MQT HomeWork  
  
Was I a little late, yes. And I hope by making some progress for these tasks can make it up,  
  
Balgradean Andrei.  
  
1.Creare Producer Kafka din consola (terminal).  
kafka-console-producer.sh --broker-list localhost:9092 --topic flower  
  
2. Creare Consumer Kafka din consola (terminal). (e de la inceput thanks to from beginning  
  
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic flower --from-beginning

3. Creare ProducerAPI care trimite evenimente sub forma de siruri de cacactere pe Kafka.   
  
import org.apache.kafka.clients.producer.\*;

import org.apache.kafka.common.serialization.StringSerializer;

import java.util.Properties;

public class StringEventProducer {

public static void main(String[] args) {

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("key.serializer", StringSerializer.class.getName());

props.put("value.serializer", StringSerializer.class.getName());

Producer<String, String> producer = new KafkaProducer<>(props);

String topic = "flower";

try {

for (int i = 0; i < 10; i++) {

String event = "Eveniment " + i;

ProducerRecord<String, String> record = new ProducerRecord<>(topic, event);

producer.send(record);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

producer.close();

}

}

}  
  
4. Crearea ConsumerAPI Kafka ptr Producer-ul de mai sus.

import org.apache.kafka.clients.consumer.\*;

import org.apache.kafka.common.serialization.StringDeserializer;

import java.time.Duration;

import java.util.Collections;

import java.util.Properties;

public class StringEventConsumer {

public static void main(String[] args) {

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("group.id", "test-group");

props.put("key.deserializer", StringDeserializer.class.getName());

props.put("value.deserializer", StringDeserializer.class.getName());

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

String topic = "flower";

try {

consumer.subscribe(Collections.singletonList(topic));

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));

for (ConsumerRecord<String, String> record : records) {

System.out.println("Eveniment citit: " + record.value());

}

}

} catch (Exception e) {

e.printStackTrace();

} finally {

consumer.close();

}

}

}

5. Creare ProducerAPI care trimite evenimente sub forma json pe Kafka.   
  
Clasa FlowerEvent:  
  
public class FlowerEvent {

private String name;

private String color;

public FlowerEvent() {

}

public FlowerEvent(String name, String color) {

this.name = name;

this.color = color;

} public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

@Override

public String toString() {

return "FlowerEvent{" +

"name='" + name + '\'' +

", color='" + color + '\'' +

'}';

}

}  
  
Clasa Producer Principala cu JSON  
  
import org.apache.kafka.clients.producer.\*;

import org.apache.kafka.common.serialization.StringSerializer;

import com.fasterxml.jackson.databind.ObjectMapper;

import java.util.Properties;

public class JsonEventProducer {

public static void main(String[] args) {

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("key.serializer", StringSerializer.class.getName());

props.put("value.serializer", StringSerializer.class.getName());

KafkaProducer<String, String> producer = new KafkaProducer<>(props);

String topic = "flower";

try {

ObjectMapper objectMapper = new ObjectMapper();

FlowerEvent flowerEvent = new FlowerEvent("rose", "red");

String jsonEvent = objectMapper.writeValueAsString(flowerEvent);

ProducerRecord<String, String> record = new ProducerRecord<>(topic, jsonEvent);

producer.send(record);

System.out.println("Eveniment trimis cu succes pe topicul " + topic + ": " + jsonEvent);

} catch (Exception e) {

e.printStackTrace();

} finally {

producer.close();

}

}

}  
  
6. Creare ConsumerAPI Kafka ptr Producer-ul de mai sus.  
  
import org.apache.kafka.clients.consumer.\*;

import org.apache.kafka.common.serialization.StringDeserializer;

import java.time.Duration;

import java.util.Collections;

import java.util.Properties;

public class FlowerConsumer {

public static void main(String[] args) {

String bootstrapServers = "localhost:9092";

String groupId = "flower-consumer-group";

String topic = "flower";

Properties properties = new Properties();

properties.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, bootstrapServers);

properties.put(ConsumerConfig.GROUP\_ID\_CONFIG, groupId);

properties.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class.getName());

properties.put(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.class.getName());

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);

consumer.subscribe(Collections.singletonList(topic));

try {

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));

for (ConsumerRecord<String, String> record : records) {

System.out.println("Received message: " + record.value());

}

}

} catch (Exception e) {

e.printStackTrace();

} finally {

consumer.close();

}

}

}

Acestea doua se regasesc in exercitiile de mai sus deoarece am folosit clasa din JAVA

