



PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage
Bengaluru 560085

Department of Computer Science and Engineering
B. Tech. CSE - 6th Semester
Jan – May 2023

UE20CS343
DATABASE TECHNOLOGIES (DBT)

Project Report

Analysis of food delivery data

TEAM #: 2

PES1UG20CS443: Sukruth Keshava Gowda

PES1UG20CS505: Vinesh S

PES1UG20CS519: Yashas Kashyap

PES1UG20CS669: P Chaitanya P S

Class of Prof. Vinay Joshi

Table of Contents

1. Introduction
2. Installation of Software [include version #s and URLs]
 - a. Streaming Tools Used
 - b. DBMS Used
3. Problem Description
4. Individual Team Member Contribution
5. Architecture Diagram
6. Input Data
 - a. Source
 - b. Description
7. Streaming Mode Experiment
 - a. Windows
 - b. Workloads
 - c. Code like SQL Scripts
 - d. Inputs and Corresponding Results
8. Batch Mode Experiment
 - a. Description
 - b. Data Size
 - c. Results
9. Comparison of Streaming & Batch Modes
 - a. Results and Discussion
10. Conclusion
11. References
 - a. URLs

1. Introduction

The project involves processing of data generated from a data generator using Apache Spark Streaming and Spark SQL. The data is then streamed to Apache Kafka Streaming for publishing and subscribing the results or producing and consuming from at least three topics. The data is then stored in a DBMS of choice such as PostgreSQL. Additional tools like Zookeeper can also be used as required.

The project aims to showcase the use of distributed computing using Apache Spark and Kafka Streaming to process and analyze data in real-time. Spark SQL queries can be used to carry out various actions, transformations, and aggregations on the input data. Kafka Streaming enables seamless streaming of data between various applications, making it an ideal choice for data streaming and analytics.

By storing the data in a DBMS like PostgreSQL, it becomes easier to access and query the data for further analysis or reporting. The project can be used for various use cases such as real-time data analysis, fraud detection, and anomaly detection.

2. Softwares used

1. Apache Spark Streaming
2. Spark SQL
3. Docker
- 4 . Apache Kafka
5. PostgreSQL (or another DBMS of your choice)
6. Zookeeper

3. Problem Description

We have a system that generates random data using an program we created . This data includes information about various events. Our goal is to analyze this data and derive insights that can help us make informed decisions to improve the business of the food delivery service .

To achieve this, we plan to use a combination of stream and batch processing techniques. The real-time data will be processed using Apache Kafka Streaming and Apache Spark Streaming, while batch processing will be performed using Apache Spark SQL. We will store the processed data in a DBMS, such as PostgreSQL, and use it to generate reports and visualizations.

Specifically, our system will consist of the following components:

Data Generation: We will use two Python scripts, generator.py and creator.py, to simulate the generation of real-time data. The generator.py script will generate events and write them to a Kafka topic, while the creator.py script will create a PostgreSQL database and table to store the processed data.

Apache Kafka Streaming: We will use Kafka to read the data from the Kafka topic and process it in real-time. We will use Kafka's publish/subscribe mechanism to consume the data from the topic and perform various operations on it, such as filtering, grouping, and aggregating.

Apache Spark Streaming: We will use Spark Streaming to process the data in real-time. We will use Spark's streaming API to read the data from Kafka and perform transformations and operations on it. We will also use Spark SQL to execute multiple workloads, such as Spark SQL queries, to carry out actions, transformations, or aggregations on the input data.

Apache Spark SQL: We will use Spark SQL to perform batch processing on the processed data. We will store the processed data in a DBMS, such as PostgreSQL, and use Spark SQL to execute complex SQL queries on it. These queries will help us derive insights and generate reports about our business.

DBMS: We will use PostgreSQL to store the processed data. We will create a table with the necessary columns to store the data and write the processed data to it. We will use SQL queries to extract and analyze the data from the table.

