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.

Github Link: https://github.com/irumhassan56/ai601-data-engineering

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:

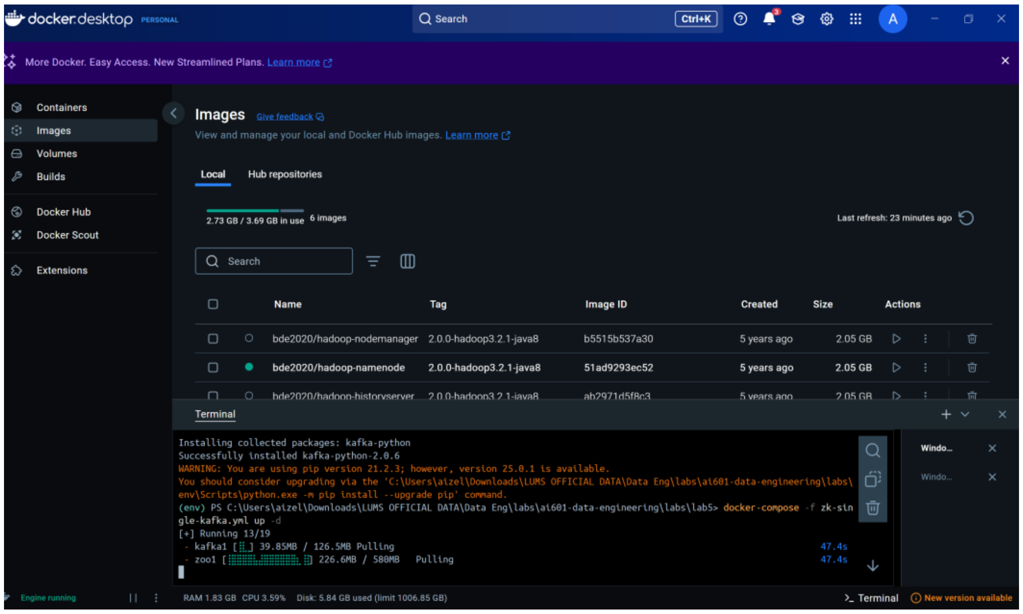
- Install and run Docker to bring up Kafka and ZooKeeper in containers.
- Launch a Python producer script to generate and send mock "music event" data (song plays) into Kafka.
- Implement a PySpark Structured Streaming job that subscribes to the Kafka topic, aggregates data by region and time window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



	Name	Tag	Image ID	Created	Size	Actions
○	bde2020/hadoop-nodemanager	2.0.0-hadoop3.2.1-java8	b5515b537a30	5 years ago	2.05 GB	
●	bde2020/hadoop-namenode	2.0.0-hadoop3.2.1-java8	51ad9293ec52	5 years ago	2.05 GB	
○	bde2020/hadoop-historyserver	2.0.0-hadoop3.2.1-java8	ab2971d5f8c3	5 years ago	2.05 GB	

```
Installing collected packages: kafka-python
Successfully installed kafka-python-2.0.6
WARNING: You are using pip version 21.2.3; however, version 25.0.1 is available.
You should consider upgrading via the 'C:\Users\aizel\Downloads\LUMS OFFICIAL DATA\Data Eng\labs\ai601-data-engineering\labs\
env\Scripts\python.exe -m pip install --upgrade pip' command.
(env) PS C:\Users\aizel\Downloads\LUMS OFFICIAL DATA\Data Eng\labs\ai601-data-engineering\labs\lab5> docker-compose -f zk-sin
gle-kafka.yml up -d
[+] Running 13/19
 - kafka1 [⣿⣀] 39.85MB / 126.5MB Pulling                                                                    47.4s
 - zoo1 [⣿⣿⣿⣿⣿⣀⣿⣿⣿⣿⣿⣿⣀ ⣿] 226.6MB / 580MB   Pulling                                                       47.4s
```

RAM 1.83 GB CPU 3.59% Disk: 5.84 GB used (limit 1006.85 GB)

