Сервисы с Apache Kafka и тестирование

Когда сервисы интегрируются при помощи [Kafka]( https://www.confluent.io/blog/building-a-microservices-ecosystem-with-kafka-streams-and-ksql/) очень удобно использовать REST API, как универсальный и стандартный способ обмена сообщениями. При увеличении количества сервисов сложность коммуникаций увеличивается. Для контроля можно и нужно использовать интеграционное тестирование. Такие библиотеки как [testcontainers](<https://www.testcontainers.org/>) или EmbeddedServer прекрасно помогают организовать такое тестирование. Существуют много примеров для [micronaut]( https://micronaut.io/), [Spring Boot]( https://spring.io/projects/spring-boot) и т.д. Но в этих примерах опущены некоторые детали, которые не позволяют с первого раза запустить код. В статье приводятся примеры с подробным описанием.

Пример

Для простоты можно принять такой REST API.

/runs - POST-метод. Инициализирует запрос в канал связи. Принимает данные и возвращает ключ запроса.

/runs/{key}/status – GET-метод. По ключу возвращает статус запроса. Может принимать следующие значения: UNKNOWN, RUNNING, DONE.

/runs /{key} – GET-метод. По ключу возвращает результат запроса.

Подобный API реализован у [livy]( <https://livy.apache.org/docs/latest/rest-api.html>), хотя и для других задач.

Реализация

Будут использоваться: micronaut, Spring Boot.

micronaut

Контроллер для API.

**import** io.micronaut.http.annotation.Body;  
**import** io.micronaut.http.annotation.Controller;  
**import** io.micronaut.http.annotation.Get;  
**import** io.micronaut.http.annotation.Post;  
**import** io.reactivex.Maybe;  
**import** io.reactivex.schedulers.Schedulers;  
  
**import** javax.inject.Inject;  
**import** java.util.UUID;  
  
@Controller(**"/runs"**)  
**public class** RunController {  
 @Inject  
 RunClient runClient;  
  
 @Inject  
 RunCache runCache;  
  
 @Post  
 **public** String runs(@Body String body) {  
 String key = UUID.randomUUID().toString();  
 runCache.statuses.put(key, RunStatus.RUNNING);  
 runCache.responses.put(key, **""**);  
 runClient.sendRun(key, **new** Run(key, RunType.REQUEST, **""**, body));  
 **return** key;  
 }  
  
 @Get(**"/{key}/status"**)  
 **public** Maybe<RunStatus> getRunStatus(String key) {  
 **return** Maybe.just(key)  
 .subscribeOn(Schedulers.io())  
 .map(it -> runCache.statuses.getOrDefault(it, RunStatus.UNKNOWN));  
 }  
  
 @Get(**"/{key}"**)  
 **public** Maybe<String> getRunResponse(String key) {  
 **return** Maybe.just(key)  
 .subscribeOn(Schedulers.io())  
 .map(it -> runCache.responses.getOrDefault(it, **""**));  
 }  
}

Отправка сообщений в kafka.

**import** io.micronaut.configuration.kafka.annotation.\*;  
**import** io.micronaut.messaging.annotation.Body;  
  
@KafkaClient  
**public interface** RunClient {  
 @Topic(**"runs"**)  
 **void** sendRun(@KafkaKey String key, @Body Run run);  
}

Получение сообщений из kafka.

**import** io.micronaut.configuration.kafka.annotation.\*;  
**import** io.micronaut.messaging.annotation.Body;  
  
**import** javax.inject.Inject;  
  
@KafkaListener(offsetReset = OffsetReset.EARLIEST)  
**public class** RunListener {  
 @Inject  
 RunCalculator runCalculator;  
  
 @Topic(**"runs"**)  
 **public void** receive(@KafkaKey String key, @Body Run run) {  
 runCalculator.run(key, run);  
 }  
}

Обработка сообщений происходит в RunCalculator. Для тестов используется особая реализация, в которой происходит переброска сообщений.

Тест.

**import** io.micronaut.http.HttpRequest;  
**import** io.micronaut.http.client.HttpClient;  
  
**import static** org.junit.jupiter.api.Assertions.assertEquals;  
  
**public abstract class** RunBase {  
 **void** run(HttpClient client) {  
 String key = client.toBlocking().retrieve(HttpRequest.POST(**"/runs"**, **"body"**));  
 RunStatus runStatus = RunStatus.UNKNOWN;  
 **while** (runStatus != RunStatus.DONE) {  
 runStatus = client.toBlocking().retrieve(HttpRequest.GET(**"/runs/"** + key + **"/status"**), RunStatus.**class**);  
 **try** {  
 Thread.sleep(500);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 String response = client.toBlocking().retrieve(HttpRequest.GET(**"/runs/"** + key), String.**class**);  
 assertEquals(**"body\_calculated"**, response);  
 }  
}

Для использования EmbeddedServer необходимо.