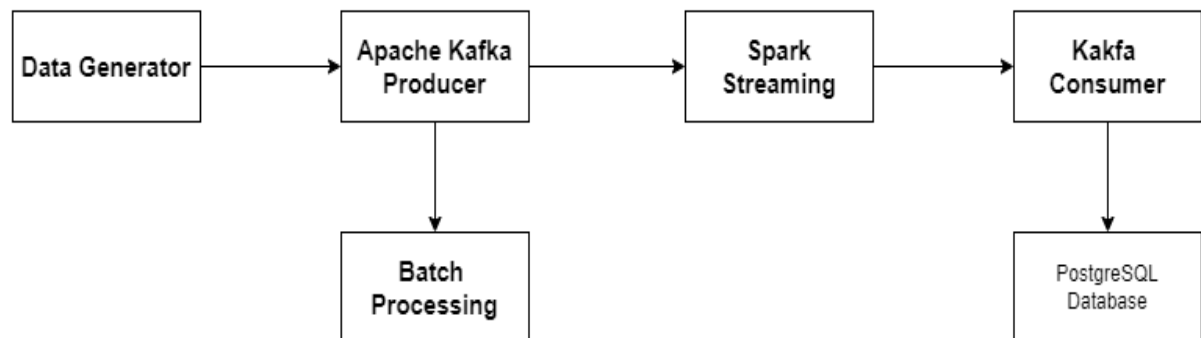
Other tools: We may use other tools, such as Zookeeper, to manage the Kafka cluster and ensure high availability and fault tolerance.

The goal of this project is to demonstrate the use of stream and batch processing techniques to analyze real-time data and derive insights that can help us make informed decisions. We will use Apache Kafka Streaming, Apache Spark Streaming, and Apache Spark SQL to process the data in real-time and perform batch processing on it. We will store the processed data in PostgreSQL and use it to generate reports and visualizations.

4. Individual team member contribution

- Dataset creation and Population – Vinesh S
- Kafka Producer And Batch Processing - Sukruth K
- Spark Streaming – P Chaitanya P S
- Dockerizing entire process and storing processed data in Database – Yashas K

5. Architecture diagram



6. Input Data

We have a Python script that generates random restaurant orders and updates their statuses. The script uses the Faker library to generate fake names, phone numbers, and other data, and the Kafka library to produce messages to Kafka topics.

The script has two main functions:

- **generate_order()**: This function generates a fake order with random items, prices, and quantities. The order is returned as a dictionary.
- **generate_order_update()**: This function selects a random order from the **generatedOrders** dictionary and updates its status to a new randomly selected status. If the current status is "out-for-delivery", the function also adds fake delivery boy data to the order. If the current status is "delivered" or "refund", the order is removed from the **generatedOrders** dictionary. The updated order is returned as a dictionary.

The script also defines several constants and variables, including Kafka topic names, a Kafka producer, a **generatedOrders** dictionary that holds all the random orders and their statuses . Overall, the script simulates a restaurant order processing system that generates new orders and updates their statuses until they are either delivered or refunded. The Kafka topics allow other services to subscribe to the order updates and take appropriate actions based on the new status.

