Laboratório LabScore (Kafka)

Configurações na AWS

1. **Criar a instância EC2**
    a. Acesse AWS Console → EC2 → Launch Instance
    b. Nome: labscore-kafka
    c. AMI: Debian
    d. Tipo de instância: **t3.small** (2 vCPU, 2 GB RAM — suficiente para Kafka em laboratório)
    e. Security group:
        - 22 (SSH)
        - 9092 (Kafka)

2. **Instalar o docker**

```
sudo apt-get update
sudo apt-get install -y ca-certificates curl gnupg lsb-release


sudo install -m 0755 -d /etc/apt/keyrings


curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg


echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null


sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin


sudo systemctl enable --now docker


docker run --rm hello-world
docker compose version

sudo usermod -aG docker admin
```

Importante: **fazer logout e login** para aplicar o grupo.

3. **criar o diretório do kafka**

```
mkdir kafka-lab
cd kafka-lab
```

4. **nano docker-compose.yml**

```
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.5.0
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-kafka:7.5.0
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://PUBLIC_IP:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "true"
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
```

Importante: Substituir o **PUBLIC IP** pelo ip da instância.

5. **Subir o kafka e zookeeper**

```
docker compose up -d
```

Configurações na máquina local

1. Criar um ambiente virtual UV
2. Adicionar o kafka-python nesse ambiente

Producer

Crie o arquivo gameSimulator.py e adicione o código abaixo:

```
from kafka import KafkaProducer
import json, time, random

# Configurações
BROKER = "localhost:9092"
TOPICO = "scoreGame"
TEAMS = ["TeamA", "TeamB"]
MAX_SCORE = 25
```

```
# Kafka Producer
producer = KafkaProducer(
    bootstrap_servers=BROKER,
    value_serializer=lambda v: json.dumps(v).encode("utf-8"),
    acks=1
)

def periodo_finalizado(score: dict):
    return abs(score["TeamA"] - score["TeamB"]) >= 2

def criar_evento(event_id: int, team: str, score: dict, event_type: str):
    return {
        "id": event_id,
        "event": event_type,
        "team": team,
        "period": 1,
        "score": score.copy(),
        "timestamp": time.time()
    }

def run():
    print("Producer iniciado...")
    score = {"TeamA": 0, "TeamB": 0}
    event_id = 0

    while True:
        scoring_team = random.choice(TEAMS)
        score[scoring_team] += 1
        event_id += 1

        evento = criar_evento(event_id, scoring_team, score, "goal")

        print("Enviando:", evento)
        producer.send(TOPICO, evento)

        # Verifica condição de encerramento
        if score[scoring_team] >= 25 and periodo_finalizado(score):
            evento['id'] += 1
            evento['event'] = "finished"
            print("Enviando:", evento)
            producer.send(TOPICO, evento)
            break

        time.sleep(1)

    producer.close()
    print("Producer finalizado.")

run()
```

Consumer

Agora que você já possui o producer enviando eventos da partida para o Kafka, vamos criar o consumidor responsável por ler e exibir o placar em tempo real.

Abaixo está um modelo básico do scoreBoard.py.
Ele contém apenas o essencial: a conexão com o Kafka e a inscrição no tópico.

Seu papel é completar esse arquivo adicionando a lógica de leitura e exibição dos eventos.

scoreBoard.py

```
from kafka import KafkaConsumer
import json

IP_AWS = "PUBLIC_IP"
TOPICO = "scoreGame"

def run():
    consumer = KafkaConsumer(
        TOPICO,
        bootstrap_servers=f"{IP_AWS}:9092",
        group_id="group1",
        value_deserializer=lambda m: json.loads(m.decode("utf-8")),
        enable_auto_commit=True,           # Commit automático
        auto_commit_interval_ms=5000,      # Commit a cada 5 segundos
        auto_offset_reset='earliest'       # Lê desde o início se for o primeiro consumo
    )

    # TODO: Ler mensagens em loop
    # TODO: Exibir o placar conforme chegam os eventos
    # TODO: Parar quando receber um evento com "event" == "finished"

    consumer.close()

run()
```

O que você deve implementar

1. Um loop que percorre as mensagens do Kafka, use:

   ```
   for msg in consumer:
   ```

2. O seu consumer deve produzir algo semelhante ao seguinte:

   ```
   Aguardando eventos...
   TeamA       TeamB
                0 - 1
   1 - 1
   2 - 1
   3 - 1
                3 - 2
   ```

```
4 - 2
5 - 2

…..

                15 - 25
Partida finalizada!
```

3. Aplicar uma condição de parada quando o evento for “finished”

At Most Once — Garantia: “No máximo uma vez”

Neste exercício, você irá modificar o scoreBoard.py para simular o comportamento At Most Once, que acontece quando o consumidor confirma (commita) o offset antes de processar a mensagem.
Se ocorrer um erro logo após o commit, a mensagem é perdida para sempre, mostrando exatamente a fragilidade desse modo de entrega.

Abaixo estão as alterações necessárias no arquivo scoreBoard.py.

1. Desabilitar o commit automático
   No KafkaConsumer(), defina:

   - enable_auto_commit=False
   - Remova auto_commit_interval_ms

   Isso permite que **você** controle exatamente quando o offset é confirmado.

2. Fazer o commit manual antes de processar a mensagem
   Logo no início do loop, assim que a mensagem chegar, chame:

   ```
   consumer.commit()
   ```

   Esse commit antecipado é o que recria a garantia **at most once**, pois a mensagem será considerada “lida” mesmo que o processamento falhe depois.

3. Simular uma falha proposital — também antes de imprimir
   Use o campo id produzido pelo simulador.
   Quando id == 5, sua aplicação deve interromper imediatamente a execução, sem processar a mensagem.

4. Processar normalmente as demais mensagens
   Depois dessas etapas, você pode continuar imprimindo o placar normalmente até chegar ao evento “finished”.

At Least Once — Garantia: “Pelo menos uma vez”

Nesta etapa vamos demonstrar o comportamento At Least Once, onde o Kafka garante que nenhuma mensagem será perdida, mesmo em caso de falhas.
Por outro lado, mensagens podem ser processadas mais de uma vez, caso o consumer falhe antes de realizar o commit do offset.

No seu scoreBoard.py, as alterações são simples:

1. Mantenha o commit automático desativado, assim como no exemplo anterior.

2. Inverta a ordem do processamento e do commit.
    - Primeiro processe (exiba o placar)
    - Depois simule a falha
    - Somente então faça o commit (se não houver falha)

Essa inversão é exatamente o que gera duplicação:
se o consumer cair após processar, mas antes de commitar, a mesma mensagem será lida novamente quando o programa reiniciar.

Exactly Once — “Nem duplicada, nem perdida”

Nesta etapa do laboratório vamos demonstrar o comportamento do Exactly Once, Para isso, utilizamos os recursos de transações do Kafka, que permitem que o producer envie mensagens de forma atômica.

O que você precisa alterar para suportar Exactly Once

1. **Producer**
    - atualize o KafkaProducer:
        - Adicione transactional_id="algum_id_unico"
        - Mantenha acks="all"
    - Inicie as transações com producer.init_transactions() no início do programa
    - Antes de cada envio:

```
producer.begin_transaction()

#Manter o trecho do código que envia os eventos normalmente

producer.commit_transaction()
```

    - Em caso de erro:

```
producer.abort_transaction()
```

2. **Consumer**
    - Manter o enable_auto_commit=False
    - O consumer não deve chamar consumer.commit()