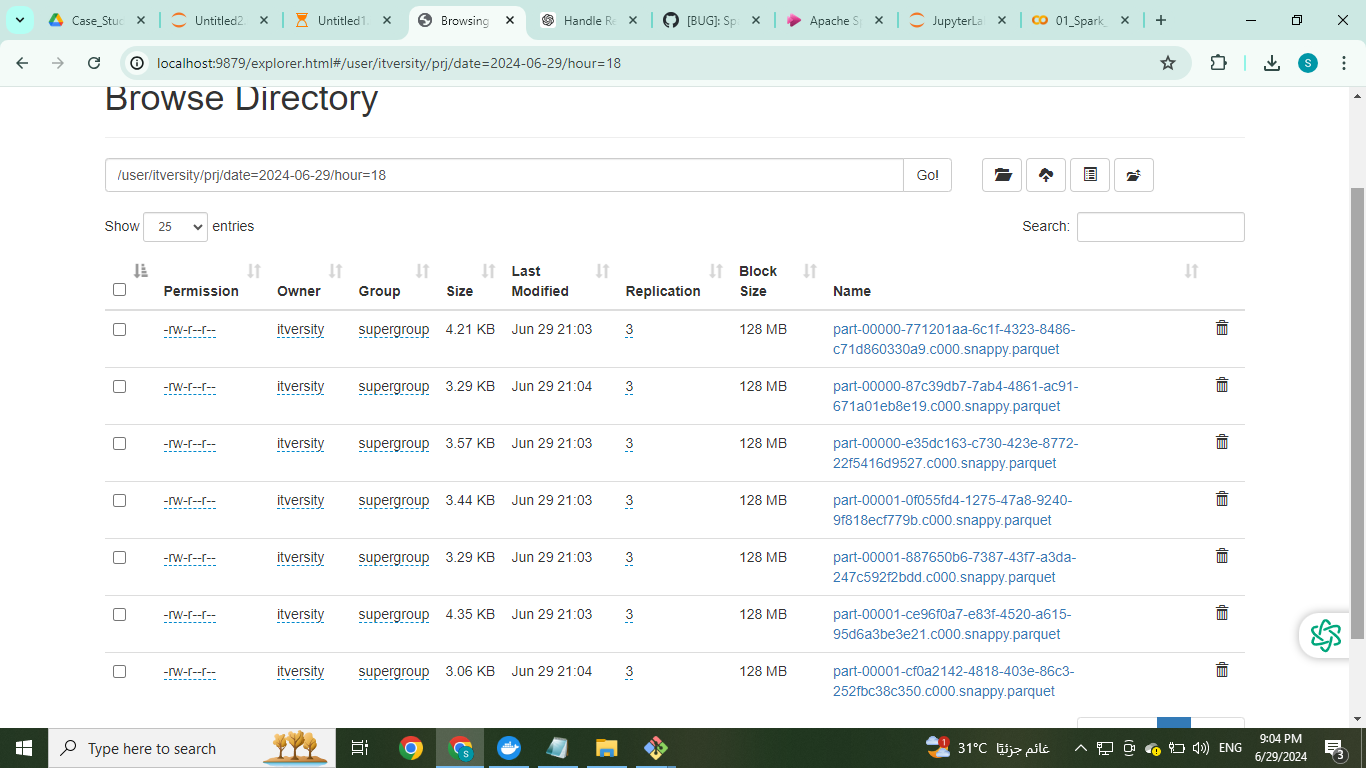
**Streaming Data Processing**

* **Data Ingestion:**
  + Read data from Kafka topic salma\_topic as a streaming DataFrame.
  + Start reading from the earliest available offset.
* **JSON Parsing and Data Transformation:**
  + Convert JSON strings to structured data using the defined schema.
  + Handle schema based on eventType to derive relevant columns (quantity, totalAmount, etc.).
  + Parse timestamp field into date and hour columns for time-based partitioning.

**Data Storage**

* **Output Format:**
  + Data is written in Parquet format to HDFS.
  + Partitioned by date and hour to optimize querying and data retrieval.
* **Output Path:**
  + /user/itversity/prj for storing the processed data.
  + Checkpoint location: /user/itversity/checkpoint to store streaming metadata.

