**Создание механизма связи Consumer-а и Producer-а через Kafka:**

[**https://habr.com/ru/articles/742786/**](https://habr.com/ru/articles/742786/)

**https://github.com/mista1984gmail/kafka-article**

**Для основного кода программы:**

1. Consumer:

- над классом пишем или @Service или @Component,

- над методом пишем @KafkaListener (topics=\*\*\* - обязательно!!!, и др.параметры id/containerFactory/properties и т.д. по необходимости).

2. Producer:

- над классом пишем или @Service или @Component,

- в методе реализовываем метод “send” класса “KafkaTemplate”, чаще всего заполняя его названием topic-а, ключом, и значением(информацией которую передаем).

**Для тестирования Consumer-а и Producer-а:**

1. Для Consumer-а и для Producer-а создаем специальные (для тестов) application.yml.
2. для Consumer-а:

- создаем класс/сы с данными для получаемой сущности,

- над классом пишем @Testcontainers и @SpringBootTest,

- создаем и заполняем начальные параметры контейнеров для БД и Kafka из готовых образов Docker-а, при этом над каждым контейнером пишем @Container,

- объединяем и запускаем созданые контейнеры используя “Startables.deepStart()”,

- заполняем динамические свойства нашего интеграционного теста в отдельном методе над которым пишем @DynamicPropertySource\*,

- делаем автозагрузку нашего интерфейса для работы с БД,

- в самом тестовом методе:

а).иммитируем работу Producer-а, для чего формируем данные для отправки, настраиваем ProducerFactory и оборачиваем KafkaTemplate-ом (*для получения дополнительных методов*) после чего отправляем сформированные данные на Kafka,

б). удостовериваемся в том, что данные отправленные Producer-ом на Kafka получены нашим Consumer-ом, после чего с дополнением доп.полями (если это необходимо) переведены в Entity и после правильно записаны в БД.

3. для Producer -а:

- создаем класс с данными для отправляемой сущности,

- над классом пишем @SpringBootTest, @DirtiesContext и @EmbeddedKafka() с параметрами,

- в самом тестовом методе:

а). создаем и отправляем на Kafka сущность,

б). создаем Consumer-а, который должен получить от Kafka отправленную нами(Producer-ом) сущность,

в). сверяем то, что отправили с тем, что получил Consumer

\* - *@DynamicPropertySource — это аннотация уровня метода, которую можно использовать для регистрации динамических свойств, которые будут добавлены в набор PropertySources в среде для ApplicationContext, загруженного для интеграционного теста. Динамические свойства полезны, когда вы заранее не знаете значение свойств — например, если свойства управляются внешним ресурсом, например, для контейнера, управляемого проектом Testcontainers.*

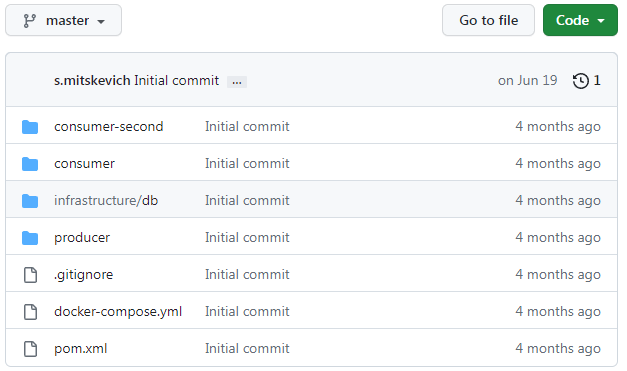
Пример будет очень простой. Допустим у нас будет три микросервиса.

