TP Kafka avec Spring Boot - Data Streaming pour Débutants

Objectifs pédagogiques

Ce TP vous permettra de :

- Comprendre le pattern **Producer** → **Topic** → **Consumer**
- Maîtriser les concepts d'**offset**, **partition**, et **message keying**
- Mesurer l'écart entre **event time** et **processing time**
- Pratiquer le découplage asynchrone avec Kafka
- Être prêt à expliquer ces concepts en entretien technique

Architecture du projet

```
kafka-streaming-tp/
├── docker-compose.yml
├── kafka-producer/
│   ├── Dockerfile
│   ├── pom.xml
│   └── src/main/java/com/example/producer/
│       ├── KafkaProducerApplication.java
│       ├── model/SensorData.java
│       ├── service/SensorDataService.java
│       └── config/KafkaProducerConfig.java
├── kafka-consumer/
│   ├── Dockerfile
│   ├── pom.xml
│   └── src/main/java/com/example/consumer/
│       ├── KafkaConsumerApplication.java
│       ├── model/SensorData.java
│       ├── service/SensorDataConsumer.java
│       └── config/KafkaConsumerConfig.java
└── README.md
```

Étape 1 : Configuration Docker et Kafka

docker-compose.yml

yaml

```yaml
version: '3.8'

services:
  # Zookeeper - Requis pour Kafka
  zookeeper:
    image: confluentinc/cp-zookeeper:7.4.0
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  # Kafka Broker
  kafka:
    image: confluentinc/cp-kafka:7.4.0
    hostname: kafka
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock

  # Kafka UI (optionnel, pour visualiser les topics)
  kafka-ui:
    image: provectuslabs/kafka-ui:latest
    container_name: kafka-ui
    depends_on:
      - kafka
    ports:
      - "8080:8080"
    environment:
      KAFKA_CLUSTERS_0_NAME: local
      KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: kafka:29092

  # Application Producer
```

```
  kafka-producer:
    build: ./kafka-producer
    container_name: kafka-producer
    depends_on:
      - kafka
    environment:
      KAFKA_BOOTSTRAP_SERVERS: kafka:29092
    restart: unless-stopped

  # Application Consumer
  kafka-consumer:
    build: ./kafka-consumer
    container_name: kafka-consumer
    depends_on:
      - kafka
    environment:
      KAFKA_BOOTSTRAP_SERVERS: kafka:29092
    restart: unless-stopped
```

Étape 2 : Application Producer

kafka-producer/pom.xml

xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.5</version>
        <relativePath/>
    </parent>

    <groupId>com.example</groupId>
    <artifactId>kafka-producer</artifactId>
    <version>1.0.0</version>
    <name>kafka-producer</name>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

kafka-producer/Dockerfile

```
dockerfile

FROM openjdk:17-jdk-slim

WORKDIR /app

COPY target/kafka-producer-1.0.0.jar app.jar

EXPOSE 8081

CMD ["java", "-jar", "app.jar"]
```

kafka-producer/src/main/java/com/example/producer/model/SensorData.java

java

```java
package com.example.producer.model;

import com.fasterxml.jackson.annotation.JsonProperty;
import java.time.Instant;

/**
 * Modèle représentant les données d'un capteur électrique
 */
public class SensorData {

    @JsonProperty("sensorId")
    private String sensorId;

    @JsonProperty("timestamp")
    private Instant timestamp;

    @JsonProperty("value")
    private double value;

    @JsonProperty("unit")
    private String unit;

    // Constructeur par défaut requis pour Jackson
    public SensorData() {}

    public SensorData(String sensorId, Instant timestamp, double value, String unit) {
        this.sensorId = sensorId;
        this.timestamp = timestamp;
        this.value = value;
        this.unit = unit;
    }

    // Getters et Setters
    public String getSensorId() { return sensorId; }
    public void setSensorId(String sensorId) { this.sensorId = sensorId; }

    public Instant getTimestamp() { return timestamp; }
    public void setTimestamp(Instant timestamp) { this.timestamp = timestamp; }

    public double getValue() { return value; }
    public void setValue(double value) { this.value = value; }

    public String getUnit() { return unit; }
    public void setUnit(String unit) { this.unit = unit; }

    @Override
    public String toString() {
```

```
        return "SensorData{" +
                "sensorId='" + sensorId + '\'' +
                ", timestamp=" + timestamp +
                ", value=" + value +
                ", unit='" + unit + '\'' +
                '}';
    }
}
```