****



from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("ReadParquetFromHDFS") \

.config("spark.hadoop.fs.defaultFS", "hdfs://localhost:9000") \

.getOrCreate()

# Specify the path to your Parquet file

parquet\_path = "/user/itversity/prj/date=2024-06-29/hour=01"

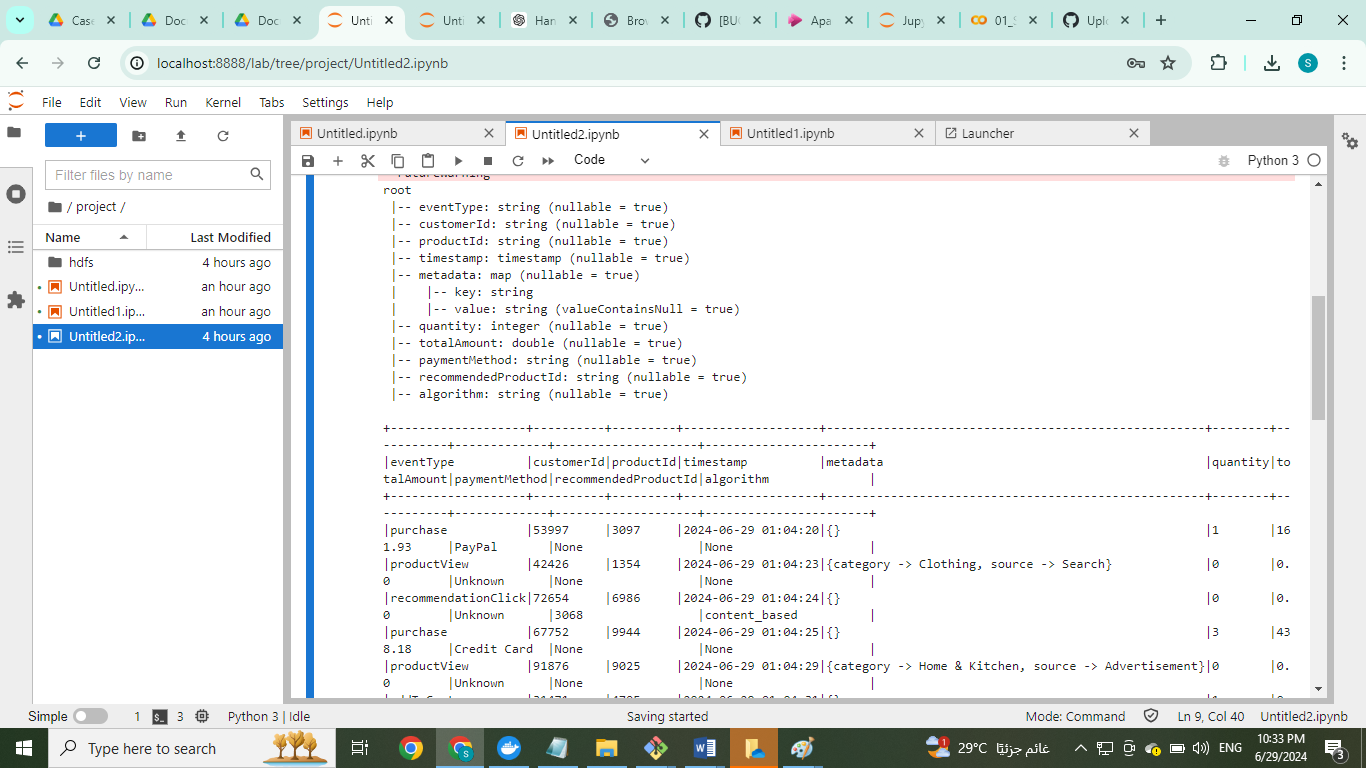
# Read the Parquet file into a DataFrame

df = spark.read.parquet(parquet\_path)

# Show the schema and preview of the data

df.printSchema()

df.show(truncate=False)



**Start Streaming Query**

* **Execution:**
  + Start the streaming query to continuously process incoming data.

query.awaitTermination()