Подключить библиотеки:

testImplementation(**"org.apache.kafka:kafka-clients:2.6.0:test"**)  
testImplementation(**"org.apache.kafka:kafka\_2.12:2.6.0"**)  
testImplementation(**"org.apache.kafka:kafka\_2.12:2.6.0:test"**)

Тест может выглядеть так.

**import** io.micronaut.context.ApplicationContext;  
**import** io.micronaut.http.client.HttpClient;  
**import** io.micronaut.runtime.server.EmbeddedServer;  
**import** org.junit.jupiter.api.Test;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**public class** RunKeTest **extends** RunBase {  
 @Test  
 **void** test() {  
 Map<String, Object> properties = **new** HashMap<>();  
 properties.put(**"kafka.bootstrap.servers"**, **"localhost:9092"**);  
 properties.put(**"kafka.embedded.enabled"**, **"true"**);  
 **try** (EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer.**class**, properties)) {  
 ApplicationContext applicationContext = embeddedServer.getApplicationContext();  
 HttpClient client = applicationContext.createBean(HttpClient.**class**, embeddedServer.getURI());  
  
 run(client);  
 }  
 }  
}

Для использования [testcontainers](<https://www.testcontainers.org/>) необходимо.

Подключить библиотеки:

implementation(**"org.testcontainers:kafka:1.14.3"**)

Тест может выглядеть так.

**import** io.micronaut.context.ApplicationContext;  
**import** io.micronaut.http.client.HttpClient;  
**import** io.micronaut.runtime.server.EmbeddedServer;  
  
**import** org.junit.jupiter.api.Test;  
  
**import** org.testcontainers.containers.KafkaContainer;  
**import** org.testcontainers.utility.DockerImageName;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**public class** RunTcTest **extends** RunBase {  
  
 @Test  
 **public void** test() {  
 **try** (KafkaContainer kafka = **new** KafkaContainer(DockerImageName.parse(**"confluentinc/cp-kafka:5.5.3"**))) {  
 kafka.start();  
 Map<String, Object> properties = **new** HashMap<>();  
 properties.put(**"kafka.bootstrap.servers"**, kafka.getBootstrapServers());  
 **try** (EmbeddedServer embeddedServer = ApplicationContext.run(EmbeddedServer.**class**, properties)) {  
 ApplicationContext applicationContext = embeddedServer.getApplicationContext();  
 HttpClient client = applicationContext.createBean(HttpClient.**class**, embeddedServer.getURI());  
  
 run(client);  
 }  
 }  
 }  
}

Spring Boot

Контроллер для API.

**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.web.bind.annotation.\*;  
  
**import** java.util.UUID;  
  
@RestController  
@RequestMapping(**"/runs"**)  
**public class** RunController {  
 @Autowired  
 **private** RunClient runClient;  
  
 @Autowired  
 **private** RunCache runCache;  
  
 @PostMapping()  
 **public** String runs(@RequestBody String body) {  
 String key = UUID.randomUUID().toString();  
 runCache.statuses.put(key, RunStatus.RUNNING);  
 runCache.responses.put(key, **""**);  
 runClient.sendRun(key, **new** Run(key, RunType.REQUEST, **""**, body));  
 **return** key;  
 }  
  
 @GetMapping(**"/{key}/status"**)  
 **public** RunStatus getRunStatus(@PathVariable String key) {  
 **return** runCache.statuses.getOrDefault(key, RunStatus.UNKNOWN);  
 }  
  
 @GetMapping(**"/{key}"**)  
 **public** String getRunResponse(@PathVariable String key) {  
 **return** runCache.responses.getOrDefault(key, **""**);  
 }  
}

Отправка сообщений в kafka.

**import** com.fasterxml.jackson.core.JsonProcessingException;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.kafka.core.KafkaTemplate;  
**import** org.springframework.stereotype.Component;  
  
  
@Component  
**public class** RunClient {  
 @Autowired  
 **private** KafkaTemplate<String, String> kafkaTemplate;  
  
 @Autowired  
 **private** ObjectMapper objectMapper;  
  
 **public void** sendRun(String key, Run run) {  
 String data = **""**;  
 **try** {  
 data = objectMapper.writerWithDefaultPrettyPrinter().writeValueAsString(run);  
 } **catch** (JsonProcessingException e) {  
 e.printStackTrace();  
 }  
 kafkaTemplate.send(**"runs"**, key, data);  
 }  
}

Получение сообщений из kafka.