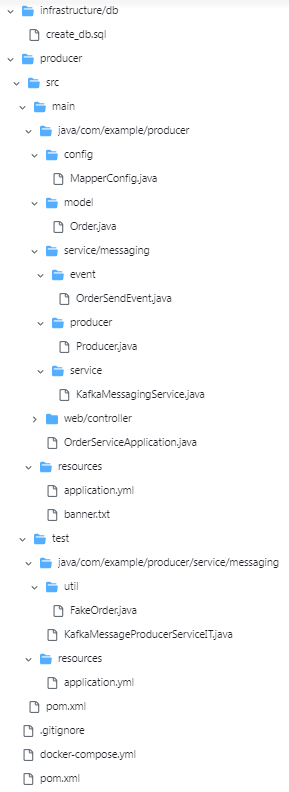
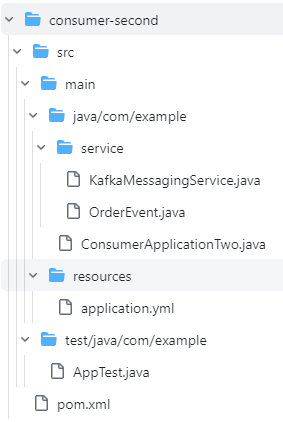
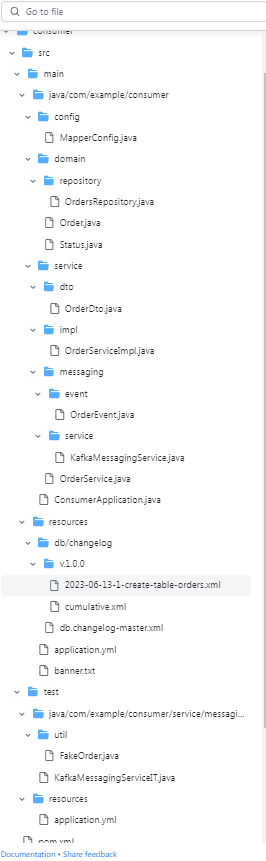
Один микросервис - это продюсер - он будет производить и отправлять сообщения в kafka, в нашем случае это будет Заказ. (стр.15)

Второй микросервис - консьюмер, который будет читать наше сообщение и записывать его в базу данных. (стр.2)

И третий микросервис - также будет читать наше сообщение и просто выводить его в консоль. (стр.11)

Kafka, zookeeper, kafka-ui (для просмотра сообщений в kafka), database (postgres) и pgadmin (для просмотра данных в базе) поднимем с помощью docker.





**CONSUMER**

package com.example.consumer.config;

import org.modelmapper.ModelMapper;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class **MapperConfig** {

@Bean

public ModelMapper modelMapper() {

return new ModelMapper();

}

}

package com.example.consumer.domain;

public enum Status {

IN\_PROGRESS, CANCELED, APPROVED

}

package com.example.consumer.domain;

import jakarta.persistence.\*;

import lombok.Builder;

import java.math.BigDecimal;

import java.time.LocalDateTime;

import java.util.Objects;

@Entity

@Table(name = "orders")

@Builder

public class **Order** {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name="id",

updatable = false)

private Long id;

@Column(name="product\_name",

nullable = false)

private String productName;

@Column(name="bar\_code",

nullable = false)

private String barCode;

@Column(name="quantity",

nullable = false)

private int quantity;

@Column(name="price")

private BigDecimal price;

@Column(name="amount")

private BigDecimal amount;

@Column(name="order\_date",

nullable = false, updatable = false)

private LocalDateTime orderDate;

@Column(name="status",

nullable = false)

@Enumerated(value = EnumType.STRING)

private Status status;

package com.example.consumer.service.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.math.BigDecimal;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class **OrderDto** {

private String productName;

private String barCode;

private int quantity;

private BigDecimal price;

}

// **OrderEvent это нахрен не нужный еще один DTO класс к DTO классу OrderDto, т.е. у него здесь DTO “в квадрате” !!!!!!!!!**

package com.example.consumer.service.messaging.event;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.math.BigDecimal;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class **OrderEvent** {

private String productName;

private String barCode;

private int quantity;

private BigDecimal price;

}

package com.example.consumer.domain.repository;

import com.example.consumer.domain.Order;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface **OrdersRepository** extends JpaRepository<Order, Long> {

}

package com.example.consumer.service;

import com.example.consumer.domain.Order;

import com.example.consumer.service.dto.OrderDto;

public interface OrderService {

Order save(OrderDto clientDto);

}

package com.example.consumer.service.impl;

import com.example.consumer.domain.Order;

import com.example.consumer.domain.Status;

import com.example.consumer.domain.repository.OrdersRepository;

import com.example.consumer.service.OrderService;

import com.example.consumer.service.dto.OrderDto;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;

import java.time.LocalDateTime;

@Slf4j

@Service

@RequiredArgsConstructor

@Transactional(readOnly = true)

public class OrderServiceImpl implements OrderService {

private final OrdersRepository ordersRepository;

