УДК 004

Использование Spring Kafka для доступа к службе потоков событий

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается пример работы Spring Kafka для доступа к службе IBM Event Streams в IBM Cloud. Для реализации будет использоваться язык программирования Java, фреймворк Spring Boot и IBM Cloud.
Ключевые слова: Java, Kafka, IBM

Using Spring Kafka to access an event stream service

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Erovleva Regina Victorovna
Sholom-Aleichem Priamursky State University
Student

Abstract
This article provides an example of how Spring Kafka works to access the IBM Event Streams service on the IBM Cloud. For implementation, the Java programming language, the Spring Boot framework and IBM Cloud will be used.
Keywords: Java, Kafka, IBM

1 Введение

1.1 Актуальность

IBM Event Streams — это масштабируемая шина сообщений с высокой пропускной способностью, которая предлагает интерфейс Apache Kafka. Spring Boot предоставляет клиент Kafka, позволяющий легко взаимодействовать с потоками событий для приложений Spring.

1.2 Обзор исследований

В.Л. Радишевский и А.Д. Кульневич описали принцип работы распределенного брокера сообщений Kafka для высокоскоростной передачи и агрегации данных [1]. Ю.А. Флёров и Л.Л. Вышинский разработали

программу для организации взаимодействия с Web-клиентами в программных комплексах [2]. А.И. Куреленко разработал программу предназначенную для генерации, приема и передачи типизированных сообщений для программного брокера Apache Kafka [3]. М.А. Потовиченко, М.В. Привалов и С.В. Корнев рассмотрели разработку программного продукта, который обеспечивает учет данных посещения занятий студентов, а также защиту их работ [4]. В.А. Сухомлин описал кратко в своей статье принцип работа комплексной программы дополнительного образования, ориентированную на подготовку разработчиков Java enterprise приложений [5].

1.3 Цель исследования

Цель исследования – использовать Spring Kafka для доступа к службе IBM Event Streams в IBM Cloud.

2 Материалы и методы

Для реализации будет использоваться язык программирования Java, фреймворк Spring Boot и IBM Cloud.

3 Результаты и обсуждения

Используя «Spring Initializer», создаем проект с зависимостями «Web» и «Kafka» (рис.1).

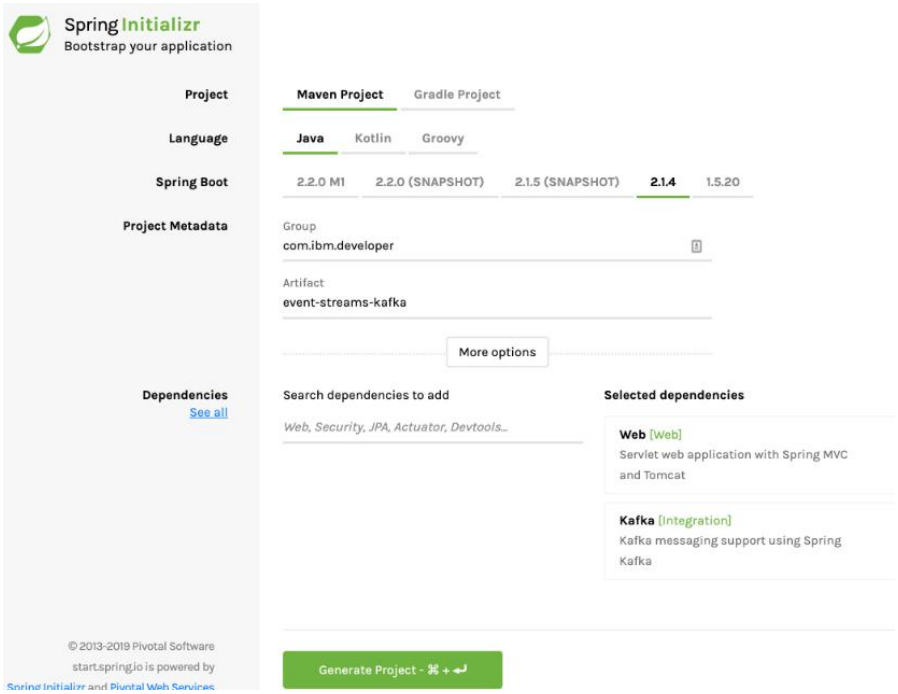

Рисунок 1 – Создание проекта

Теперь создадим экземпляр «Event Streams» в IBM Cloud (рис.2).

