ГУАП

КАФЕДРА № 43

ОТЧЕТ

ЗАЩИЩЕН С ОЦЕНКОЙ ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | П.А. Степанов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7 |
| «Разработка приложения с асинхронной очередью сообщений» |
| по курсу: ТЕХНОЛОГИИ РАЗРАБОТКИ СЕРВЕРНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |
| --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4232 |  | Е.А.Уткина |
|  |  | подпись, дата | инициалы, фамилия |

Санкт-Петербург 2024

# Цель работы:

Целью работы является реализация простой системы распределенной репликации (“писатели-читатели”).

# Задание на лабораторную работу:

1. Скачайте и разверните Apache Kafka
2. Модифицируйте свое приложение со встраиваемой базой данных так, чтобы его можно было запустить в нескольких экземплярах на разных портах
3. Реализуйте в рамках своего приложения Producer и Consumer такие, что

а. Producer при каждой операции записи оповещает соответствующий топик

b. Consumer при получении информации из топика записывает обновление в локальную (встроенную в приложение) базу

1. Продемонстрируйте, что информация, записанная одним приложением, доступна второму приложению.

# Листинг программы:

Конфигурация Продюсера Kafka (KafkaConfig):

package com.example.laba7.config;  
  
import com.example.laba7.ApartmentCreateEvent;  
import org.apache.kafka.clients.admin.NewTopic;  
import org.apache.kafka.clients.producer.ProducerConfig;  
import org.apache.kafka.common.serialization.StringSerializer;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.context.annotation.Primary;  
import org.springframework.kafka.config.TopicBuilder;  
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 java.util.HashMap;  
import java.util.Map;  
  
@Configuration  
public class KafkaConfig {  
  
 @Value("${spring.kafka.producer.bootstrap-servers}")  
 private String bootstrapServers;  
  
 @Value("${spring.kafka.producer.properties.delivery.timeout.ms}")  
 private String deliveryTimeoutMs;  
  
 @Value("${spring.kafka.producer.properties.linger.ms}")  
 private String lingerMs;  
  
 @Value("${spring.kafka.producer.properties.request.timeout.ms}")  
 private String requestTimeoutMs;  
  
 @Value("${spring.kafka.producer.properties.enable.idempotence}")  
 private boolean idempotence;  
  
 @Value("${spring.kafka.producer.properties.max.in.flight.requests.per.connection}")  
 private int maxInFlightRequestsPerConnection;  
  
 @Bean  
 public Map<String, Object> producerConfigs() {  
 Map<String, Object> config = new HashMap<>();  
 config.put(ProducerConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, bootstrapServers);  
 config.put(ProducerConfig.*KEY\_SERIALIZER\_CLASS\_CONFIG*, StringSerializer.class);  
 config.put(ProducerConfig.*VALUE\_SERIALIZER\_CLASS\_CONFIG*, JsonSerializer.class);  
 config.put(ProducerConfig.*DELIVERY\_TIMEOUT\_MS\_CONFIG*, deliveryTimeoutMs);  
 config.put(ProducerConfig.*LINGER\_MS\_CONFIG*, lingerMs);  
 config.put(ProducerConfig.*REQUEST\_TIMEOUT\_MS\_CONFIG*, requestTimeoutMs);  
 config.put(ProducerConfig.*ENABLE\_IDEMPOTENCE\_CONFIG*, idempotence);  
 config.put(ProducerConfig.*MAX\_IN\_FLIGHT\_REQUESTS\_PER\_CONNECTION*, maxInFlightRequestsPerConnection);  
 return config;  
 }  
  
 @Bean  
 public ProducerFactory<String, ApartmentCreateEvent> producerFactory() {  
 return new DefaultKafkaProducerFactory<>(producerConfigs());  
 }  
  
 @Bean  
 @Primary  
 public KafkaTemplate<String, ApartmentCreateEvent> kafkaTemplateProducer(ProducerFactory<String, ApartmentCreateEvent> producerFactory) {  
 return new KafkaTemplate<>(producerFactory);  
 }  
  
 @Bean  
 public NewTopic createTopic() {  
 return TopicBuilder.*name*("apartment-create-event-topic")  
 .partitions(3)  
 .replicas(3)  
 .configs(Map.*of*("min.insync.replicas", "2"))  
 .build();  
 }  
}

Конфигурация Консюмеров Kafka (KafkaConfigConsumer):

package com.example.laba7.config;  
  
import com.example.laba7.ApartmentCreateEvent;  
import org.apache.kafka.clients.consumer.ConsumerConfig;  
import org.apache.kafka.common.serialization.StringDeserializer;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.core.env.Environment;  
import org.springframework.kafka.annotation.EnableKafka;  
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;  
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;  
import org.springframework.kafka.core.ConsumerFactory;  
import org.springframework.kafka.support.serializer.JsonDeserializer;  
  
import java.util.HashMap;  
import java.util.Map;  
  
@Configuration  
@EnableKafka  
public class KafkaConfigConsumer {  
  
 private final Environment environment;  
  
 @Autowired  
 public KafkaConfigConsumer(Environment environment) {  
 this.environment = environment;  
 }  
  
 // Конфигурация ConsumerFactory для чтения из Kafka  
 @Bean  
 public ConsumerFactory<String, ApartmentCreateEvent> consumerFactory() {  
 Map<String, Object> consumerProps = new HashMap<>();  
 consumerProps.put(ConsumerConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, environment.getProperty("spring.kafka.consumer.bootstrap-servers"));  
 consumerProps.put(ConsumerConfig.*GROUP\_ID\_CONFIG*, "apartment-group");  
 consumerProps.put(ConsumerConfig.*KEY\_DESERIALIZER\_CLASS\_CONFIG*, StringDeserializer.class);  
  