5. Creare ProducerAPI care trimite evenimente sub forma de obiecte definite (clase Java, C#, etc) in API pe Kafka.

6. Creare ConsumerAPI Kafka ptr Producer-ul de mai sus.

7. Creare ProducerAPI si publicare mesaje folosind schema for Kafka.   
  
For this task I was using the Avro to define the schema

import org.apache.avro.Schema;

import org.apache.avro.generic.GenericData;

import org.apache.avro.generic.GenericRecord;

import org.apache.avro.io.DatumWriter;

import org.apache.avro.io.Encoder;

import org.apache.avro.io.EncoderFactory;

import org.apache.avro.specific.SpecificDatumWriter;

import org.apache.kafka.clients.producer.\*;

import org.apache.kafka.common.serialization.StringSerializer;

import io.confluent.kafka.serializers.KafkaAvroSerializer;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import java.util.Properties;

public class ProducerExample {

public static void main(String[] args) {

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("key.serializer", StringSerializer.class.getName());

props.put("value.serializer", KafkaAvroSerializer.class.getName());

props.put("schema.registry.url", "http://localhost:8081");

Producer<String, GenericRecord> producer = new KafkaProducer<>(props);

Schema.Parser parser = new Schema.Parser();

Schema schema = parser.parse("{\"type\":\"record\",\"name\":\"User\",\"fields\":[{\"name\":\"userId\",\"type\":\"int\"},{\"name\":\"username\",\"type\":\"string\"},{\"name\":\"email\",\"type\":\"string\"},{\"name\":\"age\",\"type\":\"int\"}]}");

GenericRecord avroRecord = new GenericData.Record(schema);

avroRecord.put("userId", 1);

avroRecord.put("username", "JohnDoe");

avroRecord.put("email", "johndoe@example.com");

avroRecord.put("age", 30);

ProducerRecord<String, GenericRecord> record = new ProducerRecord<>("flower", avroRecord);

producer.send(record);

producer.close();

}

}  
  
  
8. Creare ConsumerAPI for Kafka si consumare mesaje folosind schema.   
  
  
import org.apache.avro.generic.GenericRecord;

import org.apache.kafka.clients.consumer.\*;

import org.apache.kafka.common.serialization.StringDeserializer;

import org.apache.kafka.common.serialization.StringSerializer;

import io.confluent.kafka.serializers.KafkaAvroDeserializer;

import java.util.Collections;

import java.util.Properties;

public class ConsumerExample {

public static void main(String[] args) {

Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("group.id", "test-group");

props.put("key.deserializer", StringDeserializer.class.getName());

props.put("value.deserializer", KafkaAvroDeserializer.class.getName());

props.put("schema.registry.url", "http://localhost:8081");

props.put("specific.avro.reader", "false");

KafkaConsumer<String, GenericRecord> consumer = new KafkaConsumer<>(props);

consumer.subscribe(Collections.singletonList("flower"));

try {

while (true) {

ConsumerRecords<String, GenericRecord> records = consumer.poll(100);

for (ConsumerRecord<String, GenericRecord> record : records) {

GenericRecord avroRecord = record.value();

System.out.println("Received: " + avroRecord);

}

}

} finally {

consumer.close();

}

}

}

9.  
  
import org.apache.kafka.clients.consumer.\*;

import org.apache.kafka.clients.producer.\*;

import org.apache.kafka.common.serialization.StringDeserializer;

import org.apache.kafka.common.serialization.StringSerializer;

import java.util.Collections;

import java.util.Properties;

public class ConsumerToProducerPipeline {

public static void main(String[] args) {

Properties consumerProps = new Properties();

consumerProps.put("bootstrap.servers", "localhost:9092");

consumerProps.put("group.id", "test-group");

consumerProps.put("key.deserializer", StringDeserializer.class.getName());

consumerProps.put("value.deserializer", StringDeserializer.class.getName());

Properties producerProps = new Properties();

producerProps.put("bootstrap.servers", "localhost:9092");

producerProps.put("key.serializer", StringSerializer.class.getName());

producerProps.put("value.serializer", StringSerializer.class.getName());

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(consumerProps);

KafkaProducer<String, String> producer = new KafkaProducer<>(producerProps);

String consumerTopic = "flower";

String producerTopic = "Shop";

consumer.subscribe(Collections.singletonList(consumerTopic));

try {

while (true) {

ConsumerRecords<String, String> records = consumer.poll(100);

for (ConsumerRecord<String, String> record : records) {

String key = record.key();

String value = record.value();

ProducerRecord<String, String> producerRecord = new ProducerRecord<>(producerTopic, key, value);

producer.send(producerRecord);

}

}

} finally {

consumer.close();

producer.close();

}

}

}