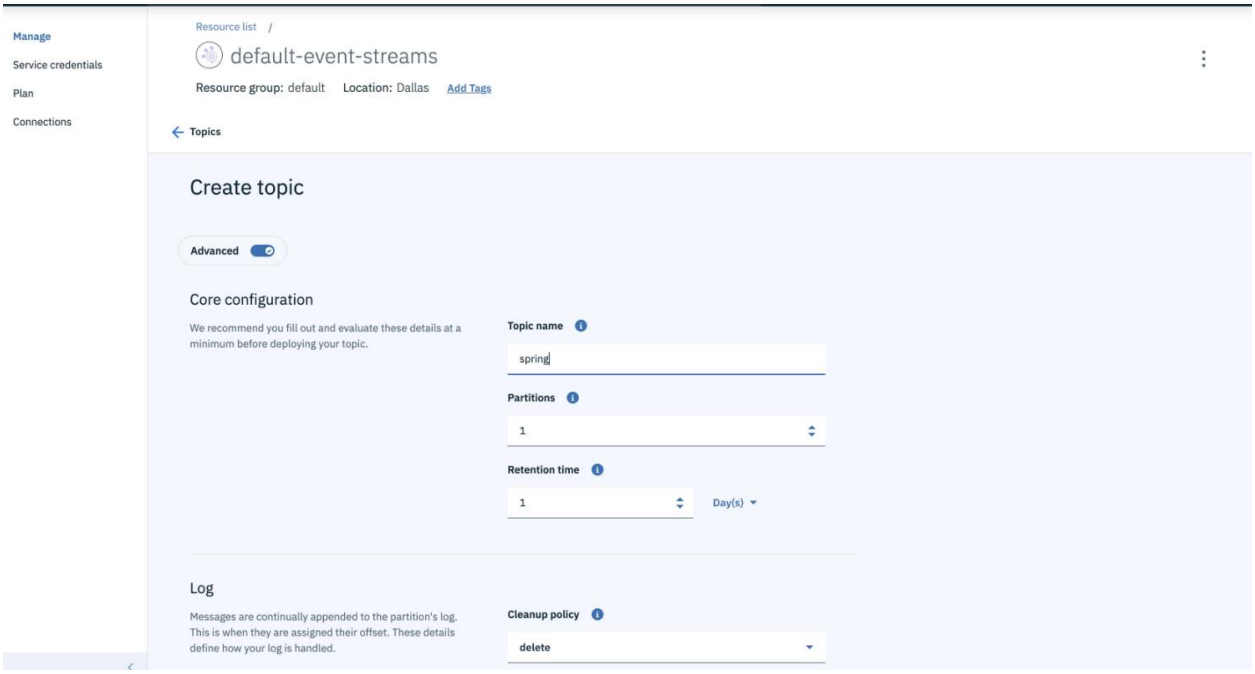

Рисунок 2 – Создание экземпляра в IBM Cloud

Теперь необходимо настроить приложение «Spring Boot» для взаимодействия со службой «Kafka», обычно можно с помощью свойств «Spring Boot» в файле «application.properties» или «application.yml».

Рассмотрим свойства, необходимые для подключения приложения «Spring Boot» к экземпляру «Event Stream» в «IBM Cloud».

