Usando Apache Kafka en Java

Usar Apache Kafka en Java implica configurar un cliente de Kafka en tu aplicación de Java para producir y consumir mensajes. Kafka es una plataforma de transmisión distribuida, y sus bibliotecas de clientes de Java hacen que sea sencillo integrarse. A continuación, te guiaré a través de los pasos básicos para comenzar.

Primero, necesitarás configurar tu entorno. Asegúrate de tener Kafka instalado y en funcionamiento en tu sistema o en un servidor. Puedes descargarlo desde el sitio web oficial de Apache Kafka y comenzar el servidor de ZooKeeper y Kafka usando los scripts proporcionados. Para simplicidad, asumiré que estás ejecutando Kafka localmente con la configuración predeterminada (por ejemplo, `localhost:9092` como el servidor de arranque).

A continuación, agrega la dependencia del cliente de Kafka a tu proyecto de Java. Si estás usando Maven, incluye esto en tu `pom.xml`:

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.6.0</version> <!-- Usa la versión más reciente -->
</dependency>
```

Ahora, escribamos algo de código. Te mostraré cómo crear un productor y un consumidor simples.

Ejemplo de Productor de Kafka

El productor envía mensajes a un tema de Kafka. Aquí tienes un ejemplo básico:

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

public class SimpleProducer {
    public static void main(String[] args) {
        // Configurar propiedades del productor
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092"); // Dirección del servidor de Kafka
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        // Crear una instancia del productor
        try (Producer<String, String> producer = new KafkaProducer<>(props)) {
            // Enviar un mensaje a un tema llamado "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("Enviado mensaje: " + value + " a la partición " + metadata.partiti
                    } else {
                        exception.printStackTrace();
                    }
                });
            }
        }
    }
}
```

En este código: - `bootstrap.servers` especifica dónde se está ejecutando Kafka. - Los serializadores definen cómo se convierten las claves y los valores (ambos cadenas aquí) a bytes. - `ProducerRecord` representa el mensaje, y `send()` lo despacha de manera asíncrona con una devolución de llamada para manejar el éxito o el fracaso.

Ejemplo de Consumidor de Kafka

El consumidor se suscribe a un tema y lee mensajes. Aquí tienes un ejemplo:

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

public class SimpleConsumer {
    public static void main(String[] args) {
        // Configurar propiedades del consumidor
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("group.id", "test-group"); // ID del grupo de consumidores
        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"); // Comenzar desde el principio del tema
```

```
        // Crear una instancia del consumidor
        try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props)) {
            // Suscribirse al tema
            consumer.subscribe(Collections.singletonList("test-topic"));

            // Esperar mensajes
            while (true) {
                ConsumerRecords<String, String> records = consumer.poll(1000); // Tiempo de espera en ms
                for (ConsumerRecord<String, String> record : records) {
                    System.out.printf("Recibido: key=%s, value=%s, partición=%d, desplazamiento=%d%n",
                            record.key(), record.value(), record.partition(), record.offset());
                }
            }
        }
    }
}
```

Aquí: - `group.id` identifica el grupo de consumidores para la coordinación. - Los deserializadores convierten los datos en bytes de nuevo a cadenas. - `poll()` recupera mensajes en un bucle, y `auto.offset.reset` asegura que comiences desde el mensaje más antiguo si no se ha comprometido ningún desplazamiento.

Ejecutar el Código

1. **Iniciar Kafka**: Asegúrate de que ZooKeeper y Kafka estén en funcionamiento (por ejemplo, `bin/zookeeper-server-start.sh config/zookeeper.properties` y `bin/kafka-server-start.sh config/server.proper` en sistemas similares a Unix).
2. **Crear un Tema**: Usa la herramienta de línea de comandos de Kafka para crear `test-topic` si no existe: `bin/kafka-topics.sh --create --topic test-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1`.
3. **Ejecutar el Productor**: Ejecuta la clase `SimpleProducer` para enviar mensajes.
4. **Ejecutar el Consumidor**: Ejecuta la clase `SimpleConsumer` para ver los mensajes siendo consumidos.

Consejos

- **Manejo de Errores**: Agrega bloques try-catch o registros para el código de producción.
- **Configuración**: Ajusta propiedades como `acks` (para la confiabilidad del productor) o `max.poll.records` (para el tamaño del lote del consumidor) según tus necesidades.
- **Bibliotecas**: Para aplicaciones más complejas, considera usar Spring Kafka, que simplifica la configuración y el uso.

Esta es una configuración básica para que puedas comenzar. ¡Házmelo saber si quieres profundizar en características específicas como particionamiento, manejo de errores o integración con una aplicación más grande!