Utilisation d'Apache Kafka en Java

Utiliser Apache Kafka en Java implique de configurer un client Kafka dans votre application Java pour produire et consommer des messages. Kafka est une plateforme de streaming distribuée, et ses bibliothèques clientes Java rendent l'intégration simple. Ci-dessous, je vais vous guider à travers les étapes de base pour commencer.

Tout d'abord, vous devrez configurer votre environnement. Assurez-vous que Kafka est installé et en cours d'exécution sur votre système ou un serveur. Vous pouvez le télécharger depuis le site officiel d'Apache Kafka et démarrer ZooKeeper et le serveur Kafka en utilisant les scripts fournis. Pour simplifier, je vais supposer que vous exécutez Kafka localement avec les paramètres par défaut (par exemple, `localhost:9092` comme serveur de démarrage).

Ensuite, ajoutez la dépendance du client Kafka à votre projet Java. Si vous utilisez Maven, incluez ceci dans votre `pom.xml` :

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.6.0</version> <!-- Utilisez la dernière version -->
</dependency>
```

Maintenant, écrivons du code. Je vais vous montrer comment créer un producteur et un consommateur simples.

Exemple de Producteur Kafka

Le producteur envoie des messages à un sujet Kafka. Voici un exemple de base :

```
import org.apache.kafka.clients.producer.*;
import java.util.Properties;

public class SimpleProducer {
    public static void main(String[] args) {
        // Configurer les propriétés du producteur
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092"); // Adresse du serveur Kafka
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        // Créer une instance de producteur
        try (Producer<String, String> producer = new KafkaProducer<>(props)) {
```

```
            // Envoyer un message à un sujet appelé "test-topic"
            String topic = "test-topic";
            for (int i = 0; i < 10; i++) {
                String key = "key" + i;
                String value = "Hello, Kafka " + i;
                ProducerRecord<String, String> record = new ProducerRecord<>(topic, key, value);

                producer.send(record, (metadata, exception) -> {
                    if (exception == null) {
                        System.out.println("Message envoyé : " + value + " à la partition " + metadata.partiti
                    } else {
                        exception.printStackTrace();
                    }
                });
            }
        }
    }
}
```

Dans ce code : - `bootstrap.servers` spécifie où Kafka est en cours d'exécution. - Les serialiseurs définissent comment les clés et les valeurs (toutes deux des chaînes ici) sont converties en octets. - `ProducerRecord` représente le message, et `send()` l'envoie de manière asynchrone avec un rappel pour gérer le succès ou l'échec.

Exemple de Consommateur Kafka

Le consommateur s'abonne à un sujet et lit les messages. Voici un exemple :

```
import org.apache.kafka.clients.consumer.*;
import java.util.Collections;
import java.util.Properties;

public class SimpleConsumer {
    public static void main(String[] args) {
        // Configurer les propriétés du consommateur
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("group.id", "test-group"); // ID du groupe de consommateurs
        props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("auto.offset.reset", "earliest"); // Démarrer depuis le début du sujet
```

```
        // Créer une instance de consommateur
        try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props)) {
            // S'abonner au sujet
            consumer.subscribe(Collections.singletonList("test-topic"));

            // Interroger les messages
            while (true) {
                ConsumerRecords<String, String> records = consumer.poll(1000); // Délai d'attente en ms
                for (ConsumerRecord<String, String> record : records) {
                    System.out.printf("Reçu : key=%s, value=%s, partition=%d, offset=%d%n",
                            record.key(), record.value(), record.partition(), record.offset());
                }
            }
        }
    }
}
```

Ici : - `group.id` identifie le groupe de consommateurs pour la coordination. - Les désérialiseurs convertissent les données en octets en chaînes. - `poll()` récupère les messages dans une boucle, et `auto.offset.reset` garantit que vous commencez par le message le plus ancien si aucun décalage n'est engagé.

Exécution du Code

1. **Démarrer Kafka** : Assurez-vous que ZooKeeper et Kafka sont en cours d'exécution (par exemple, `bin/zookeeper-server-start.sh config/zookeeper.properties` et `bin/kafka-server-start.sh config/server.properties` sur les systèmes de type Unix).
2. **Créer un Sujet** : Utilisez l'outil en ligne de commande Kafka pour créer `test-topic` s'il n'existe pas : `bin/kafka-topics.sh --create --topic test-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1`.
3. **Exécuter le Producteur** : Exécutez la classe `SimpleProducer` pour envoyer des messages.
4. **Exécuter le Consommateur** : Exécutez la classe `SimpleConsumer` pour voir les messages consommés.

Conseils

- **Gestion des erreurs** : Ajoutez des blocs try-catch ou des journaux pour le code de production.
- **Configuration** : Ajustez les propriétés comme `acks` (pour la fiabilité du producteur) ou `max.poll.records` (pour la taille du lot du consommateur) en fonction de vos besoins.
- **Bibliothèques** : Pour des applications plus complexes, envisagez d'utiliser Spring Kafka, qui simplifie la configuration et l'utilisation.

C’est une configuration de base pour vous lancer. Faites-le moi savoir si vous souhaitez approfondir des fonctionnalités spécifiques comme le partitionnement, la gestion des erreurs ou l’intégration avec une application plus large !