В проекте, который был создан ранее, по пути «/src/main/resources» откроем «application.properties» и добавим следующие свойства, используя имя пользователя и пароль, созданные при создании экземпляра в IBM Cloud (рис.3).

```
#Connection
spring.kafka.jaas.enabled=true
spring.kafka.jaas.login-module=org.apache.kafka.common.security.plain.PlainLoginModule
spring.kafka.jaas.options.username=<username>
spring.kafka.jaas.options.password=<password>
spring.kafka.bootstrap-servers=<sasl kafka brokers>
spring.kafka.properties.security.protocol=SASL_SSL
spring.kafka.properties.sasl.mechanism=PLAIN

#Producer
spring.kafka.template.default-topic=spring
spring.kafka.producer.client-id=event-streams-kafka
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

#Consumer
listener.topic=spring
spring.kafka.consumer.group-id=foo
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
```

Рисунок 3 – Файл application.properties

В «application.properties» свойства конфигурации были разделены на три группы:

- Первая группа «Connection» — это свойства, предназначенные для настройки подключения к экземпляру потока событий. Сервер «spring.kafka.bootstrap-servers» может принимать список URL-адресов серверов, разделенных запятыми.
- Вторая группа «Producer» — это свойства, определяющие отправку сообщений в kafka.

«spring.kafka.template.default-topic» определяет тему, в которую нужно будет писать.

«spring.kafka.producer.client-id» используется для ведения журнала, поэтому помимо порта и IP-адреса может быть указано логическое имя.

«spring.kafka.producer.key-serializer» и «spring.kafka.producer.value-serializer» определяет тип и класс Java для сериализации ключа и значения сообщения, отправляемого в поток kafka.

- Третья и последняя группа — это «Consumer», которая определяет чтение сообщений от kafka.

«listener.topic» не является свойством, определенным Spring, но будет использоваться дальше.

«spring.kafka.consumer.group-id» определяет группу, членом которой будет потребитель.

«spring.kafka.consumer.auto-offset-reset» сообщает потребителю, с какого смещения начинать чтение сообщений в потоке, если смещение изначально недоступно.

Как и в случае с производителем, также потребуется определить тип ключа и значения сообщения, а также способ их десериализации, что делается с помощью свойств «spring.kafka.consumer.key-deserializer» и «spring.kafka.consumer.value-deserializer».

В пакете «com.ibm.developer.eventstreamskafka» создаем новый класс с именем «EventStreamsController». Здесь будем использовать контроллер для отправки сообщений и чтения сообщений из темы, которую создали ранее, не выходя из веб-браузера (рис.4).

```java
@RestController
public class EventStreamsController {
    private KafkaTemplate<String, String> template;
    private List<String> messages = new CopyOnWriteArrayList<>();

    public EventStreamsController(KafkaTemplate<String, String> template) {
        this.template = template;
    }

    @KafkaListener(topics = "${listener.topic}")
    public void listen(ConsumerRecord<String, String> cr) throws Exception {
        messages.add(cr.value());
    }

    @GetMapping(value = "send/{msg}")
    public void send(@PathVariable String msg) throws Exception {
        template.sendDefault(msg);
    }

    @GetMapping("received")
    public String recv() throws Exception {
        String result = messages.toString();
        messages.clear();
        return result;
    }
}
```

Рисунок 4 – Контроллер EventStreamsController

Рассмотрим методы контроллера.

В контроллере поддержка клиента «Spring Kafka» основана на «KafkaTemplate<K,V>». Поскольку «EventStreamsController» компонент, управляемый «Spring», определен с одним конструктором, контейнер «Spring» автоматически предоставит файл «KafkaTemplate».

Далее идет настройка «KafkaListener». Он будет регистрировать и читать сообщения, которые были написаны в теме, для которой он был установлен. «${listener.topic}» ссылается на свойство, которое определили в «application.properties», для которого задано значение «spring».

После метод определяет GET конечную точку «/send/{msg}», которая используется для отправки сообщения в «kafka». В теле метода, который вызывает «template.sendDefault(msg)», в качестве альтернативы тема, в которую отправляется сообщение, может быть определена программно путем вызова «template.send(String topic, T data)» вместо этого.

И последний метод определяет вторую GET конечную точку «recieved» для чтения сообщений «KafkaListener» прочитанных вне «spring» темы.

