Topic: Real-Time Data Ingestion and Processing Pipeline for Music Events

Abstract

This report presents the development and implementation of a real-time data ingestion and processing pipeline which aimed at identifying trending songs in real-time.

Rabia Salman, Irum Hassan

Contents

1. Introduction

This report presents the development and implementation of a real-time data ingestion and processing pipeline which aimed at identifying trending songs in real-time. The pipeline integrates Docker-based Kafka for data ingestion and PySpark Structured Streaming for computation. The implementation simulates a live "Now Trending" feature commonly found in music streaming platforms. Also, the impact of varying time windows on latency and stability is assessed in this report too.

2. Objective

By the end of this Lab, we aimed to:

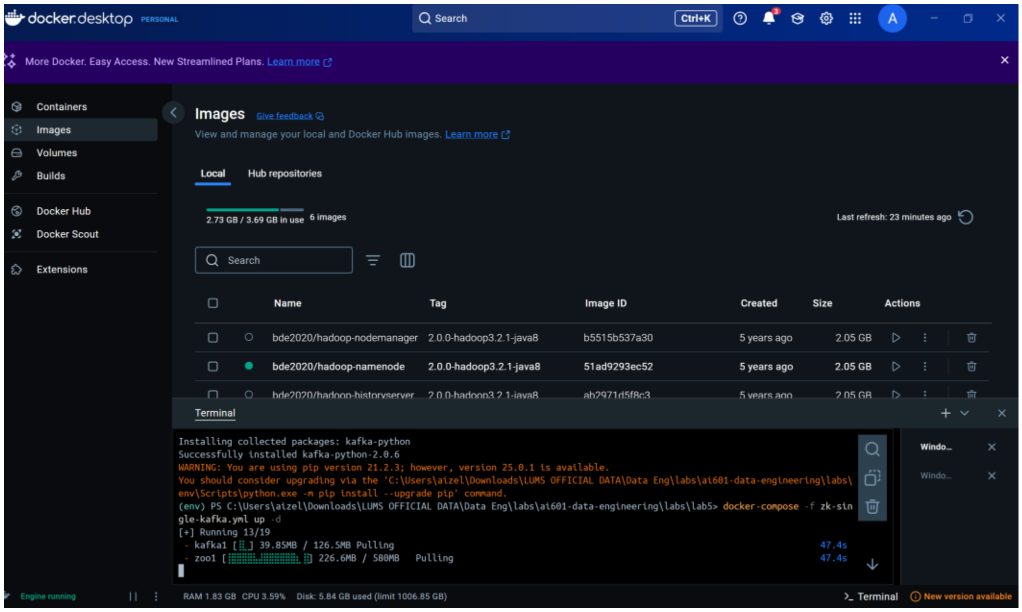
- Install and run Docker to bring up Kafka and ZooKeeper in containers.
- Launch a Python producer script to generate and send mock "music event" data (song plays) into Kafka.
- Implement a PySpark Structured Streaming job that subscribes to the Kafka topic, aggregates data by region and time window and outputs the top songs in real-time.

This Lab explains how streaming data can be processed continuously and also highlights trade-offs in system performance when varying processing parameters.

3. Prerequisites & Setup

To successfully implement this pipeline, the following components were installed and configured:

- **Docker & Docker Compose:**
  - Installed Docker Desktop to provide Docker and Docker Compose.
  - Verified installation using: docker --version and docker-compose --version.
  - Tested installation with: docker run hello-world.
- **Git:**
  - The repository for the project was cloned using:
  - git clone https://github.com/rubabzs/ai601-data-engineering/tree/main/labs/lab5
- **Python 3 Environment:**
  - Created a new virtual environment and installed relevant packages.



4. Kafka Setup

A Kafka topic named music_events was created to ingest music playback data.