kafka-producer/src/main/java/com/example/producer/config/

KafkaProducerConfig.java

java

```java
package com.example.producer.config;

import com.example.producer.model.SensorData;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;
import org.springframework.kafka.support.serializer.JsonSerializer;

import java.util.HashMap;
import java.util.Map;

@Configuration
public class KafkaProducerConfig {

    @Value("${spring.kafka.bootstrap-servers:localhost:9092}")
    private String bootstrapServers;

    /**
     * Configuration du producer Kafka
     * Key = String (sensorId pour le partitioning)
     * Value = SensorData (sérialisé en JSON)
     */
    @Bean
    public ProducerFactory<String, SensorData> producerFactory() {
        Map<String, Object> configProps = new HashMap<>();

        // Adresse du broker Kafka
        configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);

        // Sérializer pour la clé (String)
        configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);

        // Sérializer pour la valeur (JSON)
        configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);

        // Configuration pour la fiabilité
        configProps.put(ProducerConfig.ACKS_CONFIG, "all"); // Attendre confirmation de toutes les répliques
        configProps.put(ProducerConfig.RETRIES_CONFIG, 3);
        configProps.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);
        configProps.put(ProducerConfig.LINGER_MS_CONFIG, 1);
        configProps.put(ProducerConfig.BUFFER_MEMORY_CONFIG, 33554432);

        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, SensorData> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

```
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, SensorData> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

kafka-producer/src/main/java/com/example/producer/service/SensorDataService.java

java

```java
package com.example.producer.service;

import com.example.producer.model.SensorData;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.support.SendResult;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;

import java.time.Instant;
import java.util.List;
import java.util.Random;
import java.util.concurrent.CompletableFuture;

@Service
public class SensorDataService {

    private static final Logger logger = LoggerFactory.getLogger(SensorDataService.class);
    private static final String TOPIC_NAME = "sensor-data";

    private final KafkaTemplate<String, SensorData> kafkaTemplate;
    private final Random random = new Random();

    // Simulation de 3 capteurs différents
    private final List<String> sensorIds = List.of("SENSOR_001", "SENSOR_002", "SENSOR_003");

    public SensorDataService(KafkaTemplate<String, SensorData> kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }

    /**
     * Envoie des données de capteur toutes les 2 secondes
     * La clé Kafka = sensorId permet de garantir l'ordre des messages par capteur
     */
    @Scheduled(fixedRate = 2000)
    public void sendSensorData() {
        // Sélection aléatoire d'un capteur
        String sensorId = sensorIds.get(random.nextInt(sensorIds.size()));

        // Génération d'une valeur électrique réaliste (en volts)
        double voltage = 220.0 + (random.nextGaussian() * 10); // Voltage autour de 220V

        // Création du message avec timestamp actuel
        SensorData sensorData = new SensorData(
            sensorId,
            Instant.now(), // Event time = moment de création
```

```
            voltage,
            "V"
        );

        // Envoi asynchrone vers Kafka
        // La clé (sensorId) détermine la partition
        CompletableFuture<SendResult<String, SensorData>> future =
            kafkaTemplate.send(TOPIC_NAME, sensorId, sensorData);

        // Callback pour traiter le succès/échec
        future.whenComplete((result, exception) -> {
            if (exception == null) {
                logger.info("✅ Message envoyé avec succès - Capteur: {}, Valeur: {:.2f}V, " +
                        "Partition: {}, Offset: {}, Event time: {}",
                        sensorId, voltage,
                        result.getRecordMetadata().partition(),
                        result.getRecordMetadata().offset(),
                        sensorData.getTimestamp());
            } else {
                logger.error("❌ Échec envoi message - Capteur: {}, Erreur: {}",
                        sensorId, exception.getMessage());
            }
        });
    }
}
```

kafka-producer/src/main/java/com/example/producer/KafkaProducerApplication.java

java

```
package com.example.producer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
public class KafkaProducerApplication {

    public static void main(String[] args) {
        SpringApplication.run(KafkaProducerApplication.class, args);
    }
}
```

kafka-producer/src/main/resources/application.yml

yaml

```yaml
spring:
  kafka:
    bootstrap-servers: ${KAFKA_BOOTSTRAP_SERVERS:localhost:9092}
  application:
    name: kafka-producer

server:
  port: 8081

logging:
  level:
    com.example.producer: INFO
    org.springframework.kafka: INFO
```

Étape 3 : Application Consumer

kafka-consumer/pom.xml

xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.5</version>
        <relativePath/>
    </parent>

