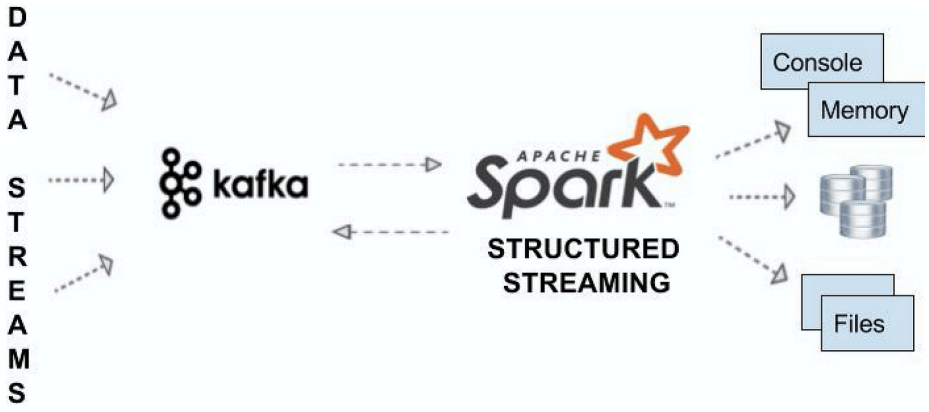

Lab 5: Build a real-time data ingestion and processing pipeline for music events

1. Objective

By the end of this lab, you will:

1. **Install & run Docker** (and Docker Compose) to bring up **Kafka** and **ZooKeeper** in containers.
2. **Launch** a Python **producer script** that sends mock "music event" data (song plays) into Kafka.
3. **Run** a PySpark Structured Streaming job that **subscribes** to the Kafka topic, aggregates data by region and time window, and outputs the top songs in near real-time.

This simulates a **live "Now Trending"** feature in a music streaming application.

2. Prerequisites

1. **Docker** & **Docker Compose** installed on your machine.
   - Check via docker --version and docker-compose --version.
   - If you don't have them, download from Docker website. https://www.docker.com/get-started/
   - Docker compose is included in docker desktop. Just in case for linux machines: sudo apt-get install docker-compose-plugin
   - Test by running: docker run hello-world
2. **Git.** Like our previous labs, clone the lab5 onto your local machine
   - git clone https://github.com/rubabzs/ai601-data-engineering/tree/main/labs/lab5
3. **Python 3** environment and relevant packages.
   - We'd recommend creating a new virtual environment using:

- Navigate to the cloned folder lab5: cd <your-path>/lab5
- Create environment: python -m venv env
- Activate environment: source env/bin/activate
- Install packages: pip install -r requirements.txt

3. Setting Up Kafka via Docker Compose

We will set up kafka now. We are not installing kafka from scratch as it takes some time to configure it and it can get difficult.

1. Launch the docker containers running zookeeper and kafka broker.
    a. docker-compose -f zk-single-kafka.yml up -d
2. Verify that the containers are running. By running the command below, you should see the output as shown in the screenshot.
    a. docker-compose -f zk-single-kafka.yml ps

(Congratulations! Kafka is running!)

4. Create & Verify Kafka Topic

We'll create a **music_events** topic that we will use for ingesting our song plays data.

1. Get inside the docker container running kafka broker:
    a. docker exec -it kafka1 /bin/bash
2. Verify that you are inside the container:
    a. kafka-topics --version

```
[appuser@kafka1 ~]$ kafka-topics --version
6.2.1-ccs (Commit:fa4bec046a2df3a6)
```

3. Create a kafka topic for music_events from inside the docker container:
    a. kafka-topics --create --topic music_events --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
    b. This should give you an output that 'Topic is created'. You can still verify with: kafka-topics.sh --list --bootstrap-server localhost:9092
    c. You can exit the container now with CTRL + D

5. Music Event Producer (Local Python)

It's time to start generating music events!

1. Run the script provided music_producer.py on your local command line:
    a. python music_producer.py
2. You should see a stream of events on your command line like below screenshot:

```
(env) rubabzahra@MacBook-Pro-3 lab5 % python3 music_producer.py
Sent event: {'song_id': 101, 'timestamp': 1741973162.676787, 'region': 'EU', 'action': 'play'}
Sent event: {'song_id': 303, 'timestamp': 1741973163.53583, 'region': 'EU', 'action': 'play'}
Sent event: {'song_id': 404, 'timestamp': 1741973164.636276, 'region': 'EU', 'action': 'play'}
Sent event: {'song_id': 303, 'timestamp': 1741973165.4418821, 'region': 'US', 'action': 'play'}
Sent event: {'song_id': 303, 'timestamp': 1741973166.229949, 'region': 'EU', 'action': 'play'}
Sent event: {'song_id': 404, 'timestamp': 1741973166.836519, 'region': 'EU', 'action': 'play'}
Sent event: {'song_id': 202, 'timestamp': 1741973167.9178722, 'region': 'EU', 'action': 'play'}
Sent event: {'song_id': 101, 'timestamp': 1741973169.723816, 'region': 'US', 'action': 'play'}
Sent event: {'song_id': 303, 'timestamp': 1741973171.1920419, 'region': 'US', 'action': 'play'}
Sent event: {'song_id': 303, 'timestamp': 1741973172.8539798, 'region': 'APAC', 'action': 'play'}
Sent event: {'song_id': 303, 'timestamp': 1741973174.181217, 'region': 'APAC', 'action': 'play'}
```

These events are being generated and published to our kafka topic **music_events** in real-time.