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

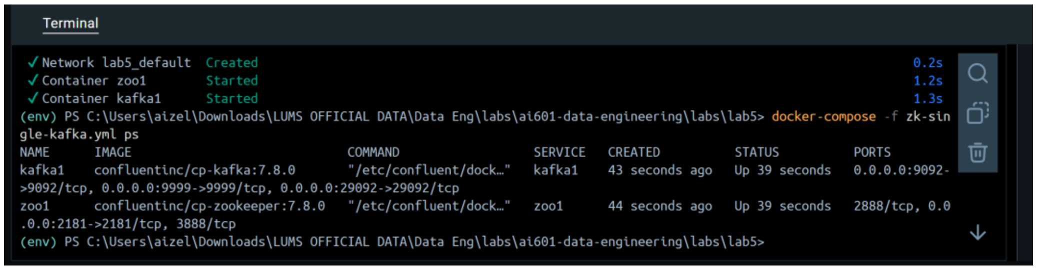
kafka-topics --version

➢ 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



```
 ✓Network lab5_default  Created                                                                          0.2s
 ✓Container zoo1        Started                                                                          1.2s
 ✓Container kafka1      Started                                                                          1.3s
(env) PS C:\Users\aizel\Downloads\LUMS OFFICIAL DATA\Data Eng\labs\ai601-data-engineering\labs\lab5> docker-compose -f zk-sin
gle-kafka.yml ps
NAME     IMAGE                            COMMAND                  SERVICE   CREATED          STATUS          PORTS
kafka1   confluentinc/cp-kafka:7.8.0      "/etc/confluent/dock…"   kafka1    43 seconds ago   Up 39 seconds   0.0.0.0:9092-
>9092/tcp, 0.0.0.0:9999->9999/tcp, 0.0.0.0:29092->29092/tcp
zoo1     confluentinc/cp-zookeeper:7.8.0  "/etc/confluent/dock…"   zoo1      44 seconds ago   Up 39 seconds   2888/tcp, 0.0
.0.0:2181->2181/tcp, 3888/tcp
(env) PS C:\Users\aizel\Downloads\LUMS OFFICIAL DATA\Data Eng\labs\ai601-data-engineering\labs\lab5>
```

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
```

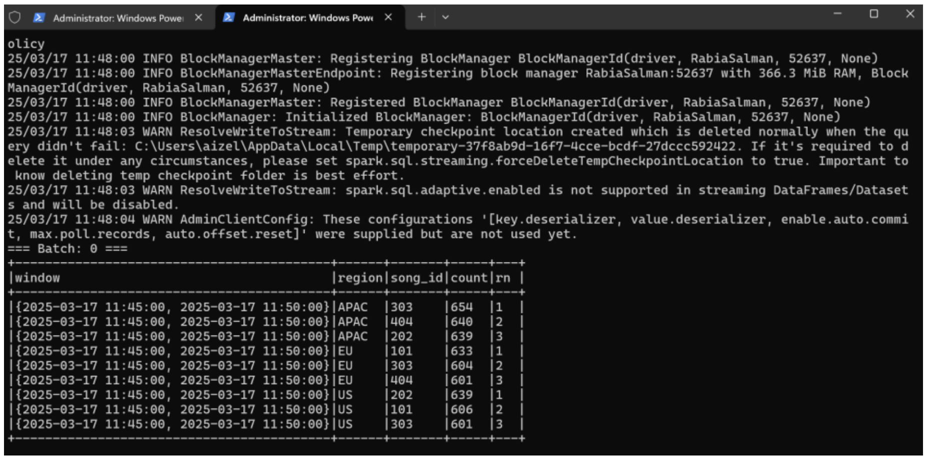
7. Data Processing & Analysis

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.

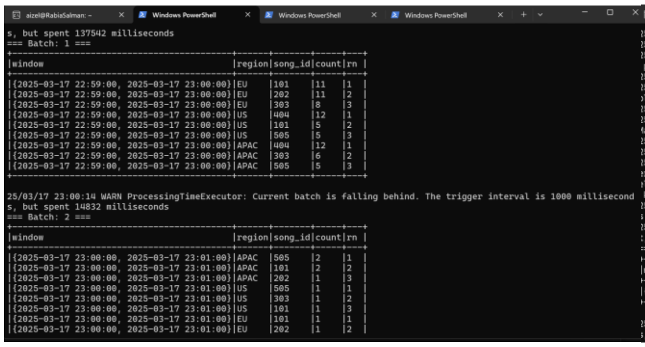


```
olicy
25/03/17 11:48:00 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, RabiaSalman, 52637, None)
25/03/17 11:48:00 INFO BlockManagerMasterEndpoint: Registering block manager RabiaSalman:52637 with 366.3 MiB RAM, Block
ManagerId(driver, RabiaSalman, 52637, None)
25/03/17 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

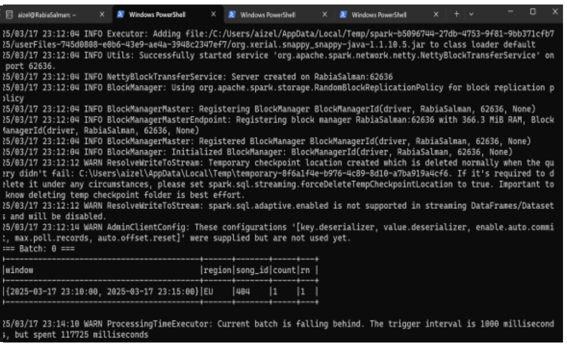
To assess the trade-off between latency and stability, we altered the time window sizes from the baseline of 5 minutes to:

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



```
s, but spent 137542 milliseconds
=== Batch: 1 ===
+------------------------------------------+------+-------+-----+---+
|window                                    |region|song_id|count|rn |
+------------------------------------------+------+-------+-----+---+
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|EU    |101    |11   |1  |
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|EU    |202    |11   |2  |
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|EU    |303    |8    |3  |
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|US    |404    |12   |1  |
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|US    |101    |5    |2  |
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|US    |505    |5    |3  |
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|APAC  |404    |12   |1  |
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|APAC  |303    |6    |2  |
|{2025-03-17 22:59:00, 2025-03-17 23:00:00}|APAC  |505    |5    |3  |
+------------------------------------------+------+-------+-----+---+

25/03/17 23:00:14 WARN ProcessingTimeExecutor: Current batch is falling behind. The trigger interval is 1000 millisecond
s, but spent 14832 milliseconds
=== Batch: 2 ===
+------------------------------------------+------+-------+-----+---+
|window                                    |region|song_id|count|rn |
+------------------------------------------+------+-------+-----+---+
|{2025-03-17 23:00:00, 2025-03-17 23:01:00}|APAC  |505    |2    |1  |
|{2025-03-17 23:00:00, 2025-03-17 23:01:00}|APAC  |101    |2    |2  |
|{2025-03-17 23:00:00, 2025-03-17 23:01:00}|APAC  |202    |1    |3  |
|{2025-03-17 23:00:00, 2025-03-17 23:01:00}|US    |505    |1    |1  |
|{2025-03-17 23:00:00, 2025-03-17 23:01:00}|US    |303    |1    |2  |
|{2025-03-17 23:00:00, 2025-03-17 23:01:00}|US    |101    |1    |3  |
|{2025-03-17 23:00:00, 2025-03-17 23:01:00}|EU    |101    |1    |1  |
|{2025-03-17 23:00:00, 2025-03-17 23:01:00}|EU    |202    |1    |2  |
```



```
25/03/17 23:12:04 INFO Executor: Adding file:/C:/Users/aizel/AppData/Local/Temp/spark-b5096744-27db-4753-9f81-9bb371cfb7
25/userFiles-745d0808-e0b6-43e9-ae4a-3948c2347ef7/org.xerial.snappy_snappy-java-1.1.10.5.jar to class loader default
25/03/17 23:12:04 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on
 port 62636.
25/03/17 23:12:04 INFO NettyBlockTransferService: Server created on RabiaSalman:62636
25/03/17 23:12:04 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication p
olicy
25/03/17 23:12:04 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, RabiaSalman, 62636, None)
25/03/17 23:12:04 INFO BlockManagerMasterEndpoint: Registering block manager RabiaSalman:62636 with 366.3 MiB RAM, Block
ManagerId(driver, RabiaSalman, 62636, None)
25/03/17 23:12:04 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, RabiaSalman, 62636, None)
25/03/17 23:12:04 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, RabiaSalman, 62636, None)
25/03/17 23:12:12 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the qu
ery didn't fail: C:\Users\aizel\AppData\Local\Temp\temporary-8f6a1f4e-b976-4c89-8d10-a7ba919a4cf6. 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 23:12:12 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Dataset
s and will be disabled.
25/03/17 23:12:14 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 23:10:00, 2025-03-17 23:15:00}|EU    |404    |1    |1  |
+------------------------------------------+------+-------+-----+---+

25/03/17 23:14:10 WARN ProcessingTimeExecutor: Current batch is falling behind. The trigger interval is 1000 millisecond
s, but spent 117725 milliseconds
```

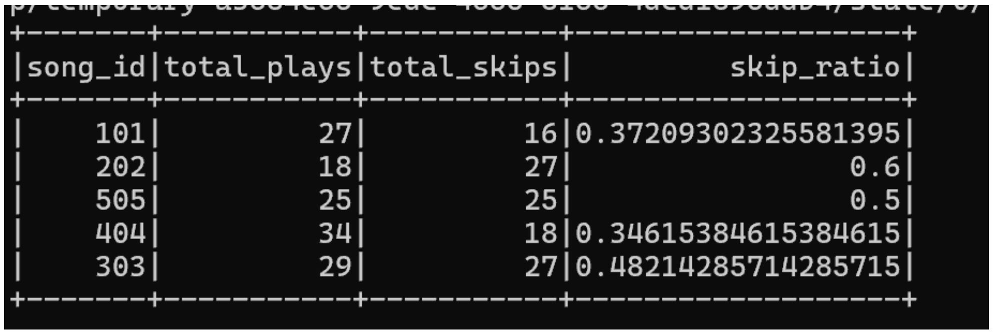
9. Comparison & Observations

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.