@Override

@Transactional

public Order save(OrderDto clientDto) {

//тут в “конвертере” для превода из DTO в Entity не используют ModelMapper т.к. у DTO есть не все поля Entity

Order order= Order.builder()

.productName(clientDto.getProductName())

.barCode(clientDto.getBarCode())

.quantity(clientDto.getQuantity())

.price(clientDto.getPrice())

.amount(new BigDecimal(clientDto.getQuantity()).multiply(clientDto.getPrice()))

.orderDate(LocalDateTime.now())

.status(Status.APPROVED)

.build();

ordersRepository.save(order);

log.info("Save order");

return order;

}

}

package com.example.consumer.service.messaging.service;

import com.example.consumer.service.OrderService;

import com.example.consumer.service.dto.OrderDto;

import com.example.consumer.service.messaging.event.OrderEvent;

import lombok.AllArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.modelmapper.ModelMapper;

import org.springframework.kafka.annotation.KafkaListener;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;

@Slf4j

@Service

@AllArgsConstructor

public class **KafkaMessagingService** {

private static final String topicCreateOrder = "${topic.send-order}";

private static final String kafkaConsumerGroupId = "${spring.kafka.consumer.group-id}";

private final OrderService orderService;

private final ModelMapper modelMapper;

@Transactional

@KafkaListener(topics = topicCreateOrder, groupId = kafkaConsumerGroupId, properties = {"spring.json.value.default.type=com.example.consumer.service.messaging.event.OrderEvent"})

public OrderEvent createOrder(OrderEvent orderEvent) {

log.info("Message consumed {}", orderEvent);

orderService.save(modelMapper.map(orderEvent, OrderDto.class));

return orderEvent;

}

}

package com.example.consumer;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class **ConsumerApplication** {

public static void main(String[] args) {

SpringApplication.run(ConsumerApplication.class, args);

}

}

**application.yml**

server:

port: 8082

spring:

kafka:

// Адрес/а брокера/ов Kafka

bootstrap-servers: localhost:29092

consumer:

group-id: "order-1"

auto-offset-reset: earliest

key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer

properties:

spring:

json:

trusted:

packages: '\*'

datasource:

url: jdbc:postgresql://${DB\_HOST:localhost}:${DB\_PORT:15432}/orders\_database

username: username

password: password

liquibase:

enabled: true

drop-first: false

change-log: classpath:db/changelog/db.changelog-master.xml

default-schema: public

jpa:

show-sql: false

open-in-view: false

hibernate:

ddl-auto: none

properties:

hibernate:

dialect: org.hibernate.dialect.PostgreSQLDialect

topic:

send-order: send-order-event

package com.example.consumer.service.messaging.service.util;

import com.example.consumer.domain.Order;

import com.example.consumer.domain.Status;

import com.example.consumer.service.messaging.event.OrderEvent;

import java.math.BigDecimal;

import java.time.LocalDateTime;

public class **FakeOrder** {

public static OrderEvent getOrderEvent(){

return new OrderEvent(

"pensil",

"0000003",

100,

new BigDecimal(0.99)

);

}

public static Order getOrder(){

return new Order(

1L,

"pensil",

"0000003",

100,

new BigDecimal(0.99),

new BigDecimal(99.0),

LocalDateTime.of(2023,06,16,15,56,00),

Status.APPROVED

);

}

}

package com.example.consumer.service.messaging.service;

import com.example.consumer.domain.Order;

import com.example.consumer.domain.repository.OrdersRepository;

import com.example.consumer.service.messaging.event.OrderEvent;

import com.example.consumer.service.messaging.service.util.FakeOrder;

import org.apache.kafka.clients.producer.ProducerConfig;

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

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

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 org.springframework.test.context.DynamicPropertyRegistry;

import org.springframework.test.context.DynamicPropertySource;

import org.testcontainers.containers.KafkaContainer;

import org.testcontainers.containers.PostgreSQLContainer;

import org.testcontainers.junit.jupiter.Container;

import org.testcontainers.junit.jupiter.Testcontainers;

import org.testcontainers.lifecycle.Startables;

import org.testcontainers.utility.DockerImageName;

import scala.concurrent.duration.Duration;

import java.math.RoundingMode;

import java.util.HashMap;

import java.util.Map;

import java.util.concurrent.TimeUnit;

import java.util.stream.Stream;

import static java.util.concurrent.TimeUnit.SECONDS;

import static org.awaitility.Awaitility.await;

import static org.junit.jupiter.api.Assertions.assertEquals;

