Apache Kafka и тестирование с Kafka Server

Существуют различные способы для написания тестов с использованием Apache Kafka. К примеру, можно использовать TestContainers (<https://www.testcontainers.org/>) и EmbeddedKafka. Об этом можно почитать, к примеру, вот здесь: <https://habr.com/ru/company/jugru/blog/499408/> . Но существует и вариант для написания тестов с использованием KafkaServer.

Предположим, необходимо разработать сервис отправки сообщений по различным каналам: email, telegram и т.п.

Пусть имя сервиса будет: SenderService.

Сервис должен: слушать заданный канал, выделять из канала нужные ему сообщения, разбирать сообщения и отправлять по нужному каналу для конечной доставки сообщений.

Для проверки сервиса необходимо сформировать сообщение для отправки с использованием канала отправки почты и убедиться в том, что сообщение было передано в конечный канал.

Конечно, в реальных приложениях тесты будут сложнее. Но для иллюстрации выбранного подхода, такого теста будет достаточно.

Сервис и тест реализованы с использованием: Java 1.8, kafka 2.1.0, junit 5.5.2, maven 3.6.1.

Сервис.

Сервис будет иметь возможность начать работу и остановить свою работу.

**void** start()

**void** stop()

При старте необходимо задать, как минимум, следующие параметры:

String bootstrapServers

String senderTopic

EmailService emailService

bootstrapServers – адрес kafka. senderTopic – топик, из которого будут считываться сообщения. emailService – сервис для конечной отправки сообщений по почте. В реальном сервисе таких конечных сервисов будет столько же сколько и конечных каналов отправки сообщений.

Теперь необходим «потребитель», который слушает канал, фильтрует и отправляет сообщения в конечные каналы. Количество таких «потребителей» можно выбирать. Подход для написания  «потребителя» описан вот здесь: <https://www.confluent.io/blog/tutorial-getting-started-with-the-new-apache-kafka-0-9-consumer-client/> .

Collection<AutoCloseable> closeables = **new** ArrayList<>();  
ExecutorService senderTasksExecutor = Executors.*newFixedThreadPool*(**senderTasksN**);  
ExecutorService tasksExecutorService = Executors.*newFixedThreadPool*(**tasksN**);  
**for** (**int** i = 0; i < **senderTasksN**; i++) {  
 SenderConsumerLoop senderConsumerLoop =  
 **new** SenderConsumerLoop(  
 **bootstrapServers**,  
 **senderTopic**,  
 **"sender"**,  
 **"sender"**,  
 tasksExecutorService,  
 **emailService** );  
 closeables.add(senderConsumerLoop);  
 senderTasksExecutor.submit(senderConsumerLoop);  
}

В цикле создается экземпляр «потребителя», запоминается в коллекции и запускается через сервис запуска задач.

При выполнении этого кода «потребители» начинают работать. Сервис ждет их завершения или сигнала для остановки.

Runtime.*getRuntime*().addShutdownHook(**new** Thread(() -> {  
 **for** (AutoCloseable autoCloseable : closeables) {  
 **try** {  
 autoCloseable.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 senderTasksExecutor.shutdown();  
 tasksExecutorService.shutdown();  
 stop();  
 **try** {  
 senderTasksExecutor.awaitTermination(5000, TimeUnit.***MILLISECONDS***);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
}));

При завершении необходимо освободить ресурсы.

«Потребитель».

«Потребитель» имеет следующие публичные методы:

**void** run()

**void** close()

Основной метод: run.

@Override  
**public void** run() {  
 **kafkaConsumer** = *createKafkaConsumerStringString*(**bootstrapServers**, **clientId**, **groupId**);  
 **kafkaConsumer**.subscribe(Collections.*singleton*(**topic**));  
 **while** (**true**) {  
 calculate(**kafkaConsumer**.poll(Duration.*ofSeconds*(1)));  
 }  
}

По входным параметрам создается экземпляр «kafka-потребителя». «kafka-потребитель» подписывается на заданный топик. В бесконечном цикле выбираются записи из топика. И отправляются на обработку.