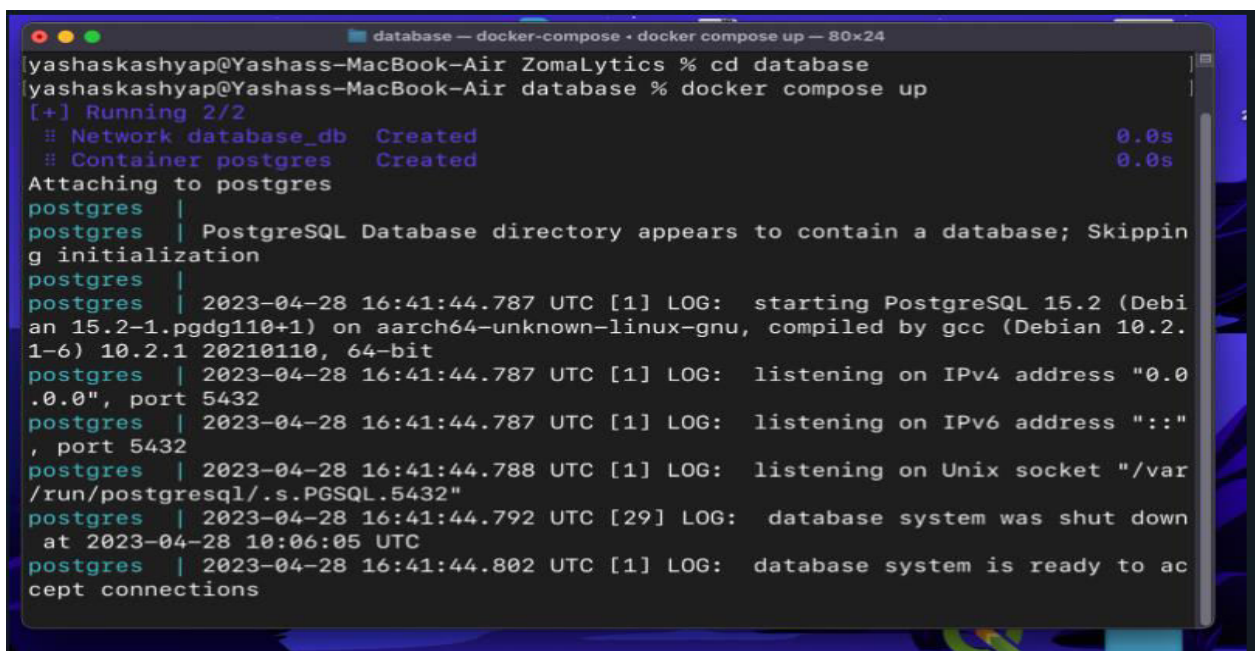
Using Dockers

In this project, Docker is used to containerize and isolate the different services and components required for stream and batch processing. Docker provides a lightweight virtualization solution that allows developers to package their applications along with their dependencies into a container, which can then be deployed across multiple environments with ease.

By using Docker, we can ensure that the various components required for stream and batch processing, such as Apache Kafka, Apache Spark, and PostgreSQL, are deployed in a consistent and reproducible manner. This helps to eliminate the issues related to configuration drift and ensures that the application runs consistently across different environments.

Moreover, Docker allows us to easily scale the application by creating multiple instances of the same container, thereby enabling horizontal scaling. Additionally, Docker provides a simple and efficient way to manage the different dependencies of the project, thereby reducing the overall complexity of the application deployment.

Overall, Docker plays a crucial role in simplifying the deployment and management of the various components required for stream and batch processing, thereby making it easier to build and maintain robust and scalable applications.