### Hive Table Creation

#### **Step 1: Define Hive Table Schema**

Hive table logs\_table is created with the following schema matching the data structure stored in Parquet format on HDFS, including partitions by date and hour

CREATE EXTERNAL TABLE IF NOT EXISTS logs\_table (

eventType STRING,

customerId STRING,

productId STRING,

`timestamp` STRING,

metadata MAP<STRING, STRING>,

quantity INT,

totalAmount DOUBLE,

paymentMethod STRING,

recommendedProductId STRING,

algorithm STRING

)

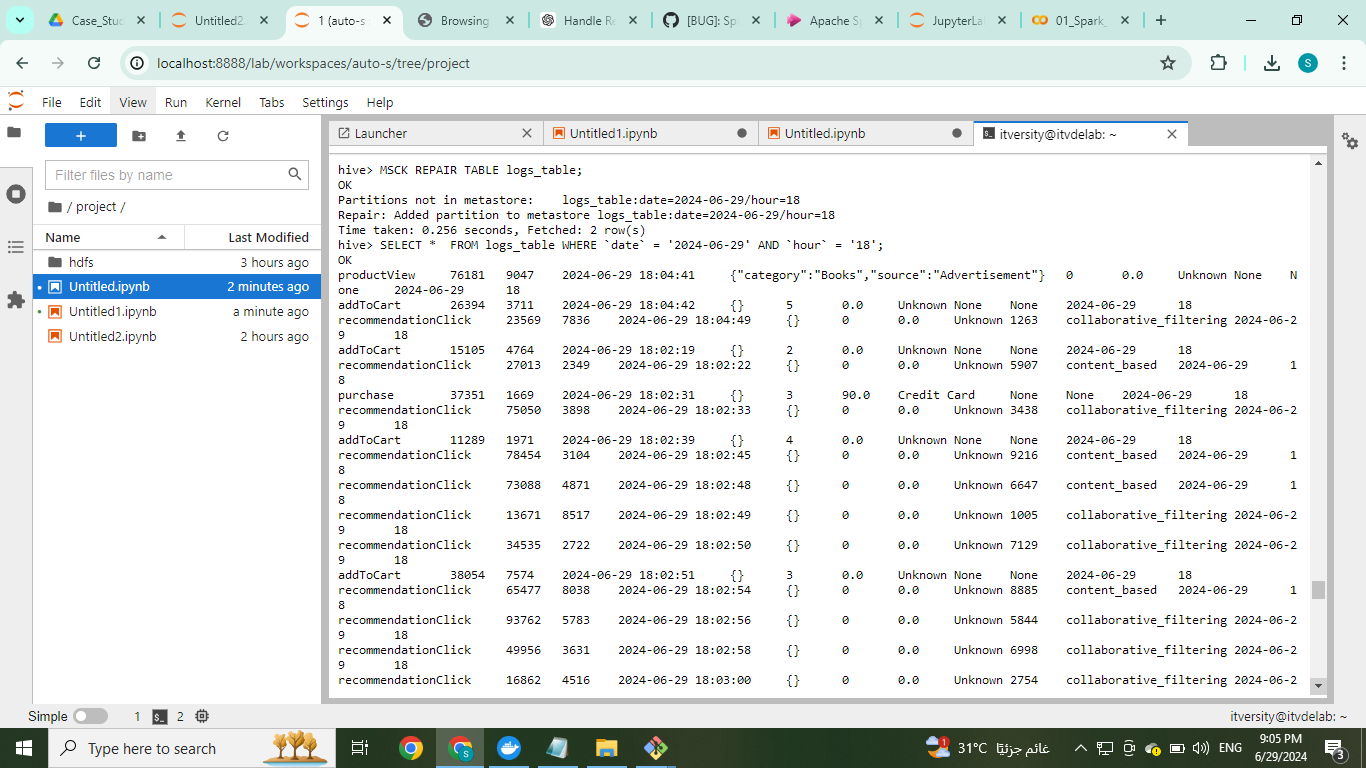
PARTITIONED BY (`date` STRING, `hour` STRING)

STORED AS PARQUET

LOCATION '/user/itversity/prj';

#### **Step 2: Add Partitions**

After defining the table, add partitions for existing data on HDFS. Since the data is partitioned by date and hour, you need to add partitions accordingly.