Для иллюстрации json-сообщения будут иметь несколько полей, которые будут задавать и тип сообщения, и данные для отправки.

Пример сообщения:

{  
 **"subject"**: {  
 **"subject\_type"**: **"send"** },  
 **"body"**: {  
 **"method"**: **"email"**,  
 **"recipients"**: **"mrbrown@ml.ml;mrblack@ml.ml;mrwhite@ml.ml"**,  
 **"title"**: **"42"**,  
 **"message"**: **"73"** }  
}

subject\_type - тип сообщения. Для сервиса нужно значение «send».

method – тип конечного сервиса для отправки. «email» - отправка через почту.

recipients – список получателей.

title – заголовок для сообщения.

message – сообщение.

Обработка всех записей:

**void** calculate(ConsumerRecords<String, String> records) {  
 **for** (ConsumerRecord<String, String> record : records) {  
 calculate(record);  
 }  
}

Обработка одной записи:

**void** calculate(ConsumerRecord<String, String> record) {  
 JSONParser jsonParser = **new** JSONParser();  
 Object parsedObject = **null**;  
 **try** {  
 parsedObject = jsonParser.parse(record.value());  
 } **catch** (ParseException e) {  
 e.printStackTrace();  
 }  
 **if** (parsedObject **instanceof** JSONObject) {  
 JSONObject jsonObject = (JSONObject) parsedObject;  
 JSONObject jsonSubject = (JSONObject) jsonObject.get(***SUBJECT***);  
 String subjectType = jsonSubject.get(***SUBJECT\_TYPE***).toString();  
 **if** (***SEND***.equals(subjectType)) {  
 JSONObject jsonBody = (JSONObject) jsonObject.get(***BODY***);  
 calculate(jsonBody);  
 }  
 }  
}

Распределение сообщений по типу:

**void** calculate(JSONObject jsonBody) {  
 String method = jsonBody.get(***METHOD***).toString();  
 **if** (***EMAIL\_METHOD***.equals(method)) {  
 String recipients = jsonBody.get(***RECIPIENTS***).toString();  
 String title = jsonBody.get(***TITLE***).toString();  
 String message = jsonBody.get(***MESSAGE***).toString();  
 sendEmail(recipients, title, message);  
 }  
}

Отправка в конечную систему:

**void** sendEmail(String recipients, String title, String message) {  
 **tasksExecutorService**.submit(() -> **emailService**.send(recipients, title, message));  
}

Отправка сообщений происходит через сервис исполнения задач.

Ожидания завершения отправки не происходит.

Создание «kafka-потребителя»:

**static** KafkaConsumer<String, String> createKafkaConsumerStringString(  
 String bootstrapServers,  
 String clientId,  
 String groupId  
) {  
 Properties properties = **new** Properties();  
 properties.setProperty(ConsumerConfig.***BOOTSTRAP\_SERVERS\_CONFIG***, bootstrapServers);  
 properties.setProperty(ConsumerConfig.***CLIENT\_ID\_CONFIG***, clientId);  
 properties.setProperty(ConsumerConfig.***GROUP\_ID\_CONFIG***, groupId);  
 properties.setProperty(  
 ConsumerConfig.***KEY\_DESERIALIZER\_CLASS\_CONFIG***,  
 **"org.apache.kafka.common.serialization.StringDeserializer"**);  
 properties.setProperty(  
 ConsumerConfig.***VALUE\_DESERIALIZER\_CLASS\_CONFIG***,  
 **"org.apache.kafka.common.serialization.StringDeserializer"**);  
 properties.put(ConsumerConfig.***AUTO\_OFFSET\_RESET\_CONFIG***, **"earliest"**);  
 **return new** KafkaConsumer<>(properties);  
}

Интерфейс для писем:

**interface** EmailService {  
 **void** send(String recipients, String title, String message);  
}

Тест.

Для теста понадобиться следующее.

