PLEASE LEAVE ANY COMMENTS TO IMPROVE THE ASSIGNMENT

IF YOU FIND BETTER EXERCISE ON INTERNET USE THAT & LET US KNOW

🧩 Kafka Project Title: Real-Time Event Streaming System for Ride-Sharing Analytics (Basic)

🎯 Goal

Build a **real-time data streaming pipeline** using **Apache Kafka** to simulate, process, and monitor events in a **ride-sharing application** (like Uber/Lyft).
The system should demonstrate Kafka's end-to-end flow: **data production, streaming, consumption, and visualization.**

🧠 Learning Outcomes

By completing this project, participants will:

1. Understand **Kafka architecture** – brokers, topics, partitions, offsets, consumer groups.
2. Learn to **produce and consume messages** programmatically.
3. Implement **stream processing** (with Kafka Streams or a connector framework).
4. Manage **schema evolution** (Avro or JSON schema) and message serialization.
5. Integrate **real-time dashboards** for monitoring.
6. Use **local or free cloud resources** to run Kafka in a reproducible environment.

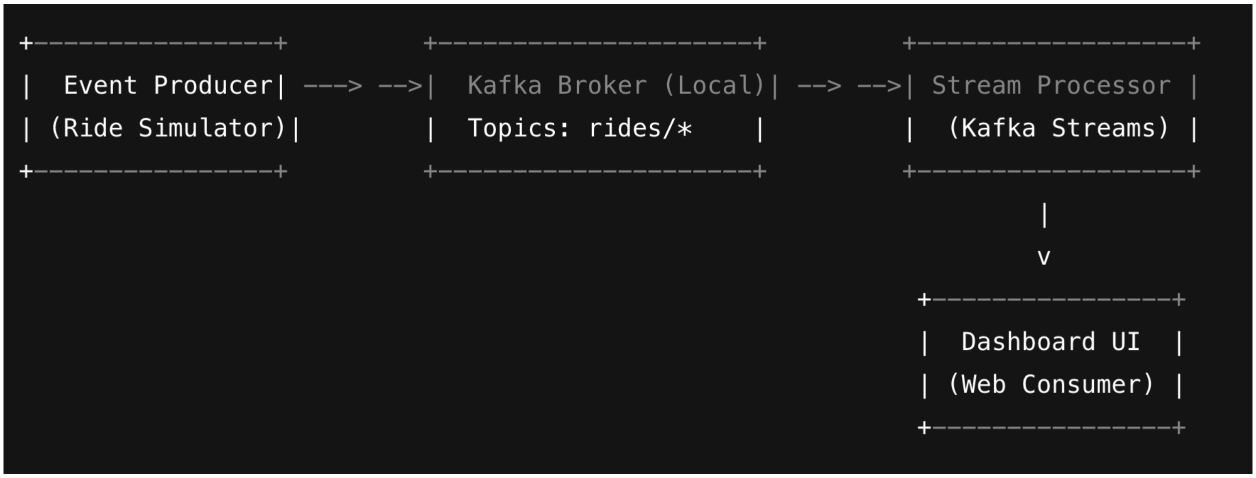
🧱 System Overview

Scenario:
A fictional ride-sharing platform called **“StreamRide”** wants to track ride activity in real time for analytics (driver activity, completed rides, surge pricing alerts, etc.).

The system will have three main components:

1. **Event Producer Service (Backend Simulation)**
   - Simulates real-world ride events and pushes them to Kafka.
   - Event types: `ride_requested`, `ride_started`, `ride_completed`, `driver_location_update`.
2. **Event Processor Service (Stream Processing)**
   - Consumes events from Kafka.
   - Enriches or aggregates data (e.g., calculate average ride time, active drivers per city).
   - Publishes processed/aggregated results to another Kafka topic.
3. **Analytics Dashboard (Consumer)**
   - Subscribes to processed topics.
   - Displays metrics in real time (e.g., number of active rides, rides per city, top driver).

🧩 Technical Architecture



```
+-----------------+        +----------------------+        +-------------------+
|  Event Producer| ---> -->|  Kafka Broker (Local)| --> -->| Stream Processor |
| (Ride Simulator)|        |  Topics: rides/*    |        |  (Kafka Streams) |
+-----------------+        +----------------------+        +-------------------+
                                                                   |
                                                                   v
                                                            +-----------------+
                                                            |  Dashboard UI  |
                                                            | (Web Consumer) |
                                                            +-----------------+
```

⚙️ Required Technologies

Category	Technology
Event Streaming	Apache Kafka (using Docker Compose or local install)
Schema Registry (optional)	Confluent Schema Registry (free local use)
Data Serialization	JSON or Apache Avro
Backend (Producer + Processor)	Java, Python, or Node.js (developer's choice)
Stream Processing	Kafka Streams (Java) or Faust (Python)
Database (optional)	PostgreSQL or SQLite for storing aggregated stats
Visualization	Grafana, Streamlit, or a simple React dashboard
Containerization	Docker Compose
Monitoring (optional)	Prometheus + Grafana or Kafdrop (UI for Kafka topics)

🧪 Detailed Requirements

1. Event Producer Service

- Simulate multiple drivers and riders.

- Emit events at configurable intervals (1–5 seconds).
- Each event should include:
    - `event_type` (e.g., ride_started)
    - `ride_id`
    - `driver_id`
    - `rider_id`
    - `city`
    - `timestamp`
    - Additional metadata (coordinates, fare estimate, etc.)
- Publish events to Kafka topics (e.g., `rides.events`).

Acceptance Criteria:

- At least 4 event types published.
- Events partitioned by `city`.
- JSON or Avro message schema used consistently.

---

2. Stream Processor Service

- Consume messages from `rides.events`.
- Process in real-time to:
    - Compute metrics: active rides, average ride duration, rides per city.
    - Detect anomalies (e.g., long rides > threshold duration).

- Publish computed results to `rides.analytics` topic.

Acceptance Criteria:

- Stream processing runs continuously.
- Metrics topic contains up-to-date aggregations.
- Graceful handling of Kafka restarts and network issues.
- Logs indicate offset commits and processing checkpoints.

---

3. Analytics Dashboard

- Subscribes to the `rides.analytics` topic.
- Displays real-time metrics:
  - Active rides (count)
  - Top 5 active cities
  - Average ride duration
- UI should auto-refresh (WebSocket or polling).

Acceptance Criteria:

- Dashboard updates at least every 5 seconds.
- Metrics accurately reflect current Kafka state.
- Can run locally (no paid services).

🚀 Setup Instructions

1. Clone project repo.
2. Run `docker-compose up` to start:
    - Kafka broker, Zookeeper
    - Schema Registry (optional)
    - Kafdrop or equivalent UI
3. Start producer, processor, and dashboard as separate services (can be run locally or in Docker).
4. Observe streaming data in the dashboard.

Expected local services:

- Kafka Broker: `localhost:9092`
- Schema Registry: `localhost:8081`
- Dashboard: `localhost:3000`
- Kafdrop UI: `localhost:9000`

🧾 Deliverables

1. **Working codebase** runnable via Docker Compose.
2. **README** with setup instructions and architecture diagram.
3. **System Design Document** including:
    - Topic design (names, partitions, replication)
    - Message schema definition
    - Error-handling strategy

  - Scaling considerations

4. **Short demo video** (2–3 min) showing end-to-end flow.