**MSCK REPAIR TABLE logs\_table;**

**Business Requirements**

* The stored data in Hive (logs\_table) is structured to support business insights and reporting.
* Example queries can be executed directly against logs\_table to derive business metrics and trends.

##### **Example Queries**

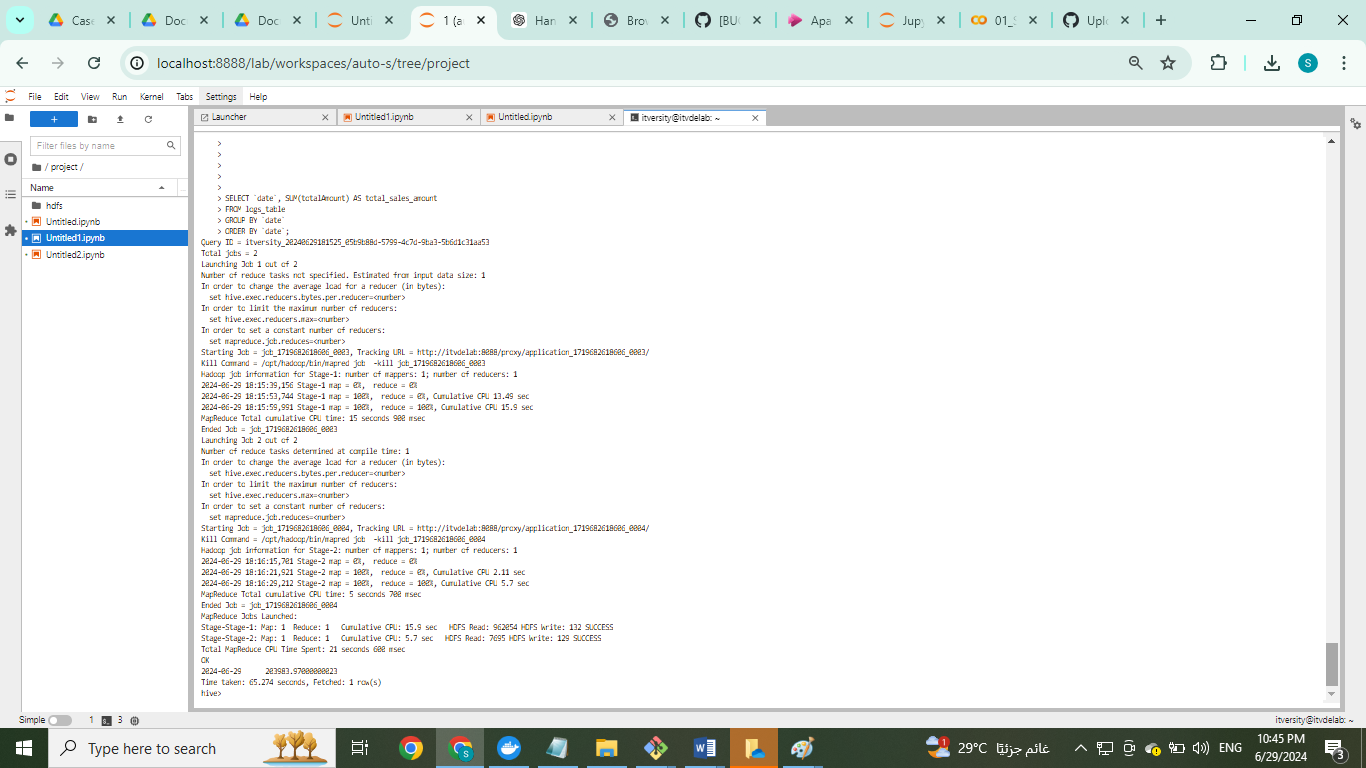
1. **Total Sales Amount by Date:**

SELECT date, SUM(totalAmount) AS total\_sales\_amount

FROM logs\_table

GROUP BY date

ORDER BY date;