Адрес «kafka-сервера». Порт для «kafka-сервера». Имя топика.

Сервис для управления «kafka-сервером». Будет описан ниже.

**public class** SenderServiceTest {  
 @Test  
 **void** consumeEmail() **throws** InterruptedException {  
 String brokerHost = **"127.0.0.1"**;  
 **int** brokerPort = 29092;  
 String bootstrapServers = brokerHost + **":"** + brokerPort;  
 String senderTopic = **"sender\_data"**;  
 **try** (KafkaServerService kafkaServerService = **new** KafkaServerService(brokerHost, brokerPort)) {  
 kafkaServerService.start();  
 kafkaServerService.createTopic(senderTopic);  
  
   
 }  
 }  
}

Задаются параметры. Создается сервис для управления «kafka-сервером». «kafka-сервером» стартует. Создается необходимый топик.

Создается «mock» конечного сервиса для отправки сообщений:

SenderService.EmailService emailService = *mock*(SenderService.EmailService.**class**);

Создается сам сервис и стартует:

SenderService senderService = **new** SenderService(bootstrapServers, senderTopic, emailService);  
senderService.start();

Задаются параметры для сообщения:

String recipients = **"mrbrown@ml.ml;mrblack@ml.ml;mrwhite@ml.ml"**;  
String title = **"42"**;  
String message = **"73"**;

Отправляется сообщение в канал:

kafkaServerService.send(senderTopic, *key*(), *createMessage*(***EMAIL\_METHOD***, recipients, title, message));

Ожидание:

Thread.*sleep*(6000);

Проверка, что сообщение дошло до конечного сервиса:

*verify*(emailService).send(recipients, title, message);

Остановка:

senderService.stop();

Все вместе:

**public class** SenderServiceTest {  
 @Test  
 **void** consumeEmail() **throws** InterruptedException {  
 String brokerHost = **"127.0.0.1"**;  
 **int** brokerPort = 29092;  
 String bootstrapServers = brokerHost + **":"** + brokerPort;  
 String senderTopic = **"sender\_data"**;  
 **try** (KafkaServerService kafkaServerService = **new** KafkaServerService(brokerHost, brokerPort)) {  
 kafkaServerService.start();  
 kafkaServerService.createTopic(senderTopic);  
 SenderService.EmailService emailService = *mock*(SenderService.EmailService.**class**);  
 SenderService senderService = **new** SenderService(bootstrapServers, senderTopic, emailService);  
 senderService.start();  
 String recipients = **"mrbrown@ml.ml;mrblack@ml.ml;mrwhite@ml.ml"**;  
 String title = **"42"**;  
 String message = **"73"**;  
 kafkaServerService.send(senderTopic, *key*(), *createMessage*(***EMAIL\_METHOD***, recipients, title, message));  
 Thread.*sleep*(6000);  
 *verify*(emailService).send(recipients, title, message);  
 senderService.stop();  
 }  
 }  
}

Вспомогательный код:

**public class** SenderFactory {  
 **public static final** String ***SUBJECT*** = **"subject"**;  
 **public static final** String ***SUBJECT\_TYPE*** = **"subject\_type"**;  
 **public static final** String ***BODY*** = **"body"**;  
 **public static final** String ***METHOD*** = **"method"**;  
 **public static final** String ***EMAIL\_METHOD*** = **"email"**;  
 **public static final** String ***RECIPIENTS*** = **"recipients"**;  
 **public static final** String ***TITLE*** = **"title"**;  
 **public static final** String ***MESSAGE*** = **"message"**;  
 **public static final** String ***SEND*** = **"send"**;  
  
 **public static** String key() {  
 **return** UUID.*randomUUID*().toString();  
 }  
  
 **public static** String createMessage(String method, String recipients, String title, String message) {  
 Map<String, Object> map = **new** HashMap<>();  
 Map<String, Object> subject = **new** HashMap<>();  
 Map<String, Object> body = **new** HashMap<>();  
 map.put(***SUBJECT***, subject);  
 subject.put(***SUBJECT\_TYPE***, ***SEND***);  
 map.put(***BODY***, body);  
 body.put(***METHOD***, method);  
 body.put(***RECIPIENTS***, recipients);  
 body.put(***TITLE***, title);  
 body.put(***MESSAGE***, message);  
 **return** JSONObject.*toJSONString*(map);  
 }  
}

Сервис для управления «kafka-сервером».