6. Running the PySpark “Now Trending” Script

Now it's time to process the incoming events and generate now trending songs!

1. We will be running the provided pyspark script now_trending.py using kafka-connector and spark-submit. Please note that this script will be run in a new terminal/command line.
    a. spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.5 now_trending.py

```
=== Batch: 1 ===
+------------------------------------------+------+-------+-----+---+
|window                                    |region|song_id|count|rn |
+------------------------------------------+------+-------+-----+---+
|{2025-03-14 22:25:00, 2025-03-14 22:30:00}|EU    |101    |1    |1  |
+------------------------------------------+------+-------+-----+---+

=== Batch: 2 ===
+------------------------------------------+------+-------+-----+---+
|window                                    |region|song_id|count|rn |
+------------------------------------------+------+-------+-----+---+
|{2025-03-14 22:25:00, 2025-03-14 22:30:00}|EU    |303    |2    |1  |
|{2025-03-14 22:25:00, 2025-03-14 22:30:00}|EU    |404    |2    |2  |
|{2025-03-14 22:25:00, 2025-03-14 22:30:00}|EU    |202    |1    |3  |
|{2025-03-14 22:25:00, 2025-03-14 22:30:00}|US    |303    |1    |1  |
+------------------------------------------+------+-------+-----+---+

=== Batch: 3 ===
+------------------------------------------+------+-------+-----+---+
|window                                    |region|song_id|count|rn |
+------------------------------------------+------+-------+-----+---+
|{2025-03-14 22:25:00, 2025-03-14 22:30:00}|APAC  |303    |1    |1  |
|{2025-03-14 22:25:00, 2025-03-14 22:30:00}|US    |303    |2    |1  |
|{2025-03-14 22:25:00, 2025-03-14 22:30:00}|US    |101    |1    |2  |
+------------------------------------------+------+-------+-----+---+
```

Producer is spitting out random (song_id, region, action=play) events to music_events.

Now Trending job:

- Every micro-batch (by default ~every few seconds), we compute the count of each (region, song_id) in the last 5 minutes of **processing time**.
- process_batch(...) ranks songs within each region's window, picks top 3, and prints to console.

Congratulations – you've built a basic streaming pipeline that shows "Now Trending" songs in near real-time, bridging Docker-based Kafka for ingestion with a PySpark streaming job for computation.

7. Extension

Vary the Time Window

- Change the current **5-minute** window to a **shorter** (e.g. 1-minute) or **longer** (e.g. 10-minute) window.
- Observe how often the "Now Trending" results change and discuss trade-offs in latency vs. stability.

Add Skip/Like Actions

- Modify the **producer** to send `"action": "skip"` or `"action": "like"`.
- In the **PySpark** job, filter those actions or compute "skip ratio" to identify songs that might be *unpopular* in real time.

Use Event Time Instead of Processing Time (Optional)

- Convert the `'timestamp'` field to a proper Spark `TimestampType` and add a **watermark** (`.withWatermark("event_time", "5 minutes")`).
- This demonstrates handling **late-arriving** events and out-of-order data.

Write Results to Another Kafka Topic (Optional)

- Instead of console output, publish the top songs to a new topic (`"now_trending_results"`).
- Then run a separate **kafka-console-consumer** or small consumer script to see the real-time output.

Set a Different Micro-Batch Interval

- In `.trigger(processingTime='5 seconds')` or `'1 second'` to see how frequently PySpark updates "Now Trending."
- Measure CPU usage (using top or htop) or see how quickly the results appear, can we debate between performance vs. overhead?

Appendix:

Docker Desktop & WSL 2:

- Download and install the latest version of Docker Desktop for Windows.
- Follow the usual installation instructions to install Docker Desktop. Depending on which version of Windows you are using, Docker Desktop may prompt you to turn on WSL 2 during installation. Read the information displayed on the screen and turn on the WSL 2 feature to continue.
- Start Docker Desktop from the Windows Start menu.
- Navigate to Settings.
- From the General tab, select Use WSL 2 based engine..
- If you have installed Docker Desktop on a system that supports WSL 2, this option is turned on by default.
- Select Apply & Restart.