➢ Accessing Kafka Broker Container

docker exec -it kafka1 /bin/bash

➢ Verifying Kafka Installation

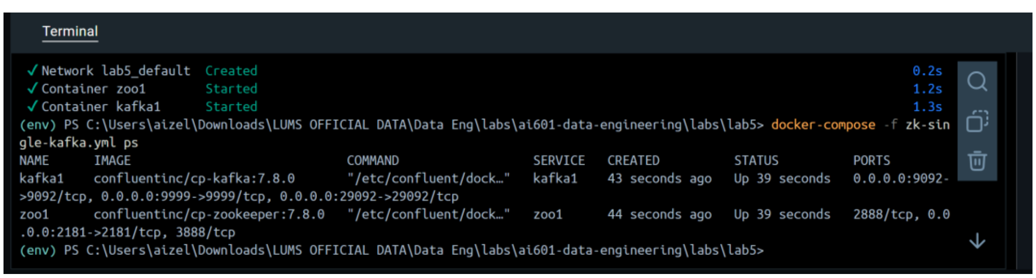
kafka-topics --version

➢ Creating the Kafka Topic

kafka-topics --create --topic music_events --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

➢ Verifying Topic Creation

kafka-topics.sh --list --bootstrap-server localhost:9092



5. Music Event Producer

A Python script (music_producer.py) was created to simulate the generation of random song playback events. This script pushes messages to the music_events topic, structured as follows:

```
{
  "song_id": "<string>",
```

```
    "timestamp": "<string>",
    "region": "<string>",
    "action": "play"
}
```

6. Running PySpark “Now Trending” Script

The PySpark script (now_trending.py) was developed to:

1. Read streaming data from Kafka using PySpark Structured Streaming.
2. Aggregate song play counts by region and time window.
3. Rank songs within each region’s window and select the top 3 songs.
4. Print results to the console every 10 seconds.

The script was executed using:

```
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.5 now_trending.py
```

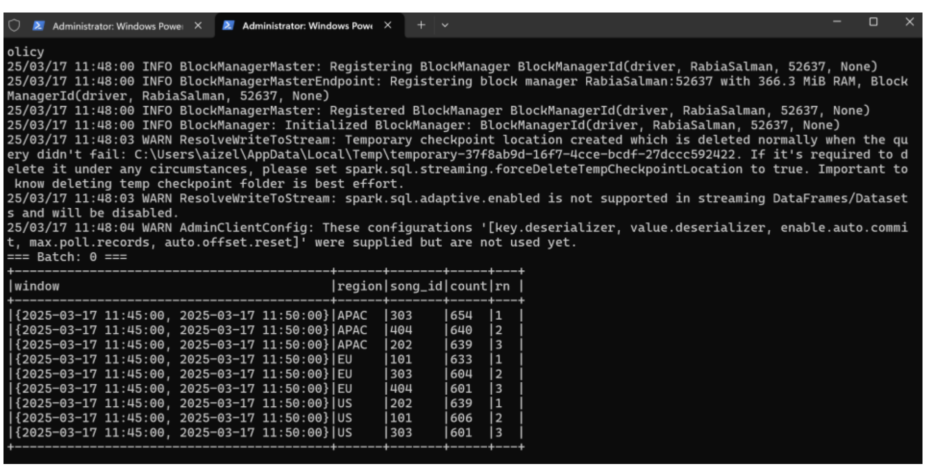
7. Data Processing & Analysis

7.1 Aggregation Logic

- Data is processed in micro-batches (default interval: every few seconds).
- Playback events are grouped by region and song_id using processing-time windows.
- Counts are aggregated within each time window.

7.2 Ranking Songs

- The top 3 songs per region are identified using the row_number() function.
- Results are printed to the console every 10 seconds, reflecting real-time computation.