 // Используем JsonDeserializer для десериализации объекта ApartmentCreateEvent  
 consumerProps.put(ConsumerConfig.*VALUE\_DESERIALIZER\_CLASS\_CONFIG*, JsonDeserializer.class); // Передаем класс, а не объект  
  
 // Указываем trusted пакеты для JsonDeserializer, чтобы разрешить десериализацию классов  
 consumerProps.put(JsonDeserializer.*TRUSTED\_PACKAGES*, "\*");  
 consumerProps.put(JsonDeserializer.*VALUE\_DEFAULT\_TYPE*, ApartmentCreateEvent.class);  
  
 return new DefaultKafkaConsumerFactory<>(consumerProps);  
 }  
  
  
 // Конфигурация KafkaListenerContainerFactory  
 @Bean  
 public ConcurrentKafkaListenerContainerFactory<String, ApartmentCreateEvent> kafkaListenerContainerFactory(  
 ConsumerFactory<String, ApartmentCreateEvent> consumerFactory) {  
  
 // Конфигурируем контейнер слушателя сообщений  
 ConcurrentKafkaListenerContainerFactory<String, ApartmentCreateEvent> factory = new ConcurrentKafkaListenerContainerFactory<>();  
 factory.setConsumerFactory(consumerFactory);  
  
 return factory;  
 }  
}

Продюсер (PatientServiceImpl):

package com.example.laba7.service;  
  
import com.example.laba7.ApartmentCreateEvent;  
import com.example.laba7.ApartmentDTO;  
import org.apache.kafka.clients.producer.ProducerRecord;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.kafka.core.KafkaTemplate;  
import org.springframework.kafka.support.SendResult;  
import org.springframework.stereotype.Service;  
  
import java.util.UUID;  
import java.util.concurrent.ExecutionException;  
  
@Service  
public class ApartmentServiceImpl implements ApartmentService {  
 private KafkaTemplate<String, ApartmentCreateEvent> kafkaTemplate;  
 private final Logger LOGGER = LoggerFactory.*getLogger*(this.getClass());  
  
 @Autowired  
 public ApartmentServiceImpl(@Qualifier("kafkaTemplateProducer") KafkaTemplate<String, ApartmentCreateEvent> kafkaTemplate) {  
 this.kafkaTemplate = kafkaTemplate;  
 }  
  
 @Override  
 public String createApartment(ApartmentDTO apartmentDTO) throws ExecutionException, InterruptedException {  
 String apartmentId = UUID.*randomUUID*().toString();  
 LOGGER.info("Creating apartment with ID: {}", apartmentId);  
  
 ApartmentCreateEvent apartmentCreateEvent = new ApartmentCreateEvent(apartmentId, apartmentDTO.getCity(), apartmentDTO.getPrice(), apartmentDTO.getAddress());  
  
 ProducerRecord<String, ApartmentCreateEvent> record = new ProducerRecord<>("apartment-create-event-topic", apartmentId, apartmentCreateEvent);  
  
 SendResult<String, ApartmentCreateEvent> result = kafkaTemplate.send(record).get();  
  
 LOGGER.info("Message sent to Kafka topic: {}", result.getRecordMetadata().topic());  
 LOGGER.info("Partition: {}", result.getRecordMetadata().partition());  
 LOGGER.info("Offset: {}", result.getRecordMetadata().offset());  
 LOGGER.info("Apartment creation event sent with ID: {}", apartmentId);  
  
 return apartmentId;  
 }  
}

Консюмер (PatientCreatedEventHandler):

package com.example.laba7;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.kafka.annotation.KafkaHandler;  
import org.springframework.kafka.annotation.KafkaListener;  
import org.springframework.stereotype.Component;  
  
@Component  
@KafkaListener(topics = "apartment-create-event-topic", groupId = "apartment-group-8082", containerFactory = "kafkaListenerContainerFactory")  
public class ApartmentCreatedEventHandler {  
  
 private final Logger LOGGER = LoggerFactory.*getLogger*(this.getClass());  
 private final ApartmentRepository apartmentRepository;  
  
 public ApartmentCreatedEventHandler(ApartmentRepository apartmentRepository) {  
 this.apartmentRepository = apartmentRepository;  
 }  
  
 @KafkaHandler  
 public void handle(ApartmentCreateEvent apartmentCreateEvent) {  
 LOGGER.info("Received ApartmentCreateEvent: {}", apartmentCreateEvent);  
 int apartmentId = apartmentCreateEvent.getId().hashCode();  
 LOGGER.info("Generated apartment ID (hash code): {}", apartmentId);  
  
 if (apartmentRepository.existsById(apartmentId)) {  
 LOGGER.info("ApartmentCreateEvent with ID {} already exists, skipping processing", apartmentCreateEvent.getId());  
 return;  
 }  
  
 Apartment apartment = new Apartment();  
 apartment.setId(apartmentId);  
 apartment.setCity(apartmentCreateEvent.getCity());  
 apartment.setPrice(apartmentCreateEvent.getPrice());  
 apartment.setAddress(apartmentCreateEvent.getAddress());  
  
 LOGGER.info("Saving Apartment with ID {}: {}", apartmentId, apartment);  
 apartmentRepository.save(apartment);  
 LOGGER.info("Apartment with ID {} successfully saved", apartment.getId());  
 }  
}

# Выводы:

Все компоненты системы (продюсер, консюмер, конфигурации Kafka) правильно настроены для обеспечения синхронной обработки сообщений и согласованности данных между несколькими экземплярами приложения.