

Spring framework 5 - Kafka

Descarga kafka

Para descargar apache kafka debes acceder a la siguiente url :

<https://kafka.apache.org/downloads>

Asegurate de descargar la versión que dice binary downloads.

Inicia kafka

Para iniciar tu servidor de kafka deberás ejecutar los siguientes comandos:

```
$ bin/zookeeper-server-start.sh  
config/zookeeper.properties
```

```
$ bin/kafka-server-start.sh  
config/server.properties
```

Esto iniciará tanto zookeeper como kafka.

Creando un topic

Los mensajes se procesan en topics, para crear uno deberás ejecutar el siguiente comando:

```
$ bin/kafka-topics.sh  
--bootstrap-server localhost:9092  
--create --topic dev4j-topic  
--partitions 5 --replication-factor 1
```

Este comando recibe los siguientes parámetros:

- bootstrap-server = Kafka server
- topic = Nombre del topic a crear
- partitions = Número de particiones
- replication-factor = Número de réplicas por broker

Listando topics

Puedes listar los topics disponibles ejecutando:

```
$ bin/kafka-topics.sh --list  
--bootstrap-server localhost:9092
```

Salida de ejemplo:

```
devs4j-topic
```

Ver definición de un topic

Si deseas consultar como se definió un topic puedes describirlo con el siguiente comando:

```
$ bin/kafka-topics.sh --describe  
--topic dev4j-topic  
--bootstrap-server localhost:9092
```

Salida de ejemplo:

```
Topic: dev4j-topic      PartitionCount: 5  
ReplicationFactor: 1  
  
Topic: dev4j-topic      Partition: 0 Leader: 0  
Replicas: 0 Isr: 0  
Topic: dev4j-topic      Partition: 1 Leader: 0  
Replicas: 0 Isr: 0  
Topic: dev4j-topic      Partition: 2 Leader: 0  
Replicas: 0 Isr: 0  
Topic: dev4j-topic      Partition: 3 Leader: 0  
Replicas: 0 Isr: 0  
Topic: dev4j-topic      Partition: 4 Leader: 0  
Replicas: 0 Isr: 0
```

De la salida podemos observar lo siguiente:

- Tiene 5 particiones, que
- Solo se tiene una replica
- Solo hay un líder el cual es quien contiene el topic especificado.

Crear un producer

Para iniciar un producer ejecutaremos el siguiente comando:

```
$ bin/kafka-console-producer.sh --topic  
devs4j-topic --bootstrap-server localhost:9092
```

Crear un consumer

Para iniciar un consumer ejecutaremos el siguiente comando:

```
$ bin/kafka-console-consumer.sh --topic  
devs4j-topic --from-beginning --bootstrap-server  
localhost:9092
```

El parámetro --from-beginning permite especificar si queremos recibir solo los mensajes nuevos o queremos leer todos desde el inicio.

Configuración

Para configurar Spring kafka se debe incluir la siguiente dependencia:

```
<dependency>  
<groupId>org.springframework.kafka</groupId>  
<artifactId>spring-kafka</artifactId>  
</dependency>
```

Consumer properties

A continuación se listan las configuraciones de un consumer de kafka:

```
@Bean  
public Map<String, Object> consumerProps() {  
    Map<String, Object> props = new HashMap<>();  
  
    props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");  
    props.put(ConsumerConfig.GROUP_ID_CONFIG, "group");  
    props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, true);  
  
    props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "100");  
    props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "15000");  
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,  
IntegerDeserializer.class);  
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,  
StringDeserializer.class);  
    return props;  
}
```

A continuación la explicación de cada una:

- **BOOTSTRAP_SERVERS_CONFIG** : Lista de brokers de kafka en el cluster.
- **GROUP_ID_CONFIG** : Consumer group que consumirá los mensajes
- **ENABLE_AUTO_COMMIT_CONFIG** : Determina si se hará commit al offset de forma periódica
- **AUTO_COMMIT_INTERVAL_MS_CONFIG** : Determina la frecuencia en milisegundos en la que se hará commit a los offsets, solo es necesaria si **ENABLE_AUTO_COMMIT_CONFIG = true**.
- **SESSION_TIMEOUT_MS_CONFIG** : Timeout utilizado para determinar errores en los clientes.
- **KEY_DESERIALIZER_CLASS_CONFIG** : Clase a utilizar para deserializar la llave
- **VALUE_DESERIALIZER_CLASS_CONFIG** : Clase a utilizar para deserializar el mensaje

Lectura de mensajes de kafka

Una vez definidas las propiedades se debe configurar el listener:

```
@Bean  
public ConsumerFactory<Integer, String> consumerFactory() {  
    return new DefaultKafkaConsumerFactory<>(consumerProps());  
}  
  
@Bean  
public ConcurrentKafkaListenerContainerFactory<Integer, String>  
kafkaListenerContainerFactory() {  
    ConcurrentKafkaListenerContainerFactory<Integer, String> factory =  
new ConcurrentKafkaListenerContainerFactory<>();  
    factory.setConsumerFactory(consumerFactory());  
    return factory;  
}
```

Para utilizar el listener crearemos el siguiente método en un componente de spring

```
@KafkaListener(topics = "devs4j-topic", groupId = "consumer")  
public void listen(String message) {  
    System.out.println("Received Message in group foo: "+message);  
}
```

Producer properties

A continuación se listan las configuraciones de un producer de kafka:

```
private Map<String, Object> producerProps() {  
    Map<String, Object> props = new HashMap<>();  
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,  
"localhost:9092");  
    props.put(ProducerConfig.RETRIES_CONFIG, 0);  
    props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);  
    props.put(ProducerConfig.LINGER_MS_CONFIG, 1);  
    props.put(ProducerConfig.BUFFER_MEMORY_CONFIG, 33554432);  
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,  
IntegerSerializer.class);  
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,  
StringSerializer.class);  
    return props;  
}
```

@Bean

```
public KafkaTemplate<Integer, String> createTemplate() {  
    Map<String, Object> senderProps = producerProps();  
    ProducerFactory<Integer, String> pf = new  
DefaultKafkaProducerFactory<Integer, String>(senderProps);  
    KafkaTemplate<Integer, String> template = new  
KafkaTemplate<>(pf);  
    return template;  
}
```

Se incluyen las siguientes nuevas propiedades:

- **RETRIES_CONFIG**: Define los reintentos que se realizarán en caso de error.
- **BATCH_SIZE_CONFIG**: El producer agrupará los registros en batches, mejorando el performance (está definido en bytes).
- **LINGER_MS_CONFIG**: Los batches se agruparán de acuerdo de un periodo de tiempo, está definido en milisegundos.
- **BUFFER_MEMORY_CONFIG**: Define el espacio de memoria que se asignará para colocar los mensajes que están pendientes por enviar.

Utilizando el producer

Una vez configurado el kafka template lo utilizaremos como se muestra a continuación:

@Autowired

```
private KafkaTemplate<Integer, String> kafkaTemplate;
```

```
public Role createRole(Role role) {  
    Role save = repository.save(role);  
    kafkaTemplate.send  
("devs4j-topic", save.getId(), save.getName());  
    return save;  
}
```

El código anterior publicará un mensaje en el topic dev4j-topic con un message key igual al id del objeto y un mensaje con el contenido del objeto.

Terminar tu ambiente

Para limpiar el ambiente de los ejemplos de prueba terminaremos los procesos en el siguiente orden:

- Control + C en los consumers y producers
- Control + C en los brokers de kafka
- Control + C en el servidor de zookeeper

Si se desea borrar la información, ejecutar:

```
$ rm -rf /tmp/kafka-logs /tmp/zookeeper
```



www.twitter.com/devs4j



www.facebook.com/devs4j

www.devs4j.com