TP2 BIG Data

TP 2 - Kafka avec Java

Application Java Producer/Consumer

Créez un petit projet Java Maven pour interagir avec Kafka.

1. **Créer un projet Maven**
2. **Ajouter les dépendances pour Kafka** dans `pom.xml`
   https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients

```
<dependencies>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>3.7.0</version>
    </dependency>
</dependencies>
```

Pour les deux questions suivantes, vous pouvez écrire un message basique comme un String dans le topic

3. Créer dans une classe un producer qui écrit dans le topic "etudiants" crée au TP précédent

```
public class Producer {
    public static void main(String[] args) {
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        try (KafkaProducer<String, String> producer = new KafkaProducer<>(props)) {
            ProducerRecord<String, String> record = new ProducerRecord<>
```

```
("etudiants", "Hello Kafka!");
            producer.send(record);
            System.out.println("Message envoyé !");
        }
    }
}
```

Lancer la classe avec un fonction main et vérifier via sh ou via kafka-ui qu'un message existe dans le topic etudiants

4. Créer dans une classe un consumer qui lit dans le topic "etudiants" crée au TP précédent

```
public class Consumer {
    public static void main(String[] args) {
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("group.id", "group-string");
        props.put("key.deserializer", "org.apache.kafka.common.serialization.
StringDeserializer");
        props.put("value.deserializer", "org.apache.kafka.common.serializatio
n.StringDeserializer");
        props.put("auto.offset.reset", "earliest");

        try (KafkaConsumer<String, String> consumer = new KafkaConsumer
<>(props)) {
            consumer.subscribe(List.of("etudiants"));
            while (true) {
                ConsumerRecords<String, String> records = consumer.poll(Durat
ion.ofMillis(100));
                for (ConsumerRecord<String, String> record : records) {
                    System.out.println("Message reçu : " + record.value());
                }
            }
        }
```

```
    }
}
```

Lancer la classe avec un fonction main et vérifier dans les logs que le message crée précédemment est bien consommé.
Pourquoi n'apparait-il pas ?

Par défaut, un consommateur commence à lire les messages qui arrivent après son démarrage. Si le message a été envoyé avant que le consommateur ne soit actif, il est ignoré.

Comment faire pour relire le message à chaque fois ?

- Utiliser `props.put("auto.offset.reset", "earliest")`.
- Changer le `group.id` (si le groupe a déjà "validé" la lecture, il ne relira pas).
- Ou utiliser `consumer.seekToBeginning()`.

5. Crée un nouveau producer et consommateur pour envoyer des objets java byte à byte représentant des étudiants :

```
{
    "firstName": "Jean",
    "lastName": "Dupont",
    "age": 21,
    "engineeringDegree": "IT"
}
```

Note : Pour sérializer bytes à bytes, il faudra créer un sérializer et un déserializer.

```
Deserializer<?> et Serializer<?>
```

```
public class Etudiant implements Serializable {
    public String firstName;
    public String lastName;
    public int age;
    public String engineeringDegree;
```

```
}
public class EtudiantSerializer implements Serializer<Etudiant> {
    @Override
    public byte[] serialize(String topic, Etudiant data) {
        if (data == null) return null;
        byte[] firstName = data.getFirstName().getBytes(StandardCharsets.UTF_8);
        byte[] lastName = data.getLastName().getBytes(StandardCharsets.UTF_8);
        byte[] degree = data.getEngineeringDegree().getBytes(StandardCharsets.UTF_8);

        ByteBuffer buffer = ByteBuffer.allocate(4 + firstName.length + 4 + lastName.length + 4 + 4 + degree.length);
        buffer.putInt(firstName.length);
        buffer.put(firstName);
        buffer.putInt(lastName.length);
        buffer.put(lastName);
        buffer.putInt(data.getAge());
        buffer.putInt(degree.length);
        buffer.put(degree);
        return buffer.array();
    }
}
```

6. Crée un nouveau producer et consommateur pour envoyer des objets JSON représentant des étudiants

Note : Rajouter les dépendances pour pouvoir utiliser la transformation Json de la librairie Jackson

```
<dependency>
    <groupId>tools.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>3.0.3</version>
</dependency>
```

```
public class EtudiantJsonSerializer implements Serializer<Etudiant> {
    private final JsonMapper mapper = new JsonMapper();

    @Override
    public byte[] serialize(String topic, Etudiant data) {
        try {
            return mapper.writeValueAsBytes(data);
        } catch (Exception e) {
            return null;
        }
    }
}

public class EtudiantDeserializer implements Deserializer<Etudiant> {
    @Override
    public Etudiant deserialize(String topic, byte[] data) {
        if (data == null) return null;

        ByteBuffer buffer = ByteBuffer.wrap(data);

        // Lecture du Prénom
        int firstNameLength = buffer.getInt();
        byte[] firstNameBytes = new byte[firstNameLength];
        buffer.get(firstNameBytes);
        String firstName = new String(firstNameBytes, StandardCharsets.UTF_
8);

        // Lecture du Nom
        int lastNameLength = buffer.getInt();
        byte[] lastNameBytes = new byte[lastNameLength];
        buffer.get(lastNameBytes);
        String lastName = new String(lastNameBytes, StandardCharsets.UTF_
8);

        // Lecture de l'âge
        int age = buffer.getInt();

        // Lecture du diplôme
```

```
        int degreeLength = buffer.getInt();
        byte[] degreeBytes = new byte[degreeLength];
        buffer.get(degreeBytes);
        String engineeringDegree = new String(degreeBytes, StandardCharset
s.UTF_8);

        return new Etudiant(firstName, lastName, age, engineeringDegree);
    }
}

public class EtudiantJsonDeserializer implements Deserializer<Etudiant> {
    private final JsonMapper mapper = new JsonMapper();

    @Override
    public Etudiant deserialize(String topic, byte[] data) {
        if (data == null) return null;
        try {
            // Utilisation de Jackson pour transformer les bytes en objet Java
            return mapper.readValue(data, Etudiant.class);
        } catch (IOException e) {
            throw new RuntimeException("Erreur de désérialisation JSON", e);
        }
    }
}
```

7. Créer un second consommateur dans le même groupe. Observez comment les partitions sont réparties.

8. Expérimentez avec les partitions :

- Créez un topic avec 1 partition, puis un autre avec 5 partitions.
- Lancez plusieurs consommateurs et observez la différence.

```
Properties props = new Properties();
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:90
```

```
92");
props.put(ConsumerConfig.GROUP_ID_CONFIG, "mon-groupe-etudiants");
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDes
erializer.class.getName());
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, Etudia
ntJsonDeserializer.class.getName());
props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
```