**import** com.fasterxml.jackson.core.JsonProcessingException;  
**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** org.apache.kafka.clients.consumer.ConsumerRecord;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.kafka.annotation.KafkaListener;  
**import** org.springframework.stereotype.Component;  
  
@Component  
**public class** RunListener {  
 @Autowired  
 **private** ObjectMapper objectMapper;  
  
 @Autowired  
 **private** RunCalculator runCalculator;  
  
 @KafkaListener(topics = **"runs"**, groupId = **"m-group"**)  
 **public void** receive(ConsumerRecord<?, ?> consumerRecord) {  
 String key = consumerRecord.key().toString();  
 Run run = **null**;  
 **try** {  
 run = objectMapper.readValue(consumerRecord.value().toString(), Run.**class**);  
 } **catch** (JsonProcessingException e) {  
 e.printStackTrace();  
 }  
 runCalculator.run(key, run);  
 }  
}

Обработка сообщений происходит в RunCalculator. Для тестов используется особая реализация, в которой происходит переброска сообщений.

Тест.

**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.test.web.servlet.MockMvc;  
**import** org.springframework.test.web.servlet.MvcResult;  
**import** org.springframework.test.web.servlet.request.MockMvcRequestBuilders;  
  
**import static** org.junit.jupiter.api.Assertions.assertEquals;  
**import static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;  
  
**public abstract class** RunBase {  
 **void** run(MockMvc mockMvc, ObjectMapper objectMapper) **throws** Exception {  
 MvcResult keyResult = mockMvc.perform(MockMvcRequestBuilders.post(**"/runs"**)  
 .content(**"body"**)  
 .contentType(MediaType.APPLICATION\_JSON)  
 .accept(MediaType.APPLICATION\_JSON))  
 .andExpect(status().isOk())  
 .andReturn();  
  
 String key = keyResult.getResponse().getContentAsString();  
 RunStatus runStatus = RunStatus.UNKNOWN;  
 **while** (runStatus != RunStatus.DONE) {  
 MvcResult statusResult = mockMvc.perform(MockMvcRequestBuilders.get(**"/runs/"** + key + **"/status"**)  
 .contentType(MediaType.APPLICATION\_JSON)  
 .accept(MediaType.APPLICATION\_JSON))  
 .andExpect(status().isOk())  
 .andReturn();  
 runStatus = objectMapper.readValue(statusResult.getResponse().getContentAsString(), RunStatus.**class**);  
 **try** {  
 Thread.sleep(500);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 String response = mockMvc.perform(MockMvcRequestBuilders.get(**"/runs/"** + key)  
 .contentType(MediaType.APPLICATION\_JSON)  
 .accept(MediaType.APPLICATION\_JSON))  
 .andExpect(status().isOk())  
 .andReturn().getResponse().getContentAsString();  
 assertEquals(**"body\_calculated"**, response);  
 }  
}

Для использования EmbeddedServer необходимо.

Подключить библиотеки:

<**dependency**>  
 <**groupId**>org.springframework.kafka</**groupId**>  
 <**artifactId**>spring-kafka</**artifactId**>  
 <**version**>2.5.10.RELEASE</**version**>  
</**dependency**>

<**dependency**>  
 <**groupId**>org.springframework.kafka</**groupId**>  
 <**artifactId**>spring-kafka-test</**artifactId**>  
 <**version**>2.5.10.RELEASE</**version**>  
 <**scope**>test</**scope**>  
</**dependency**>

Тест может выглядеть так.

**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.boot.test.context.TestConfiguration;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.Import;  
**import** org.springframework.kafka.test.context.EmbeddedKafka;  
**import** org.springframework.test.web.servlet.MockMvc;  
  
@AutoConfigureMockMvc  
@SpringBootTest  
@EmbeddedKafka(partitions = 1, brokerProperties = {**"listeners=PLAINTEXT://localhost:9092"**, **"port=9092"**})  
@Import(RunKeTest.RunKeTestConfiguration.**class**)  
**public class** RunKeTest **extends** RunBase {  
 @Autowired  
 **private** MockMvc mockMvc;  
  
 @Autowired  
 **private** ObjectMapper objectMapper;  
  
 @Test  
 **void** test() **throws** Exception {  
 run(mockMvc, objectMapper);  
 }  
  
 @TestConfiguration  
 **static class** RunKeTestConfiguration {  
 @Autowired  
 **private** RunCache runCache;  
  
 @Autowired  
 **private** RunClient runClient;  
  
 @Bean  
 **public** RunCalculator runCalculator() {  
 RunCalculatorWithWork runCalculatorWithWork = **new** RunCalculatorWithWork();  
 runCalculatorWithWork.runCache = runCache;  
 runCalculatorWithWork.runClient = runClient;  
 **return** runCalculatorWithWork;  
 }  
 }  
}

Для использования [testcontainers](<https://www.testcontainers.org/>) необходимо.