A terminal window titled 'database — docker-compose · docker compose up — 80x24' showing the execution of 'docker compose up' in a directory named 'database'. The output shows the network 'database_db' and container 'postgres' being created. It then shows the container attaching to 'postgres' and displaying logs for PostgreSQL 15.2, including startup messages and a previous shutdown notice.

```
[yashaskashyap@Yashass-MacBook-Air ZomaLytics % cd database
[yashaskashyap@Yashass-MacBook-Air database % docker compose up
[+] Running 2/2
  ⚙ Network database_db Created                                0.0s
  ⚙ Container postgres Created                                0.0s
Attaching to postgres
postgres |
postgres | PostgreSQL Database directory appears to contain a database; Skipping initialization
postgres |
postgres | 2023-04-28 16:41:44.787 UTC [1] LOG:  starting PostgreSQL 15.2 (Debian 15.2-1.pgdg110+1) on aarch64-unknown-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
postgres | 2023-04-28 16:41:44.787 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
postgres | 2023-04-28 16:41:44.787 UTC [1] LOG:  listening on IPv6 address ":::", port 5432
postgres | 2023-04-28 16:41:44.788 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
postgres | 2023-04-28 16:41:44.792 UTC [29] LOG:  database system was shut down at 2023-04-28 10:06:05 UTC
postgres | 2023-04-28 16:41:44.802 UTC [1] LOG:  database system is ready to accept connections
```

```

yashaskashyap@Yashass-MacBook-Air kafka_config % docker compose up
[+] Running 3/3
   :: Network kafka_config_default
ted 0.2siner kafka          Creating          0.3s
   :: Container zookeeper
ted 0.4s
   :: Container kafka
ted 0.4s
   " zookeeper The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested 0.0s
Attaching to kafka, zookeeper
kafka      | waiting for kafka to be ready
kafka      | [Configuring] 'port' in '/opt/kafka/config/server.properties'
kafka      | [Configuring] 'advertised.host.name' in '/opt/kafka/config/server.p
properties'
kafka      | Excluding KAFKA_HOME from broker config
kafka      | [Configuring] 'log.dirs' in '/opt/kafka/config/server.properties'
kafka      | Excluding KAFKA_VERSION from broker config
kafka      | [Configuring] 'zookeeper.connect' in '/opt/kafka/config/server.prop
erties'

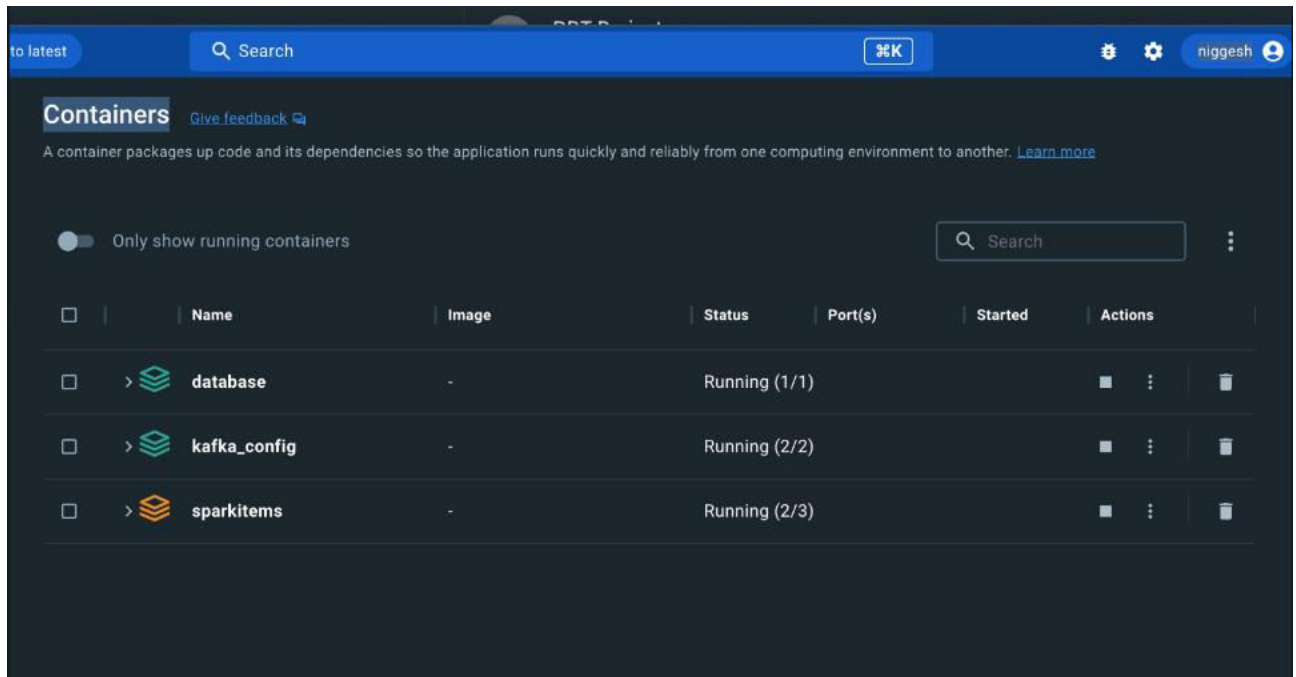
```

```

yashaskashyap@Yashass-MacBook-Air sparkItems % docker compose up
[+] Running 4/4
   :: Network sparkitems_default          Created          0.0s
   :: Container sparkitems-spark-master-1 Created          0.1s
   :: Container sparkitems-spark-worker-a-1 Created          0.1s
   :: Container sparkitems-spark-worker-b-1 Created          0.1s
Attaching to sparkitems-spark-master-1, sparkitems-spark-worker-a-1, sparkitems-spark-worker-b-1
sparkitems-spark-master-1 | spark 16:47:52.73
sparkitems-spark-master-1 | spark 16:47:52.73 Welcome to the Bitnami spark container
sparkitems-spark-master-1 | spark 16:47:52.73 Subscribe to project updates by watching https://github.com/bitnami/containers
sparkitems-spark-master-1 | spark 16:47:52.73 Submit issues and feature requests at https://github.com/bitnami/containers/issues
sparkitems-spark-master-1 | spark 16:47:52.74
sparkitems-spark-master-1 | spark 16:47:52.75 INFO ==> ** Starting Spark set up **
sparkitems-spark-master-1 | spark 16:47:52.76 INFO ==> Generating Spark configuration file...
sparkitems-spark-master-1 | find: '/docker-entrypoint-initdb.d/': No such file or directory
sparkitems-spark-master-1 | spark 16:47:52.77 INFO ==> No custom scripts in /docker-entrypoint-initdb.d

```

Three dockers used for database , Kafka and Spark



All three dockers running successfully

7. Streaming mode Experiment

We have defined a SparkSession, set some configurations, and defined the schema for the incoming Kafka messages. We then read from the Kafka topic, parses the incoming messages with the defined schema, and applies some transformations on the data. Finally, we perform several aggregations on the transformed data and writes the results to the console output in a streaming fashion.