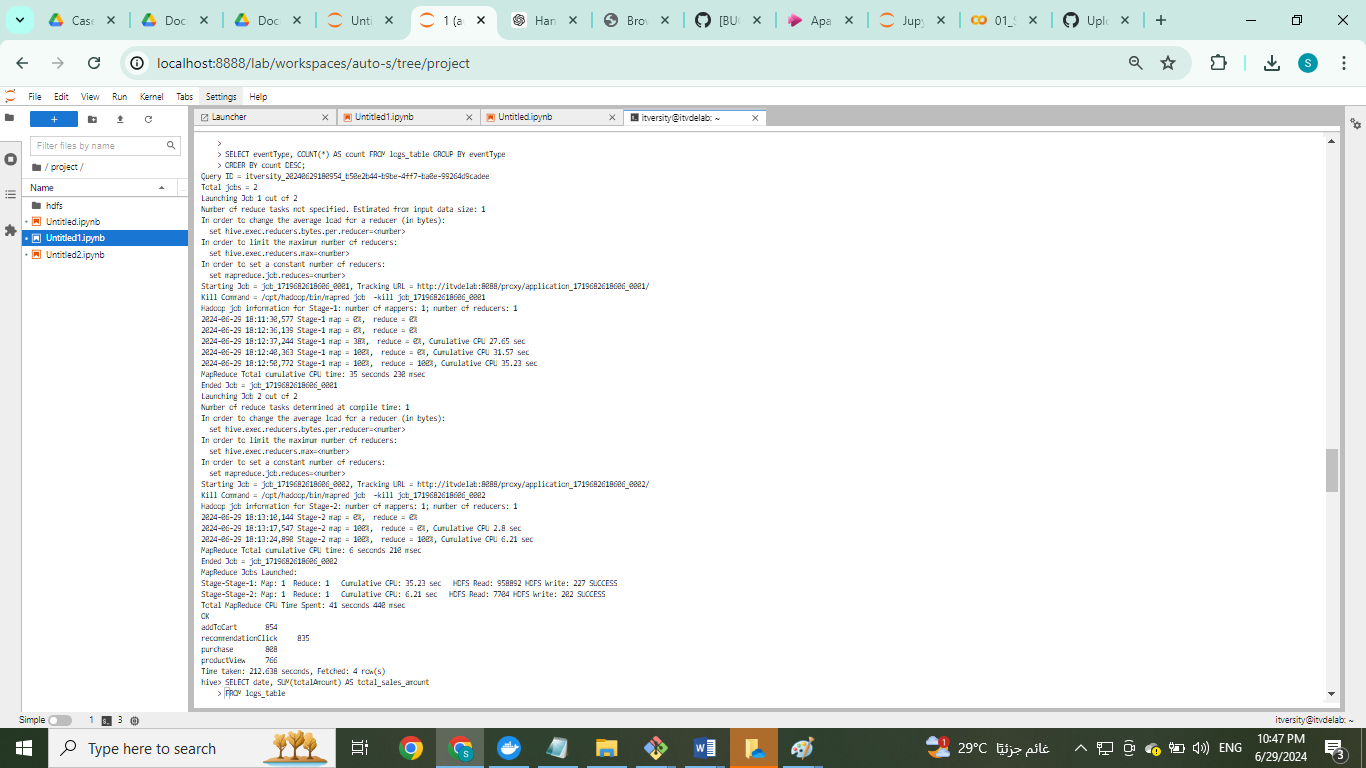
1. **Event Type Count Analysis**

SELECT eventType, COUNT(\*) AS count

FROM logs\_table

GROUP BY eventType

ORDER BY count DESC;



### Data Compaction

#### Background

To optimize storage and query performance, we observed that the streaming job was creating small files. Hence, we implemented a data compaction job that runs hourly to consolidate data for each hour partitioned by date and hour.

#### Implementation

* **Compaction Job:**
  + **Path Existence Check:**
    - Before execution, ensure that the input path for compaction exists using HDFS commands.
  + **Compaction Logic:**
    - Retrieve data from the previous hour's partition.
    - Repartition and write the compacted data back to the same location.
  + **Integration with Streaming Job:**
    - The compaction job is scheduled using crontab to run every hour, compacting data from the previous hour's partition.