@**Test**containers

@SpringBoot**Test**

class **KafkaMessagingServiceIT** {

public static final Long ORDER\_ID = 1L;

public static final String TOPIC\_NAME\_SEND\_ORDER= "send-order-event";

@Container

static PostgreSQLContainer<?> postgreSQLContainer = new PostgreSQLContainer<>("postgres:12")

.withUsername("username")

.withPassword("password")

.withExposedPorts(5432)

.withReuse(true);

@Container

static final KafkaContainer kafkaContainer =

new KafkaContainer(DockerImageName.parse("confluentinc/cp-kafka:6.2.4"))

.withEmbeddedZookeeper()

как в Docker-compose

.withEnv("KAFKA\_LISTENERS", "PLAINTEXT://0.0.0.0:9093 ,BROKER://0.0.0.0:9092")

.withEnv("KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP", "BROKER:PLAINTEXT,PLAINTEXT:PLAINTEXT")

.withEnv("KAFKA\_INTER\_BROKER\_LISTENER\_NAME", "BROKER")

.withEnv("KAFKA\_BROKER\_ID", "1")

.withEnv("KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR", "1")

.withEnv("KAFKA\_OFFSETS\_TOPIC\_NUM\_PARTITIONS", "1")

.withEnv("KAFKA\_TRANSACTION\_STATE\_LOG\_REPLICATION\_FACTOR", "1")

дополнительные настройки

.withEnv("KAFKA\_TRANSACTION\_STATE\_LOG\_MIN\_ISR", "1")

.withEnv("KAFKA\_LOG\_FLUSH\_INTERVAL\_MESSAGES", Long.MAX\_VALUE + "")

.withEnv("KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS", "0");

static {

Startables.deepStart(Stream.of(postgreSQLContainer, kafkaContainer)).join();

}

@DynamicPropertySource

static void overrideProperties(DynamicPropertyRegistry registry) {

registry.add("spring.kafka.bootstrap-servers", kafkaContainer::getBootstrapServers);

registry.add("spring.datasource.url", postgreSQLContainer::getJdbcUrl);

registry.add("spring.datasource.username", postgreSQLContainer::getUsername);

registry.add("spring.datasource.password", postgreSQLContainer::getPassword);

registry.add("spring.datasource.driver-class-name", postgreSQLContainer::getDriverClassName);

}

@Autowired

private OrdersRepository ordersRepository;

@Test

void save\_order() throws InterruptedException {

**//Иммитируем работу Producer-а**

//формируем данные для отправки

String bootstrapServers = kafkaContainer.getBootstrapServers();

OrderEvent orderEvent = FakeOrder.getOrderEvent();

Order order = FakeOrder.getOrder();

//настраиваем ProducerFactory

Map<String, Object> configProps = new HashMap<>();

configProps.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, bootstrapServers);

configProps.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.class);

configProps.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, JsonSerializer.class);

ProducerFactory<String, OrderEvent> producerFactory = new DefaultKafkaProducerFactory<>(configProps);

//оборачиваем KafkaTemplate-ом, для получения дополнительных методов

KafkaTemplate<String, OrderEvent> kafkaTemplate = new KafkaTemplate<>(producerFactory);

//отправляем сформированные данные на Kafka

SECONDS.sleep(5);

kafkaTemplate.send(TOPIC\_NAME\_SEND\_ORDER, orderEvent.getBarCode(), orderEvent);

SECONDS.sleep(5);

//Удостовериваемся в том, что данные отправленные Producer-ом на Kafka получены нашим Consumer-ом, после чего с дополнением доп.полями переведены в Entity и после правильно записаны в БД