Подключить библиотеки:

<**dependency**>  
 <**groupId**>org.testcontainers</**groupId**>  
 <**artifactId**>kafka</**artifactId**>  
 <**version**>1.14.3</**version**>  
 <**scope**>test</**scope**>  
</**dependency**>

Тест может выглядеть так.

**import** com.fasterxml.jackson.databind.ObjectMapper;  
**import** org.apache.kafka.clients.consumer.ConsumerConfig;  
**import** org.apache.kafka.clients.producer.ProducerConfig;  
**import** org.apache.kafka.common.serialization.StringDeserializer;  
**import** org.apache.kafka.common.serialization.StringSerializer;  
**import** org.junit.ClassRule;  
**import** org.junit.jupiter.api.Test;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;  
**import** org.springframework.boot.test.context.SpringBootTest;  
**import** org.springframework.boot.test.context.TestConfiguration;  
**import** org.springframework.context.annotation.Bean;  
**import** org.springframework.context.annotation.Import;  
**import** org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;  
**import** org.springframework.kafka.core.\*;  
**import** org.springframework.test.web.servlet.MockMvc;  
**import** org.testcontainers.containers.KafkaContainer;  
**import** org.testcontainers.utility.DockerImageName;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
@AutoConfigureMockMvc  
@SpringBootTest  
@Import(RunTcTest.RunTcTestConfiguration.**class**)  
**public class** RunTcTest **extends** RunBase {  
 @ClassRule  
 **public static** KafkaContainer kafka = **new** KafkaContainer(DockerImageName.parse(**"confluentinc/cp-kafka:5.5.3"**));  
  
 **static** {  
 kafka.start();  
 }  
  
 @Autowired  
 **private** MockMvc mockMvc;  
  
 @Autowired  
 **private** ObjectMapper objectMapper;  
  
 @Test  
 **void** test() **throws** Exception {  
 run(mockMvc, objectMapper);  
 }  
  
 @TestConfiguration  
 **static class** RunTcTestConfiguration {  
 @Autowired  
 **private** RunCache runCache;  
  
 @Autowired  
 **private** RunClient runClient;  
  
 @Bean  
 ConcurrentKafkaListenerContainerFactory<Integer, String> kafkaListenerContainerFactory() {  
 ConcurrentKafkaListenerContainerFactory<Integer, String> factory = **new** ConcurrentKafkaListenerContainerFactory<>();  
 factory.setConsumerFactory(consumerFactory());  
 **return** factory;  
 }  
  
 @Bean  
 **public** ConsumerFactory<Integer, String> consumerFactory() {  
 **return new** DefaultKafkaConsumerFactory<>(consumerConfigs());  
 }  
  
 @Bean  
 **public** Map<String, Object> consumerConfigs() {  
 Map<String, Object> props = **new** HashMap<>();  
 props.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, kafka.getBootstrapServers());  
 props.put(ConsumerConfig.AUTO\_OFFSET\_RESET\_CONFIG, **"earliest"**);  
 props.put(ConsumerConfig.GROUP\_ID\_CONFIG, **"m-group"**);  
 props.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.**class**);  
 props.put(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, StringDeserializer.**class**);  
 **return** props;  
 }  
  
 @Bean  
 **public** ProducerFactory<String, String> producerFactory() {  
 Map<String, Object> configProps = **new** HashMap<>();  
 configProps.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, kafka.getBootstrapServers());  
 configProps.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.**class**);  
 configProps.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, StringSerializer.**class**);  
 **return new** DefaultKafkaProducerFactory<>(configProps);  
 }  
  
 @Bean  
 **public** KafkaTemplate<String, String> kafkaTemplate() {  
 **return new** KafkaTemplate<>(producerFactory());  
 }  
  
 @Bean  
 **public** RunCalculator runCalculator() {  
 RunCalculatorWithWork runCalculatorWithWork = **new** RunCalculatorWithWork();  
 runCalculatorWithWork.runCache = runCache;  
 runCalculatorWithWork.runClient = runClient;  
 **return** runCalculatorWithWork;  
 }  
 }  
}

Перед всеми тестами необходимо стартовать kafka. Это делается вот таким вот образом:

kafka.start();

Дополнительные свойства для kafka в тестах можно задать в ресурсном файле.

application.yml

**spring**:  
 **kafka**:  
 **consumer**:  
 **auto-offset-reset**: earliest

Ресурсы и ссылки

[Код для micronaut]( <https://github.com/kartzum/s-space/tree/main/r-streams>)

[Код для Spring Boot](<https://github.com/kartzum/s-space/tree/main/s-streams>)

[PART 1: TESTING KAFKA MICROSERVICES WITH MICRONAUT](<https://piotrminkowski.wordpress.com/2019/10/09/part-1-testing-kafka-microservices-with-micronaut/>)

[Testing Kafka and Spring Boot](<https://www.baeldung.com/spring-boot-kafka-testing>)