Основные методы:

**void** start()

**void** close()

**void** createTopic(String topic)

В методе «start» происходит создание сервера и вспомогательных объектов:

Создание «zookeeper» и сохранение его адреса:

**zkServer** = **new** EmbeddedZookeeper();  
String zkConnect = **zkHost** + **":"** + **zkServer**.port();

Создание клиента «zookeeper»:

**zkClient** = **new** ZkClient(zkConnect, 30000, 30000, ZKStringSerializer$.***MODULE$***);  
**zkUtils** = ZkUtils.*apply*(**zkClient**, **false**);

Задание свойств для сервера:

Properties brokerProps = **new** Properties();  
brokerProps.setProperty(**"zookeeper.connect"**, zkConnect);  
brokerProps.setProperty(**"broker.id"**, **"0"**);  
**try** {  
 brokerProps.setProperty(**"log.dirs"**, Files.*createTempDirectory*(**"kafka-"**).toAbsolutePath().toString());  
} **catch** (IOException e) {  
 **throw new** RuntimeException(e);  
}  
brokerProps.setProperty(**"listeners"**, **"PLAINTEXT://"** + **brokerHost** + **":"** + **brokerPort**);  
brokerProps.setProperty(**"offsets.topic.replication.factor"**, **"1"**);  
KafkaConfig config = **new** KafkaConfig(brokerProps);

Создание сервера:

**kafkaServer** = TestUtils.*createServer*(config, **new** MockTime());

Все вместе:

**public void** start() {  
 **zkServer** = **new** EmbeddedZookeeper();  
 String zkConnect = **zkHost** + **":"** + **zkServer**.port();  
 **zkClient** = **new** ZkClient(zkConnect, 30000, 30000, ZKStringSerializer$.***MODULE$***);  
 **zkUtils** = ZkUtils.*apply*(**zkClient**, **false**);  
 Properties brokerProps = **new** Properties();  
 brokerProps.setProperty(**"zookeeper.connect"**, zkConnect);  
 brokerProps.setProperty(**"broker.id"**, **"0"**);  
 **try** {  
 brokerProps.setProperty(**"log.dirs"**, Files.*createTempDirectory*(**"kafka-"**).toAbsolutePath().toString());  
 } **catch** (IOException e) {  
 **throw new** RuntimeException(e);  
 }  
 brokerProps.setProperty(**"listeners"**, **"PLAINTEXT://"** + **brokerHost** + **":"** + **brokerPort**);  
 brokerProps.setProperty(**"offsets.topic.replication.factor"**, **"1"**);  
 KafkaConfig config = **new** KafkaConfig(brokerProps);  
 **kafkaServer** = TestUtils.*createServer*(config, **new** MockTime());  
}

Остановка сервиса:

@Override  
**public void** close() {  
 **kafkaServer**.shutdown();  
 **zkClient**.close();  
 **zkServer**.shutdown();  
}

Создание топика:

**public void** createTopic(String topic) {  
 AdminUtils.*createTopic*(  
 **zkUtils**, topic, 1, 1, **new** Properties(), RackAwareMode.Disabled$.***MODULE$***);  
}

В заключении нужно отметить, что приведенный здесь код лишь иллюстрирует выбранный способ.

Для создания и тестирования сервисов с использованием «kafka» можно обратиться к следующему ресурсу:

<https://github.com/confluentinc/kafka-streams-examples>

Ссылки и ресурсы:

Исходный код:

<https://github.com/kartzum/intro_to_apache_kafka/tree/main/story/resources/k-service>

Код для тестирования с «kafka-сервером»:

<https://gist.github.com/asmaier/6465468>