Order orderFromDB = ordersRepository.findById(ORDER\_ID).get();

assertEquals(orderFromDB.getId(), ORDER\_ID);

assertEquals(orderFromDB.getProductName(), order.getProductName());

assertEquals(orderFromDB.getBarCode(), order.getBarCode());

assertEquals(orderFromDB.getQuantity(), order.getQuantity());

assertEquals(orderFromDB.getPrice(), order.getPrice().setScale(2, RoundingMode.HALF\_DOWN));

assertEquals(orderFromDB.getAmount(), order.getAmount().setScale(2));

assertEquals(orderFromDB.getOrderDate().getYear(), order.getOrderDate().getYear());

assertEquals(orderFromDB.getStatus(), order.getStatus());

}

}

**application.yml специальный для тестов Consumer-а**

spring:

kafka:

consumer:

group-id: "order-1"

auto-offset-reset: earliest

key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer

properties:

spring:

json:

trusted:

packages: '\*'

liquibase:

enabled: true

drop-first: false

change-log: classpath:db/changelog/db.changelog-master.xml

default-schema: public

topic:

send-order: send-order-event

**pom.xml**

<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">

<parent>

<artifactId>kafka-article</artifactId>

<groupId>com.example</groupId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>consumer</artifactId>

<packaging>jar</packaging>

<name>consumer</name>

<url>http://maven.apache.org</url>

<properties>

</properties>

<dependencies>

<!--JPA-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!--LIQUIBASE-->

<dependency>

<groupId>org.liquibase</groupId>

<artifactId>liquibase-core</artifactId>

</dependency>

<!--DATABASE-->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

</project>

**CONSUMER-SECOND**

package com.example.service;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.math.BigDecimal;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class **OrderEvent** {

private String productName;

private String barCode;

private int quantity;

private BigDecimal price;

}

package com.example.service;

import lombok.AllArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.kafka.annotation.KafkaListener;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;

@Slf4j

@Service

@AllArgsConstructor

public class **KafkaMessagingService** {

private static final String topicCreateOrder = "${topic.send-order}";

private static final String kafkaConsumerGroupId = "${spring.kafka.consumer.group-id}";

@Transactional

@KafkaListener(topics = topicCreateOrder, groupId = kafkaConsumerGroupId, properties = {"spring.json.value.default.type=com.example.service.OrderEvent"})

public OrderEvent printOrder(OrderEvent orderEvent) {

log.info("The product: {} was ordered in quantity: {} and at a price: {}", orderEvent.getProductName(), orderEvent.getQuantity(), orderEvent.getPrice());

log.info("To pay: {}", new BigDecimal(orderEvent.getQuantity()).multiply(orderEvent.getPrice()));

return orderEvent;

}

}

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class **ConsumerApplicationTwo** {

public static void main(String[] args) {

SpringApplication.run(ConsumerApplicationTwo.class, args);

}

}

**application.xml**

server:

port: 8083

spring:

kafka:

bootstrap-servers: localhost:29092

consumer:

group-id: "order-2"

auto-offset-reset: earliest

key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer

properties:

spring:

json:

trusted:

packages: '\*'

topic:

send-order: send-order-event

package com.example;

import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;

// Unit test for simple App.

public class **AppTest** extends TestCase {

// Create the test case

// @param testName name of the test case

public AppTest( String testName )

{ super( testName ); }

/@return the suite of tests being tested

public static Test suite()

{ return new TestSuite( AppTest.class ); }

}

**pom.xml**

<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">

<parent>

<artifactId>kafka-article</artifactId>

<groupId>com.example</groupId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>consumer-second</artifactId>

<packaging>jar</packaging>

<name>consumer-second</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**INFRASTRUCTURE/DB**

**create\_db.sql**

create database orders\_database;

**PRODUCER**

package com.example.producer.config;

import org.modelmapper.ModelMapper;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class **MapperConfig** {

@Bean

public ModelMapper modelMapper() {

return new ModelMapper();

}

}

package com.example.producer.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.math.BigDecimal;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class **Order** {

private String productName;

private String barCode;

private int quantity;

private BigDecimal price;

}

package com.example.producer.service.messaging.event;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import java.math.BigDecimal;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class **OrderSendEvent** {

private String productName;

private String barCode;

private int quantity;

private BigDecimal price;

}

package com.example.producer.service.messaging.producer;

import com.example.producer.model.Order;

import com.example.producer.service.messaging.event.OrderSendEvent;

import com.example.producer.service.messaging.service.KafkaMessagingService;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.modelmapper.ModelMapper;

import org.springframework.stereotype.Component;