Теперь можно вызвать конечную точку REST для отправки, «http://localhost:8080/send/Hello». Этот запрос отправит сообщение, «Hello» используя «KafkaTemplate» потоки событий (рис.5).

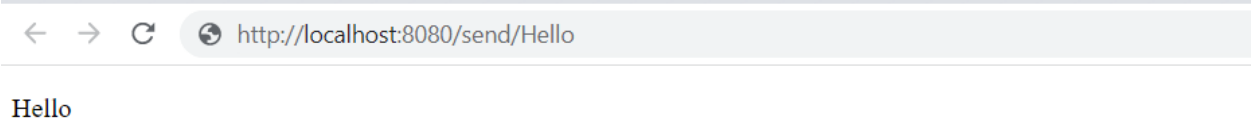

Рисунок 5 – Вызов первой конечной точки

Отправив сообщение, можно также вызвать конечную точку REST для получения, «http://localhost:8080/received». Там должны увидеть ответ от конечной точки с содержимым отправленного сообщения (рис.6).

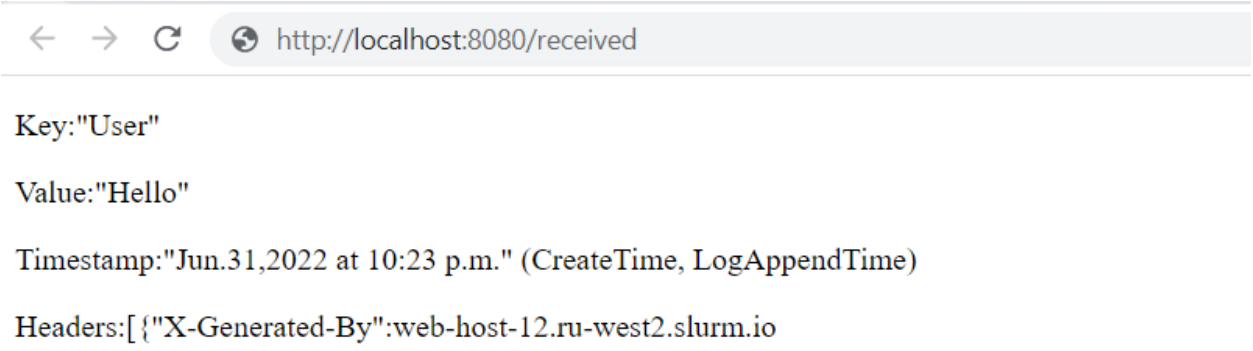

Рисунок 6 – Вызов второй конечной точки

Выводы

Поддержка «Spring Kafka» упрощает отправку и получение сообщений в потоки событий с использованием API-интерфейсов «Spring KafkaTemplate» и «KafkaListener» с конфигурацией Spring.

В данной статье был рассмотрен процесс использования Spring Kafka для доступа к службе потоков событий.

Библиографический список

1. Радишевский В.Л., Кульневич А.Д. Распределенный брокер сообщений kafka для высокоскоростной передачи и агрегации данных // Молодёжь и современные информационные технологии. 2018. № 1. С. 284-285.
2. Флёров Ю.А., Вышинский Л.Л. Программа организации интерфейса web-серверов с java-приложениями // Аллея науки. 2016. № 7. С. 58-61.
3. Куреленко А.И. Генератор сообщений для программного брокера сообщений apache kafka. Тихоокеанский государственный университет. 2017. С. 171-176.
4. Потовиченко М.А., Привалов М.В., Корнев С.В. Компьютеризированная подсистема учета текущей успеваемости студента в условиях вуза //

Информатика, управляющие системы, математическое и компьютерное моделирование. 2019. № 2. С. 71-75.

5. Сухомлин В.А. Подготовка разработчиков корпоративных java-приложений в режиме дистанционного обучения // Информатика, управляющие системы, математическое и компьютерное моделирование. 2013. № 9. С. 175-180.