The aggregations include finding the top selling item, the most selling item category, the maximum order value till now, and the hotel with maximum order-value excluding discount and the hotel with maximum orders. The results are output to the console using Spark's structured streaming API.

A. Window

- Window("createdAt", "60 seconds"): This specifies a sliding window of 60 seconds based on the createdAt column. This means that the aggregation will be performed on data that arrived in the last 60 seconds and will be updated every 60 seconds.
- window("createdAt", "60 seconds", "10 seconds"): This specifies a tumbling window of 60 seconds with a sliding interval of 10 seconds. This means that the aggregation

will be performed on data that arrived in the last 60 seconds and will be updated every 10 seconds.

B. Workload

Dataset which contains around 37000 values is passed into Spark for stream processing via Kafka Producer

C. Code

```
max_order_hotel = df \
    .withColumn("totalOrderValue", col("netPrice") -
col("discountPrice")) \
    .groupBy("hotelName") \
    .agg(count("orderId").alias("orderCount"),
sum("totalOrderValue").alias("totalOrderValue"))

max_order_value_hotel_name = max_order_hotel \
    .orderBy(desc("totalOrderValue")) \
    .limit(1) \
    .select("hotelName")

max_order_value_hotel_name_query = max_order_value_hotel_name \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .option("truncate", "false") \
    .start()

max_order_hotel_name = max_order_hotel \
    .orderBy(desc("orderCount")) \
    .limit(1) \
    .select("hotelName")

max_order_hotel_name_query = max_order_hotel_name \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .option("truncate", "false") \
    .start()
```