@Slf4j

@Component

@RequiredArgsConstructor

public class **Producer** {

private final KafkaMessagingService kafkaMessagingService;

private final ModelMapper modelMapper;

public Order sendOrderEvent(Order order) {

kafkaMessagingService.sendOrder(modelMapper.map(order, OrderSendEvent.class));

log.info("Send order from producer {}", order);

return order;

}

}

package com.example.producer.service.messaging.service;

import com.example.producer.service.messaging.event.OrderSendEvent;

import lombok.RequiredArgsConstructor;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.kafka.core.KafkaTemplate;

import org.springframework.stereotype.Service;

@Service

@RequiredArgsConstructor

public class **KafkaMessagingService** {

@Value("${topic.send-order}")

private String sendClientTopic;

private final KafkaTemplate<String , Object> kafkaTemplate;

public void sendOrder(OrderSendEvent orderSendEvent) {

kafkaTemplate.send(sendClientTopic, orderSendEvent.getBarCode(), orderSendEvent);

}

}

package com.example.producer.web.controller;

import com.example.producer.model.Order;

import com.example.producer.service.messaging.producer.Producer;

import lombok.RequiredArgsConstructor;

import lombok.extern.slf4j.Slf4j;

import org.springframework.http.HttpStatus;

import org.springframework.validation.annotation.Validated;

import org.springframework.web.bind.annotation.\*;

@Slf4j

@Validated

@RestController

@RequiredArgsConstructor

@RequestMapping("/api/v1/orders")

public class **OrderController** {

private final Producer producer;

@PostMapping

@ResponseStatus(HttpStatus.OK)

public Order sendOrder(@RequestBody Order order) {

log.info("Send order to kafka");

producer.sendOrderEvent(order);

return order;

}

}

package com.example.producer;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class **OrderServiceApplication** {

public static void main(String[] args) {

SpringApplication.run(OrderServiceApplication.class, args);

}

}

**application.yml**

server:

port: 8081

spring:

kafka:

bootstrap-servers: localhost:29092

producer:

key-serializer: org.apache.kafka.common.serialization.StringSerializer

value-serializer: org.springframework.kafka.support.serializer.JsonSerializer

properties:

acks: 1

spring:

json:

add:

type:

headers: false

topic:

send-order: send-order-event

package com.example.producer.service.messaging.util;

import com.example.producer.service.messaging.event.OrderSendEvent;

import java.math.BigDecimal;

public class **FakeOrder** {

public static OrderSendEvent getOrderSendEvent(){

return new OrderSendEvent(

"pensil",

"0000003",

100,

new BigDecimal(0.99)

);

}

}

package com.example.producer.service.messaging;

import com.example.producer.service.messaging.event.OrderSendEvent;

import com.example.producer.service.messaging.service.KafkaMessagingService;

import com.example.producer.service.messaging.util.FakeOrder;

import org.apache.kafka.clients.consumer.ConsumerConfig;

import org.apache.kafka.clients.consumer.ConsumerRecords;

import org.apache.kafka.clients.consumer.KafkaConsumer;

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

import org.junit.jupiter.api.Test;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.kafka.support.serializer.JsonDeserializer;

import org.springframework.kafka.test.context.EmbeddedKafka;

import org.springframework.test.annotation.DirtiesContext;

import java.time.Duration;

import java.util.Arrays;

import java.util.Properties;

import static org.junit.jupiter.api.Assertions.assertEquals;

@SpringBoot**Test**

//@DirtiesContext — это аннотация для тестирования Spring . Это указывает на то, что связанный тест или класс изменяет ApplicationContext . Он сообщает среде тестирования закрыть и воссоздать контекст для последующих тестов.

@DirtiesContext

//Spring для Apache Kafka предоставляет удобный способ тестирования проектов с помощью встроенного брокера Apache Kafka. Чтобы использовать эту функцию, аннотируйте тестовый класс с помощью @EmbeddedKafka из модуля spring-kafka-test. Т.е. вместо контейнера для Kafka “@Container static final KafkaContainer kafkaContainer = \*\*\* “ в Consumer-е, в Producer-е используем аннотацию @EmbeddedKafka с настройками.