    <groupId>com.example</groupId>
    <artifactId>kafka-consumer</artifactId>
    <version>1.0.0</version>
    <name>kafka-consumer</name>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

kafka-consumer/Dockerfile

```
dockerfile

FROM openjdk:17-jdk-slim

WORKDIR /app

COPY target/kafka-consumer-1.0.0.jar app.jar

EXPOSE 8082

CMD ["java", "-jar", "app.jar"]
```

kafka-consumer/src/main/java/com/example/consumer/model/SensorData.java

java

```java
package com.example.consumer.model;

import com.fasterxml.jackson.annotation.JsonProperty;
import java.time.Instant;

/**
 * Modèle identique au producer pour désérialiser les messages JSON
 */
public class SensorData {

    @JsonProperty("sensorId")
    private String sensorId;

    @JsonProperty("timestamp")
    private Instant timestamp;

    @JsonProperty("value")
    private double value;

    @JsonProperty("unit")
    private String unit;

    // Constructeur par défaut requis pour Jackson
    public SensorData() {}

    public SensorData(String sensorId, Instant timestamp, double value, String unit) {
        this.sensorId = sensorId;
        this.timestamp = timestamp;
        this.value = value;
        this.unit = unit;
    }

    // Getters et Setters
    public String getSensorId() { return sensorId; }
    public void setSensorId(String sensorId) { this.sensorId = sensorId; }

    public Instant getTimestamp() { return timestamp; }
    public void setTimestamp(Instant timestamp) { this.timestamp = timestamp; }

    public double getValue() { return value; }
    public void setValue(double value) { this.value = value; }

    public String getUnit() { return unit; }
    public void setUnit(String unit) { this.unit = unit; }

    @Override
    public String toString() {
```

```
        return "SensorData{" +
                "sensorId='" + sensorId + '\'' +
                ", timestamp=" + timestamp +
                ", value=" + value +
                ", unit='" + unit + '\'' +
                '}';
    }
}
```

kafka-consumer/src/main/java/com/example/consumer/config/

KafkaConsumerConfig.java

```java
package com.example.consumer.config;

import com.example.consumer.model.SensorData;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import org.springframework.kafka.support.serializer.JsonDeserializer;

import java.util.HashMap;
import java.util.Map;

@Configuration
@EnableKafka
public class KafkaConsumerConfig {

    @Value("${spring.kafka.bootstrap-servers:localhost:9092}")
    private String bootstrapServers;

    /**
     * Configuration du consumer Kafka
     * Key = String (sensorId)
     * Value = SensorData (désérialisé depuis JSON)
     */
    @Bean
    public ConsumerFactory<String, SensorData> consumerFactory() {
        Map<String, Object> props = new HashMap<>();

        // Adresse du broker Kafka
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);

        // Identifiant unique du groupe de consumers
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "sensor-consumer-group");

        // Désérializer pour la clé (String)
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);

        // Désérializer pour la valeur (JSON -> SensorData)
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);

        // Configuration du JsonDeserializer
        props.put(JsonDeserializer.TRUSTED_PACKAGES, "*");
```

```
        props.put(JsonDeserializer.VALUE_DEFAULT_TYPE, SensorData.class);

        // Stratégie de lecture des offsets
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        // Commit automatique des offsets
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, true);
        props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, 1000);

        return new DefaultKafkaConsumerFactory<>(props);
    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory<String, SensorData> kafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, SensorData> factory =
            new ConcurrentKafkaListenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }
}
```