D.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
o yashaskashyap@Yashass-MacBook-Air ZomaLytics % python -u "/Users/yashaskashyap/Desktop/ZomaLytics/datasource/generator.py"
Published order: {'orderId': 'c92b7170-a935-42e8-931f-7d8caa912276', 'customerId': 'bc101fb6-f3ee-44f9-bc33-fb3a3794d4c5', 'hotelId': 67,
'hotelName': 'Hotel 67', 'items': [{'category': 'Pizza', 'name': 'Hawaiian', 'price': 571, 'quantity': 2}], 'netPrice': 571, 'discountPri
ce': 97.81, 'createdAt': '2023-04-28 22:15:03', 'payment': 'cod'}
Published order: {'orderId': 'd08c255b-16e4-4e33-9d08-8d7ef705301a', 'customerId': 'be19b650-3ca9-4663-8b73-424aed263a9e', 'hotelId': 21,
'hotelName': 'Hotel 21', 'items': [{'category': 'Chinese', 'name': 'Mapo Tofu', 'price': 221, 'quantity': 2}, {'category': 'Burger', 'nam
e': 'Grilled Chicken Burger', 'price': 540, 'quantity': 1}], 'netPrice': 761, 'discountPrice': 165.64, 'createdAt': '2023-04-28 22:15:05',
'payment': 'wallet'}
Published order: {'orderId': 'd222847e-4381-4f91-870f-8f49b0475cb8', 'customerId': '5ee63e04-1e39-486b-926d-05e82d39d89f', 'hotelId': 38,
'hotelName': 'Hotel 38', 'items': [{'category': 'Indian', 'name': 'Paneer Tikka', 'price': 595, 'quantity': 2}, {'category': 'Chinese', '
name': 'Kung Pao Chicken', 'price': 528, 'quantity': 2}], 'netPrice': 1123, 'discountPrice': 11.55, 'createdAt': '2023-04-28 22:15:08', 'p
ayment': 'cod'}
Published order: {'orderId': '54441e78-77d6-4a20-b517-afab14dcde8b', 'customerId': '4d9d068e-4139-43f5-943a-e14b134c24da', 'hotelId': 29,
'hotelName': 'Hotel 29', 'items': [{'category': 'Pizza', 'name': 'Hawaiian', 'price': 844, 'quantity': 3}, {'category': 'Burger', 'name':
'Veggie Burger', 'price': 673, 'quantity': 3}], 'netPrice': 1517, 'discountPrice': 299.92, 'createdAt': '2023-04-28 22:15:13', 'payment':
'cod'}

```

Input to Spark for stream processing from Kafka Producer

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Batch: 1
+-----+-----+-----+
|window|name|total_quantity|
+-----+-----+-----+
|{2023-04-28 22:24:00, 2023-04-28 22:25:00}|Veggie Burger|2|
+-----+-----+-----+

Batch: 1
+-----+-----+
|window|max_net_price|
+-----+-----+
|{2023-04-28 22:23:40, 2023-04-28 22:24:40}|2004.0|
|{2023-04-28 22:23:30, 2023-04-28 22:24:30}|2004.0|
|{2023-04-28 22:23:50, 2023-04-28 22:24:50}|2004.0|
|{2023-04-28 22:24:00, 2023-04-28 22:25:00}|2004.0|
|{2023-04-28 22:24:10, 2023-04-28 22:25:10}|2004.0|
|{2023-04-28 22:23:20, 2023-04-28 22:24:20}|2004.0|
+-----+-----+

Batch: 2
+-----+-----+-----+
|window|category|total_quantity|
+-----+-----+-----+
|{2023-04-28 22:24:00, 2023-04-28 22:25:00}|Burger|6|
+-----+-----+-----+

Batch: 2
+-----+-----+-----+
|window|name|total_quantity|
+-----+-----+-----+
|{2023-04-28 22:24:00, 2023-04-28 22:25:00}|Portobello Mushroom Burger|4|
+-----+-----+-----+

[Stage 23:>(94 + 1) / 200][Stage 24:> (0 + 0) / 1][Stage 27:> (0 + 0) / 1]1]

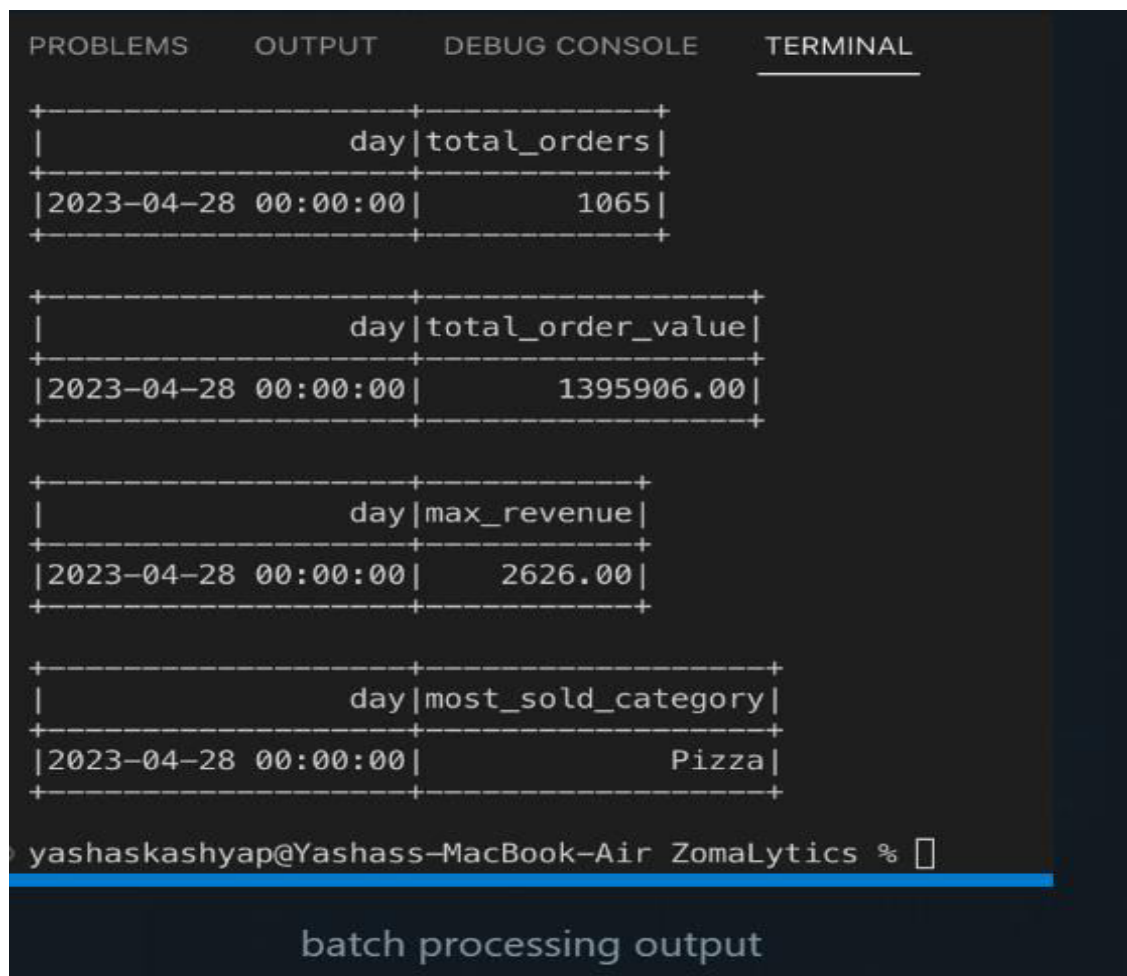
```

