docker java kafka spring

Mensajería con Kafka y Spring Boot

January 24, 2019

Kafka es un programa de mensajería pensado para comunicaciones asíncronas. Básicamente la idea es que los clientes o **consumidores** se subscriben a un tipo de noticia o **topic** y cuando un emisor o **broker** manda un mensaje sobre ese **topic** Kafka lo distribuye a los **consumidores** suscritos.

Para probar este programa deberemos tener un servidor funcionando con los *topics* ya definidos . En la página https://kafka.apache.org/quickstart hay un manual rápido y muy claro de como levantar uno en apenas 10 minutos.

Hay una extensa documentación sobre **Kafka** en internet, por lo cual no voy a profundizar demasiado en su funcionamiento, ni instalación. No obstante, aclarare dos conceptos básicos de Kafka.

- Siempre se trabaja sobre **topics**. Poniendo un símil con la prensa escrita, un **topic** seria el periódico al que nos hemos subscrito. Solo recibiremos las ediciones (mensajes en Kafka) de ese periódico. Por supuesto una misma persona (suscriptor) mismo puede estar subscrito a muchos periódicos.
- Los subscritores siempre forman **grupos** , aunque en el grupo no haya mas que una sola subscritor.Kafka se encargara que un mensaje solo sea enviado a un subscritor de un grupo.Hay que pensar que Kafka es una tecnología enfocada a la nube y lo normal es que un mismo programa (normalmente un microservicio) este ejecutándose en varios

instancias del servicio y no por todas ellas.

De esta manera suponiendo que el mensaje implicaría realizar un apunte en una base de datos central, solo se realizaría un único apunte y no uno por cada una de las instancias.

En esta entrada voy a explicar como mandar y recibir mensajes usando **Spring Boot** a un servidor Kafka. Además usaremos **Docker** para realizar ciertas pruebas.

El fuente de esta entrada están en https://github.com/chuchip/KafkaTest Empezaremos creando un proyecto **Spring Boot**, con los siguientes starters. - Web - kafka

El starter **Web** lo vamos a necesitar para las pruebas que haremos, pero no lo necesitaremos en un caso *real*.

1. Configuración

La configuración es muy simple. Solo tendremos que poner en el fichero *application.properties*, la dirección del servidor **Kafka** con el parámetro **spring.kafka.bootstrap-server**

```
message.topic.name=${topicname}
message.topic.name2=${topicname2}
message.group.name=${groupid}
spring.kafka.bootstrap-servers=kafkaserver:9092
spring.kafka.consumer.group-id=myGroup
```

Con el parámetro **spring.kafka.consumer.group-id** podemos definir el grupo al que por defecto pertenecerán los *listeners* pero esto es configurable en cada uno de ellos y no es necesario.

Los demás parámetros los usaremos más adelante y son solo para poder realizar pruebas.

2. Enviando mensajes

Los mensajes los enviaremos desde la clase `KafkaMessageProducer` , la cual pongo a continuación.

```
@Component
public class KafkaMessageProducer {
        @Autowired
        private KafkaTemplate<String, String> kafkaTemplate;

        @Value(value = "${message.topic.name:profesorp}")
        private String topicName;

        public void sendMessage(String topic,String message) {
                if (topic==null || topic.trim().equals(""))
                        topic=topicName;
                ListenableFuture<SendResult<String, String>> future = kafkaTemplate
                future.addCallback(new ListenableFutureCallback<SendResult<String,
                        @Override
                        public void onSuccess(SendResult<String, String> result) {
                                System.out.println("Sent message=[" + message + "]
                        }
                        @Override
```

```
                });
        }       }
}
```

Lo primero será pedir a **Spring** que nos inyecte un objeto tipo **KafkaTemplate** .

El *topic* por defecto, sobre el que enviaremos los mensajes lo definimos en la variable **topicname** que, por defecto, tendrá el valor de **message.topic.name** establecida en el fichero *properties* de **Spring Boot**

En la función **sendMessage** sobre el *topic* mandado mandaremos el mensaje deseado.

Para ello crearemos un **ListenableFuture** a partir de **kafkaTemplate**. De esta manera la llamada al servidor de Kafka será asíncrona. Para hacerla simplemente usaremos la función **addCallback** de la clase **ListenableFuture**, pasándole el interface **ListenableFutureCallback**.

La función **onSuccess** será ejecutada si todo va bien y la función **onFailure** en caso de error.

3. Recibiendo mensajes

Los mensajes los recibiremos en la clase `KafkaTestListener`

```
@Component
public class KafkaTestListener {

        @KafkaListener(topics = "${message.topic.name:profesorp}", groupId = "${m
```

```
}

        @KafkaListener(topics = "${message.topic.name2:profesorp-group}", groupI
        public void listenTopic2(String message) {
                System.out.println("Recieved Message of topic2 in  listener "+me
        }
}
```

En la función **listenTopic1** con la etiqueta **@KafkaListener**, definiremos el *topics* , en plural pues pueden ser varios, que queremos escuchar. En este caso escucharemos los definidos en la variable **message.topic.name** del fichero *properties* de **Spring Boot**. Si esa variable no estuviera definida, tendrá el valor `profesorp` . Además especificamos el **grupo** al que pertenece el *listener*. Recordar si no lo definimos cogerá el que hayamos configurado con el parametro **spring.kafka.consumer.group-id**

En la función **listenTopic2** recibiremos los mensajes del *topic* **message.topic.name2**.