@EmbeddedKafka(partitions = 1, brokerProperties = { "listeners=PLAINTEXT://localhost:9092", "port=9092" })

public class **KafkaMessageProducerServiceIT** {

public static final String TOPIC\_NAME\_SEND\_CLIENT = "send-order-event";

@Autowired

private KafkaMessagingService kafkaMessagingService;

@Test

public void it\_should\_send\_order\_event() {

//создаем и отправляем на Kafka сущность

OrderSendEvent order = FakeOrder.getOrderSendEvent();

kafkaMessagingService.sendOrder(order);

//Создаем Consumer-а, который должен получить от Kafka отправленную нами(*Producer-ом*) сущность

Properties properties = new Properties();

properties.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9092");

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

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

properties.put(JsonDeserializer.TRUSTED\_PACKAGES, "\*");

properties.put(ConsumerConfig.GROUP\_ID\_CONFIG, "group-java-test");

properties.put(ConsumerConfig.AUTO\_OFFSET\_RESET\_CONFIG, "earliest");

properties.put(JsonDeserializer.VALUE\_DEFAULT\_TYPE, OrderSendEvent.class);

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

consumer.subscribe(Arrays.asList(TOPIC\_NAME\_SEND\_CLIENT));

ConsumerRecords<String, OrderSendEvent> records = consumer.poll(Duration.ofMillis(10000L));

consumer.close();

//сверяем то, что отправили с тем, что получил Consumer

assertEquals(1, records.count());

assertEquals(order.getProductName(), records.iterator().next().value().getProductName());

assertEquals(order.getBarCode(), records.iterator().next().value().getBarCode());

assertEquals(order.getQuantity(), records.iterator().next().value().getQuantity());

assertEquals(order.getPrice(), records.iterator().next().value().getPrice());

}

}

**application.yml специальный для тестов Producer-а**

spring:

kafka:

producer:

key-serializer: org.apache.kafka.common.serialization.StringSerializer

value-serializer: org.springframework.kafka.support.serializer.JsonSerializer

properties:

acks: 1

spring:

json:

add:

type:

headers: false

topic:

send-order: send-order-event

**pom.xml**

<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">

<parent>

<artifactId>kafka-article</artifactId>

<groupId>com.example</groupId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<modelVersion>4.0.0</modelVersion>

<artifactId>producer</artifactId>

<packaging>jar</packaging>

<name>producer</name>

</project>

**DOCKER-COMPOSE**

services:

zookeeper:

image: confluentinc/cp-zookeeper:latest

//блок для тестирования

healthcheck:

test: [ "CMD", "nc", "-vz", "localhost", "2181" ]

interval: 10s

timeout: 3s

retries: 3

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

ZOOKEEPER\_TICK\_TIME: 2000

ports:

- 22181:2181

kafka:

image: confluentinc/cp-kafka:latest

depends\_on:

zookeeper:

condition: service\_healthy

ports:

- 29092:29092

//блок для тестирования

healthcheck:

test: [ "CMD", "nc", "-vz", "localhost", "9092" ]

interval: 10s

timeout: 3s

retries: 3

environment:

KAFKA\_BROKER\_ID: 1

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

KAFKA\_LISTENERS: OUTSIDE://:29092,INTERNAL://:9092

KAFKA\_ADVERTISED\_LISTENERS: OUTSIDE://localhost:29092,INTERNAL://kafka:9092

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: INTERNAL:PLAINTEXT,OUTSIDE:PLAINTEXT

KAFKA\_INTER\_BROKER\_LISTENER\_NAME: INTERNAL

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

kafka-ui:

image: provectuslabs/kafka-ui

container\_name: kafka-ui

ports:

- "8080:8080"

restart: always

depends\_on:

kafka:

condition: service\_healthy

environment:

KAFKA\_CLUSTERS\_0\_NAME: local

KAFKA\_CLUSTERS\_0\_BOOTSTRAPSERVERS: kafka:9092

service-db:

image: postgres:14.7-alpine

environment:

POSTGRES\_USER: username

POSTGRES\_PASSWORD: password

//блок для тестирования

healthcheck:

test: ["CMD-SHELL", "pg\_isready", "-d", "clients\_database"]

interval: 10s

timeout: 3s

retries: 3

ports:

- "15432:5432"

volumes:

- ./infrastructure/db/create\_db.sql:/docker-entrypoint-initdb.d/create\_db.sql

restart: unless-stopped

pgadmin:

container\_name: pgadmin4\_container

image: dpage/pgadmin4:7

restart: always

environment:

PGADMIN\_DEFAULT\_EMAIL: admin@admin.com

PGADMIN\_DEFAULT\_PASSWORD: root

ports:

- "5050:80"

kafka-topics-generator:

image: confluentinc/cp-kafka:latest

depends\_on:

kafka:

condition: service\_healthy

entrypoint: [ '/bin/sh', '-c' ]

command: |

"

# блокируется до тех пор, пока Кафка не станет доступна

kafka-topics --bootstrap-server kafka:9092 --list

echo -e 'Creating kafka topics'

kafka-topics --bootstrap-server kafka:9092 --create --if-not-exists --topic send-order-event --replication-factor 1 --partitions 2

echo -e 'Successfully created the following topics:'

kafka-topics --bootstrap-server kafka:9092 --list

"

|  |  |  |
| --- | --- | --- |
| **CONSUMER 1**  **application.yml**  server:  port: 8082  spring:  kafka:  bootstrap-servers: localhost:29092  **consumer:**  group-id: "order-1"  auto-offset-reset: earliest  key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer  properties:  spring:  json:  trusted:  packages: '\*'    datasource:  url: jdbc:postgresql://${DB\_HOST:localhost}:${DB\_PORT:15432}/orders\_database  username: username  password: password  liquibase:  enabled: true  drop-first: false  change-log: classpath:db/changelog/db.changelog-master.xml  default-schema: public  jpa:  show-sql: false  open-in-view: false  hibernate:  ddl-auto: none  properties:  hibernate:  dialect: org.hibernate.dialect.PostgreSQLDialect  topic:  send-order: send-order-event | **CONSUMER 2**  **application.xml**  server:  port: 8083  spring:  kafka:  bootstrap-servers: localhost:29092  **consumer:**  group-id: "order-2"  auto-offset-reset: earliest  key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer  properties:  spring:  json:  trusted:  packages: '\*'  topic:  send-order: send-order-event | **PRODUCER**  **application.yml**  server:  port: 8081  spring:  kafka:  bootstrap-servers: localhost:29092  **producer:**  key-serializer: org.apache.kafka.common.serialization.StringSerializer  value-serializer: org.springframework.kafka.support.serializer.JsonSerializer  properties:  acks: 1  spring:  json:  add:  type:  headers: false  topic:  send-order: send-order-event |

Тоже самое, только в нормальном формате:

|  |  |
| --- | --- |
| **CONSUMER 1**  **application.properties**  server.port=8082  spring.kafka.bootstrap-servers=localhost:29092  spring.kafka.consumer.group-id= "order-1"  spring.kafka.consumer.auto-offset-reset= earliest  spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer  spring.kafka.consumer. value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer  spring.kafka.consumer. properties.spring.json.trusted.packages='\*'  spring.topic.send-order=send-order-event  spring.datasource.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect  spring.datasource.jpa.show-sql=false  spring.datasource.jpa.open-in-view=false  spring.datasource.jpa. hibernate.ddl-auto=none  spring.datasource.url.jdbc:postgresql=//${DB\_HOST:localhost}:${DB\_PORT:15432}/orders\_database  spring.datasource.username=username  spring.datasource.password=password  spring.datasource.liquibase.enabled=true  spring.datasource.liquibase.drop-first= false  spring.datasource.liquibase. change-log: classpath=db/changelog/db.changelog-master.xml  spring.datasource.liquibase.default-schema=public | **CONSUMER 2**  **application.properties**  server.port=8083  spring.kafka.bootstrap-servers=localhost:29092  spring.kafka.consumer.group-id="order-2"  spring.kafka.consumer.auto-offset-reset=earliest  spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer  spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer  spring.kafka. consumer.properties.spring.json.trusted.packages='\*'  spring.topic.send-order=send-order-event |

|  |
| --- |
| **PRODUCER**  **application. properties**  server.port=8081  spring.kafka.bootstrap-servers=localhost:29092  spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer  spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer  spring.kafka.producer. properties.acks= 1  spring.kafka.producer. properties.spring.json.add.type.headers=false  spring.topic.send-order=send-order-event |