**Kafka Streaming Pipeline - README**

**Project Overview**

This project sets up a **Kafka-based streaming data pipeline** using **Docker, Kafka and Python**.

The pipeline:

* Ingests real-time login data via a Kafka topic (user-login).
* Filters messages to only process Android logins.
* Publishes processed data to another Kafka topic (processed-logins).

Github Repository: <https://github.com/kevtom404/Kafka_streaming/tree/main>

**Prerequisites**

* **Install Docker & Docker Compose:**

**Windows & macOS:** Install Docker Desktop →

<https://docs.docker.com/desktop/setup/install/windows-install/>

* **Verify Installation:**

>>> docker --version

>>> docker-compose –version

**Setup Kafka & Data Generator with Docker**

* **Step 1: Clone the Repository**

>>> git clone https://github.com/kevtom404/Kafka\_streaming.git

>>> cd Kafka\_streaming

* **Step 2: Start Kafka, Zookeeper & Data Generator**

>>> docker-compose up -d

* **Step 3: Verify Kafka is Running**

Check if Kafka is producing messages:

>>> docker exec -it $(docker ps --filter "name=kafka" --format "{{.ID}}") kafka-console-consumer --bootstrap-server localhost:29092 --topic user-login --from-beginning

If messages appear, Kafka is running successfully.

**Running the Python Kafka Consumer**

* **Step 1: Install Python Dependencies**

>>> pip install confluent-kafka

* **Step 2: Run the Consumer**

>>> python consumer.py

If everything is working, you should see logs like:

Processing message: {"user\_id": "abc123", "device\_type": "android", "app\_version": "1.0.0", ...}

**Verify Processed Data in Kafka**

* To check if processed messages are being published to processed-logins, run:

>>> docker exec -it $(docker ps --filter "name=kafka" --format "{{.ID}}") kafka-console-consumer --bootstrap-server localhost:29092 --topic processed-logins --from-beginning

**Design Choices, Data Flow and Pipeline Efficiency**

**Design Choices:**

* Used **Docker Compose** to set up Kafka, Zookeeper and the data generator.
* Used **Kafka** for real-time messaging, streaming ability and scalability.
* Implemented a **Kafka Consumer** in Python to process messages and publish filtered results to a new Kafka topic.

**Data Flow:**

1. **Data Generator** produces login events and sends them to the user-login Kafka topic.
2. Then **Kafka Consumer** reads messages from the user-login and filters only android logins.
3. **Processed messages** are published to the processed-logins Kafka topic.
4. **Kafka Console Consumer** can be used to verify messages in both topics.

I have attached screenshots of Docker Desktop app and powershell images for reference at the end of the document.

**Efficiency, Scalability & Fault Tolerance:**

* Kafka is designed to handle **growth** by allowing multiple consumers to read and process messages at the same time.
* **Consumer Groups** can be used to distribute load among multiple consumer instances.
* **Error handling** in the Python consumer code prevents crashes due to malformed messages.
* **Kafka’s replication feature** ensures messages are not lost in case of broker failure.
* Since the consumer doesn't store any data between messages, it can easily be **scaled** by adding more instances.

**Troubleshooting**

Some of the troubleshooting I did while working on this challenge for reference.

**Consumer Group Not Found**

If you see Consumer group 'login-consumer-group' does not exist, restart the consumer:

>>> python consumer.py

**Check Kafka Messages**

If no messages appear in user-login, restart the data generator:

>>> docker-compose restart my-python-producer

**Additional Questions:**

**1. How would you deploy this application in production?**

* Use **Kubernetes** or **Docker Swarm** to manage containerized services. If the requirement is allowing to use public cloud service Azure Kubernetes Service , Amazon EKS ,etc can be used.
* Run multiple Kafka brokers across different nodes to ensure reliability and prevent failures if there is any outages.
* Store logs and monitoring metrics using **Prometheus & Grafana** for better monitoring and analytics.
* When working with various external systems for Apache Kafka we can use Kafka connect which helps in integrating external systems. It enables **data ingestion and export** without writing custom code, using **pre-built connectors**.

**2. What other components would you want to add to make this production ready?**

* **Monitoring & Logging:** Use **Prometheus, Grafana** and other similar services for better logging and monitoring.
* **Security:** Implement **SSL/TLS encryption** and **authentication** for better security of the system.
* **Auto-Scaling:** Use **Kubernetes Horizontal Pod Autoscaler** for dynamic consumer scaling for better scalability on demand.
* **Alerting System:** Use **Prometheus Alertmanager** for real-time incident notifications when there is an error or any issues that needs immediate attention in production.

**3. How can this application scale with a growing dataset?**

* **Increase Kafka partitions** to allow more parallel processing.
* **Deploy multiple consumer instances** in a **consumer group** to distribute workload when there is need in more processing.
* **Use Kafka Streams** for real-time stream processing.
* **Leverage cloud-based Kafka solutions** for managed scaling.
* **Optimize message processing** by using **batch consumption and async processing**.

**Screenshots:**