kafka-consumer/src/main/java/com/example/consumer/service/

SensorDataConsumer.java

java

```java
package com.example.consumer.service;

import com.example.consumer.model.SensorData;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

import java.time.Duration;
import java.time.Instant;

@Service
public class SensorDataConsumer {

    private static final Logger logger = LoggerFactory.getLogger(SensorDataConsumer.class);

    /**
     * Écoute les messages du topic "sensor-data"
     * Affiche les informations détaillées sur chaque message reçu
     */
    @KafkaListener(topics = "sensor-data", groupId = "sensor-consumer-group")
    public void consumeSensorData(ConsumerRecord<String, SensorData> record) {

        // Processing time = moment où le message est traité
        Instant processingTime = Instant.now();

        // Récupération des données du message
        SensorData sensorData = record.value();
        String sensorId = record.key();

        // Métadonnées Kafka
        int partition = record.partition();
        long offset = record.offset();

        // Calcul de la latence (écart entre event time et processing time)
        Duration latency = Duration.between(sensorData.getTimestamp(), processingTime);

        // Affichage détaillé du message reçu
        logger.info("📥 MESSAGE REÇU - " +
                "Capteur: {} | " +
                "Valeur: {:.2f}{} | " +
                "Partition: {} | " +
                "Offset: {} | " +
                "Event time: {} | " +
                "Processing time: {} | " +
                "Latence: {}ms",
```

```
                sensorId,
                sensorData.getValue(),
                sensorData.getUnit(),
                partition,
                offset,
                sensorData.getTimestamp(),
                processingTime,
                latency.toMillis());

        // Simulation d'un traitement métier (optionnel)
        processBusinessLogic(sensorData);
    }

    /**
     * Simulation d'un traitement métier sur les données du capteur
     */
    private void processBusinessLogic(SensorData sensorData) {
        // Exemple : alertes si tension anormale
        if (sensorData.getValue() < 200 || sensorData.getValue() > 240) {
            logger.warn("⚠ ALERTE - Tension anormale détectée sur {} : {:.2f}V",
                    sensorData.getSensorId(), sensorData.getValue());
        }

        // Simulation d'un traitement plus long
        try {
            Thread.sleep(100); // 100ms de traitement
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

kafka-consumer/src/main/java/com/example/consumer/KafkaConsumerApplication.java

java

```
package com.example.consumer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class KafkaConsumerApplication {

    public static void main(String[] args) {
        SpringApplication.run(KafkaConsumerApplication.class, args);
    }
}
```

kafka-consumer/src/main/resources/application.yml

yaml

```yaml
spring:
  kafka:
    bootstrap-servers: ${KAFKA_BOOTSTRAP_SERVERS:localhost:9092}
  application:
    name: kafka-consumer

server:
  port: 8082

logging:
  level:
    com.example.consumer: INFO
    org.springframework.kafka: INFO
```

Étape 4 : Création du topic avec 2 partitions

create-topic.sh

bash

```bash
#!/bin/bash

# Attendre que Kafka soit disponible
echo "Attente de Kafka..."
sleep 10

# Créer le topic avec 2 partitions
docker exec kafka kafka-topics --create \
  --topic sensor-data \
  --bootstrap-server localhost:9092 \
  --partitions 2 \
  --replication-factor 1

echo "Topic 'sensor-data' créé avec 2 partitions"

# Vérifier la création
docker exec kafka kafka-topics --list --bootstrap-server localhost:9092
```

Étape 5 : Instructions de démarrage

README.md

markdown

Démarrage du TP Kafka

Prérequis
- Docker et Docker Compose installés
- Java 17 et Maven (pour build local)

Étapes de démarrage

1. Build des applications
```bash
# Producer
cd kafka-producer
mvn clean package -DskipTests
cd ..

# Consumer
cd kafka-consumer
mvn clean package -DskipTests
cd ..
```

2. Démarrage de l'infrastructure

bash

```bash
# Démarrer Kafka et Zookeeper
docker-compose up -d zookeeper kafka

# Attendre 30 secondes puis créer le topic
chmod +x create-topic.sh
./create-topic.sh

# Démarrer les applications
docker-compose up -d kafka-producer kafka-consumer
```

3. Observation des logs

bash

```bash
# Producer
docker logs -f kafka-producer

# Consumer
docker logs -f kafka-consumer

# Kafka UI (optionnel)
# Accéder à http://localhost:8080
```

4. Arrêt

bash

docker-compose down

Étape 6 : Points clés à retenir pour l'entretien

Concepts fondamentaux

1. **Producer → Topic → Consumer** : Pattern de découplage asynchrone
2. **Offset** : Position unique d'un message dans une partition
3. **Partition** : Subdivision d'un topic pour la scalabilité
4. **Message Key** : Détermine la partition (même clé = même partition)
5. **Event Time vs Processing Time** : Important pour l'analyse de latence

Choix techniques justifiés

- **1 topic** : Simplicité pédagogique
- **2 partitions** : Démonstration du partitioning sans complexité
- **Message key = sensorId** : Garantit l'ordre par capteur
- **JSON serialization** : Lisible et standard
- **Auto-commit** : Simplifie la gestion des offsets

Questions d'entretien probables

1. **Pourquoi utiliser Kafka ?** → Découplage, scalabilité, durabilité
2. **Rôle des partitions ?** → Parallélisation et ordre garanti par clé
3. **Que se passe si un consumer crash ?** → Reprise depuis le dernier offset committé
4. **Comment garantir l'ordre ?** → Même clé = même partition
5. **Différence event time / processing time ?** → Latence réseau, traitement async

Ce TP vous donne une base solide pour comprendre Kafka et être confiant en entretien !