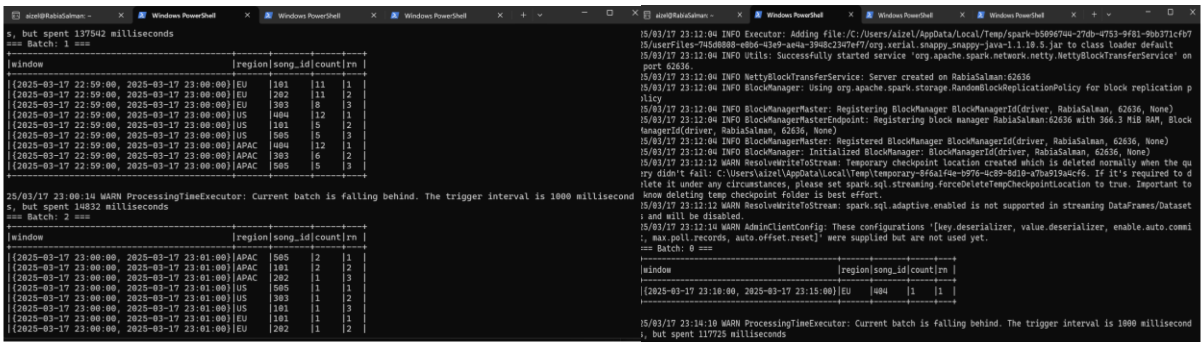


```
Administrator: Windows Powe   Administrator: Windows Powe
olicy
25/03/17 11:48:00 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, RabiaSalman, 52637, None)
25/03/17 11:48:00 INFO BlockManagerMasterEndpoint: Registering block manager RabiaSalman:52637 with 366.3 MiB RAM, Block
ManagerId(driver, RabiaSalman, 52637, None)
25/03/17 11:48:00 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, RabiaSalman, 52637, None)
25/03/17 11:48:00 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, RabiaSalman, 52637, None)
25/03/17 11:48:03 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the qu
ery didn't fail: C:\Users\aizel\AppData\Local\Temp\temporary-37f8ab9d-16f7-4cce-bcdf-27dccc592422. If it's required to d
elete it under any circumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to
 know deleting temp checkpoint folder is best effort.
25/03/17 11:48:03 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Dataset
s and will be disabled.
25/03/17 11:48:04 WARN AdminClientConfig: These configurations '[key.deserializer, value.deserializer, enable.auto.commi
t, max.poll.records, auto.offset.reset]' were supplied but are not used yet.
=== Batch: 0 ===
+------------------------------------------+------+-------+-----+---+
|window                                    |region|song_id|count|rn |
+------------------------------------------+------+-------+-----+---+
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|APAC  |303    |654  |1  |
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|APAC  |404    |640  |2  |
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|APAC  |202    |639  |3  |
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|EU    |101    |633  |1  |
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|EU    |303    |604  |2  |
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|EU    |404    |601  |3  |
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|US    |202    |639  |1  |
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|US    |101    |606  |2  |
|{2025-03-17 11:45:00, 2025-03-17 11:50:00}|US    |303    |601  |3  |
+------------------------------------------+------+-------+-----+---+
```

8. Extension: Varying the Time Window

To assess the trade-off between latency and stability, we altered the time window sizes from the baseline of 5 minutes to:

- **1-Minute Window:** High responsiveness, frequent updates, unstable results.
- **5-Minute Window:** High stability, less frequent updates, slower trend detection.



9. Comparison & Observations

Aspect	**1-Minute Window**	**5-Minute Window (Baseline)**	**10-Minute Window**
Responsiveness	High	Moderate	Low
Stability	Low	Moderate	High
Computational Cost	High	Moderate	Low
Trend Detection	Short-lived	Temporary & Broad	Persistent & Robust
Latency	Low	Moderate	High

1. The 1-minute window offers real-time responsiveness but generates highly fluctuating results.
2. The 5-minute window provides a balanced view, capturing trends accurately with moderate stability.
3. The 10-minute window offers the most stability but is slow to detect emerging trends.

10. Additional Extension: Skip/Like Actions

The task aims to create a real-time streaming pipeline for a music streaming platform, showcasing a live "Now Trending" feature. The main goal is to handle various user interactions, including play, skip, and like actions. To implement this task, a Kafka topic called "music_events" is created and the Event Producer (Python) script is imported. Random user interactions are generated for different songs and sent to the topic. Real-time processing with PySpark Structured Streaming is then performed to process incoming Kafka events, filter only play, skip, and like actions, and aggregate data by region and time window. The Kafka Producer sends events with `song_id`, `timestamp`, `region`, and `action`, sending data continuously with random delays between 0.5 and 2.0 seconds. The Kafka Consumer reads data from Kafka and aggregates it by region and time window.

song_id	total_plays	total_skips	skip_ratio
101	27	16	0.37209302325581395
202	18	27	0.6
505	25	25	0.5
404	34	18	0.34615384615384615
303	29	27	0.48214285714285715

11. Trade-offs in Latency vs. Stability

- **Shorter Windows (1 minute):** Provide high responsiveness but are prone to erratic results.
- **Longer Windows (10 minutes):** Smooth out fluctuations, offering more stable rankings but at the cost of slower trend detection.
- **Balanced Approach (5 minutes):** Offers a reasonable compromise between responsiveness and stability.

Conclusion

The real-time data ingestion and processing pipeline was successfully implemented, demonstrating the ability to capture trending songs in near real-time. The comparison between different time windows provided valuable insights into the trade-offs between latency and stability. Potential improvements include integrating dashboards for real-time visualization and optimizing the pipeline for higher scalability.