Output of stream processing

8. Batch Mode Experiment

- i. Create a SparkSession with necessary configurations for reading data from Kafka and PostgreSQL.
- ii. Reads data from PostgreSQL tables: Orders, Order_Items, and Items and joins them to create a single DataFrame named **joined_df**.
- iii. Groups the **joined_df** DataFrame by day using the **date_trunc** function and aliasing the result as **day**.
- iv. Computes the following metrics for each day:
 - total number of orders per day (**total_orders_per_day**)
 - total order value per day (**total_order_value_per_day**)
 - maximum revenue generated in a day (**max_revenue_per_day**)
 - most sold category in a day (**most_sold_category_per_day**).
- v. Displays the computed metrics using the **show()** method.

The dataset is the same as used in stream processing which contains around 37000 values.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
+-----+
|          day|total_orders|
+-----+
|2023-04-28 00:00:00|      1065|
+-----+

+-----+
|          day|total_order_value|
+-----+
|2023-04-28 00:00:00|    1395906.00|
+-----+

+-----+
|          day|max_revenue|
+-----+
|2023-04-28 00:00:00|     2626.00|
+-----+

+-----+
|          day|most_sold_category|
+-----+
|2023-04-28 00:00:00|          Pizza|
+-----+

yashaskashyap@Yashass-MacBook-Air ZomaLytics %
```

batch processing output

9. Comparison of Streaming & Batch Modes

In our project, batch processing and stream processing can be compared as follows:

Batch processing: In batch processing, data is processed in large volumes at once. This means that the data is collected over a certain period of time and processed in a batch. The processing is typically performed on static data that does not change frequently. Batch processing is useful for scenarios where the data needs to be analyzed in-depth, for example, generating reports or performing complex data analysis.

In our food delivery analytics project, batch processing can be used to generate reports on order history, restaurant performance, and customer behavior. For example, we can analyze the data to determine which restaurants are performing well, which menu items are popular, and which customers place the most orders.

Stream processing: In stream processing, data is processed in real-time as it arrives. This means that data is processed as soon as it is generated, making it ideal for scenarios where data is constantly changing. Stream processing is useful for scenarios where data needs to be acted upon immediately, for example, fraud detection, real-time monitoring, and alerting.

In our food delivery project, stream processing can be used to monitor incoming orders and detect fraud in real-time. For example, we can analyze the orders as they come in and detect any suspicious activity, such as a large number of orders from a single customer or a restaurant that suddenly starts receiving a large number of orders.

10. Conclusion

In conclusion, batch processing and stream processing are two different approaches to processing data, and the choice between them depends on the requirements of the specific use case. In our food delivery analytics project, we used stream processing with Kafka to handle real-time data processing and analysis of incoming orders. Stream processing allowed us to quickly react to new orders and provide timely feedback to the customer, hotel, and delivery personnel. This real-time processing also helped us to identify any issues with the order and address them promptly. However, for other use cases, such as generating reports or analyzing historical data, batch processing may be more appropriate. Ultimately, the choice between batch processing and stream processing depends on the specific requirements of the use case and the desired outcome.