Probando la aplicación

En la clase `KafkaTestController` he puesto un controlador **REST** muy sencillo que escucha en **/add/{topic}** . Acepta peticiones **POST** y lo único que hace es llamar a **kafkaMessageProducer.sendMessage** mandando un mensaje al servidor **Kafka**. Este es el código.

```
public class KafkaTestController {
        @Autowired
        KafkaMessageProducer kafkaMessageProducer;

        @PostMapping("/add/{topic}")
        public void addIdCustomer( @PathVariable String topic,@RequestBody  String
        {
                kafkaMessageProducer.sendMessage(topic,body);
        }
}
```

Para hacer las pruebas he creado el fichero **jar** a través de maven (***maven package***) y he creado este pequeño script al que he llamado `javaapp.sh` que lo ejecuta:

```
java -Dtopicname=$TOPICNAME  -Dtopicname2=$TOPICNAME2 -Dgroupid=$GROUPID -jar  ./ka
```

Como variables de entorno habremos definido TOPICNAME,TOPICNAME2 y GROUPID.

```
export TOPICNAME="mytopic_1"
export TOPICNAME2="mytopic_2"
EXPORT GROUPID="profe_group"
```

```
> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --
> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --
```

Para comprobar que están creados los *topics* podemos ejecutar el comando

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Una vez tengamos la aplicación corriendo en un terminal, abrimos otro, y ejecutamos la sentencia:

```
> curl --request POST localhost:8080/add/mytopic_1 -d "Mensaje para topic1 "
```

La cual hace una petición POST a la URL **localhost:8080/add/topic2** . En el cuerpo de esta petición ira el texto *“Mensaje para topic profegroup”*

La salida que veremos en nuestro programa será la siguiente

```
Recieved Message of mytopic_1 in  listener:  Mensaje+para+topic1=
```

Con lo cual vemos, como se ha enviado y luego recibido correctamente en el **topic1**.

Si ahora ejecutamos la sentencia:

```
curl --request POST localhost:8080/add/mytopic_2 -d "Mensaje para topic2"
```

Observaremos la siguiente salida:

```
Sent message=[Mensaje+para+topic+mytopic_2=] with offset=[32]
Recieved Message of topic1 in  listener: Mensaje+para+topic+topic2=
```

Dockerizando la aplicación

Para probar la aplicación en un entorno más real vamos a dockerizar nuestra aplicación.

El fichero `DockerFile` sera el siguiente:

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
```

```
ENV TOPICNAME2 profegroup
ENV GROUPID profe_group
COPY $JAR_FILE /tmp/kafkatest.jar
COPY javaapp.sh /tmp/
EXPOSE 8080
ENTRYPOINT "/tmp/javaapp.sh"
```

En el directorio donde este el fichero **DockerFile** deberemos tener también **javaapp.sh** y **kafakatest.jar**

Creamos la imagen con el comando:

```
docker build -t kafkatest .
```

Antes de ejecutarlo debemos asegurarnos que nuestro servidor de Kafka este bien configurado.

Una vez un consumidor llama al servidor, lo primero que hace este servidor es comunicarle al cliente la dirección donde esta el servidor *lider* con el que debe comunicarse. Normalmente esto no es problema, pero cuando la red es un poco compleja, como en el caso de Docker o máquinas virtuales hay que asegurarse que el nombre del *lider* que suministra Kafka, es accesible por el *consumidor*.

Como Docker crea siempre un interface virtual en la dirección **172.17.0.1**, añadiremos esta IP al fichero **/etc/hosts**, añadiéndole la línea

Y en el fichero **config/server.properties** definiremos las siguientes variables:

```
listeners=PLAINTEXT://kafkaserver:9092
advertised.listeners=PLAINTEXT://kafkaserver:9092
```

La primera línea especifica donde debe escuchar el servidor de Kafka y la segunda el nombre que mandara al **consumidor** cuando pregunte por el *lider* con el que debe conectarse.

Reiniciamos el servidor de Kafka, y ejecutamos el contenedor docker con la siguiente sentencia:

```
docker run -i -t  --name=kafkatest1 -p 8880:8080  --hostname kafkatest1   -e "TOPIC
```

En otro terminal ejecutamos el mismo comando cambiando el nombre del contenedor y mapeando el puerto 8080 al 8881

```
docker run -i -t  --name=kafkatest2 -p 8881:8080  --hostname kafkatest2   -e "TOPIC
```

```
curl --request POST localhost:8880/add/my_topic1 -d "Mensaje para topic numero 1"
```

Observaremos que solo uno de los dos programas recibe el mensaje de Kafka pues ambos pertenecen al mismo grupo (**profe_group**)

Abriendo un nuevo terminal, volvemos a lanzar una nueva instancia de docker, especificando **otro_grupo**.

```
docker run -i -t  --name=kafkatest3 -p 8881:8080  --hostname kafkatest2   -e "TOPIC
```

Veremos que la anterior petición **curl** hace que los mensajes sean recibidos en dos instancias en **docker**. En una de las dos, que tiene el grupo **profe_group** y en la que tiene el grupo **otro_grupo**

Por supuesto este programa se puede mejorar mucho, pues no hemos utilizado serialización para recibir objetos directamente, ni hemos hablado de las particiones de Kafka, ni del tratamiento de errores, seguridad, etc, pero como una pequeña introducción al mundo de Kafka y la mensajería. Como siempre se agradecerá cualquier comentario aquí o en mi cuenta de Twitter @chuchip.

¡¡Hasta la próxima clase!!