Kafka Tutorial :

Aim:

A diagram of a business

Description automatically generated

https://stream.wikimedia.org/v2/stream/recentchange

<https://esjewett.github.io/wm-eventsource-demo/>

Use Rest APi exposed by Wikimedia a stream and save it to database using Kafka as Messaging broker.

A diagram of a company

Description automatically generated

Inside Borkers weh have Topics , inside Topics we have Partition 1 , partiton 2 …etc, we have offset which means the boxes in erray imagine , to fetch from for receivers .

We need to start Zookeeper , Zookeeper is as service use it to manage Kafak brokers in Kafka Cluster.

A diagram of a system

Description automatically generated

Run zookeeper and then in new terminal run kafka , then you will se it running on 9092 port

Recorded new controller, from now on will use node 192.168.0.23:9092 (id: 0 rack: null) (kafka.server.BrokerToControllerRequestThread)

Create a springboot application online , in pom file use pacakgaing as pom so we makeit as a parent

And create a module kafka-producer-wikimeida. Since it is not a spring boot project , Create a package and a class in it , annotate with @SpringBootApplication

@SpringBootApplication  
public class SpringBootProducerApplication {  
 public static void main(String[] args){  
 SpringApplication.*run*(SpringBootProducerApplication.class);  
 }  
}

Run it with spring boot starter 2.7.5 for java 8 or 3.0 for java 17

Use maven clean install to see build is success

Use packaging as jar for module project

<packaging>jar</packaging>

A white background with black text

Description automatically generated

WE create ed parent and child proects.

Go to producer project > resources >application.properties file. Here we configure our producer.

spring.kafka.producer.bootstrap-servers:localhost:9092

Here we tell Spring boot that kafka broker is running on localhost:9092.

Whenever we create a kafka producer it will read realtime stream data from wikimeida and connect to kafka brker and write it to kafka broker

spring.kafka.producer.key-serializer:org.apache.kafka.common.serialization.StringSerializer  
spring.kafka.producer.value-serializer:org.apache.kafka.common.serialization.StringSerializer

Kafka producer uses these classes to serialize input data.

Now create a kafka topic

*@Configuration  
public class* KafkaTopicConfig{  
 *@Bean  
 public* NewTopic topic(){  
 *return* TopicBuilder.name("wikimedia\_recentchange").build();  
 }  
}

A white background with black text

Description automatically generated

Create WikimediaChangesProducer , lets inject kafka template to send messages to kafka broker.

Spring will inject whenever it find single parameterized constructor, no need to ad @Autwired for dependency.

To listen to Wikimedia stream realtime data add okhttp and jackson core and djacksn databind for json

*@Service  
public class* WikimediaChangesProducer {  
 *private static final* Logger LOGGER = LoggerFactory.getLogger(WikimediaChangesProducer.*class*);  
  
 *private* KafkaTemplate<String , String> kafkaTemplate;  
  
 *public* WikimediaChangesProducer(KafkaTemplate<String, String> kafkaTemplate) {  
 *this*.kafkaTemplate = kafkaTemplate;  
 }  
  
 *public void* sendMesage(){  
 String topic = "wikimedia\_recentchange";  
  
 *//To read real time stream data from wikimedia , we use event source .* }  
  
  
}

TO handle events, create a separate event handler . Whenever there is a new event in Wikimedia, then this handler will be triggered , in which on message will be triggered using send method

*public class* WikimediaChangesHandler *implements* EventHandler {  
 *private static final* Logger LOGGER = LoggerFactory.getLogger(WikimediaChangesHandler.*class*);  
 *private* KafkaTemplate<String,String> kafkaTemplate;  
 *private* String topic;  
 *public* WikimediaChangesHandler(KafkaTemplate<String, String> kafkaTemplate, String topic) {  
 *this*.kafkaTemplate = kafkaTemplate;  
 *this*.topic = topic;  
 }  
 *@Override  
 public void* onOpen() *throws* Exception {  
  
 }  
  
 *@Override  
 public void* onClosed() *throws* Exception {  
  
 }  
  
 *@Override  
 public void* onMessage(String s, MessageEvent messageEvent) *throws* Exception {  
 LOGGER.info(String.format("event data ->%s" , messageEvent.getData()));  
 kafkaTemplate.send(topic, messageEvent.getData());  
; }  
  
 *@Override  
 public void* onComment(String s) *throws* Exception {  
  
 }  
  
 *@Override  
 public void* onError(Throwable throwable) {  
  
 }  
}

After this, the producer code will be as follows

*@Service  
public class* WikimediaChangesProducer {  
 *private static final* Logger LOGGER = LoggerFactory.getLogger(WikimediaChangesProducer.*class*);  
  
 *private* KafkaTemplate<String , String> kafkaTemplate;  
  
 *public* WikimediaChangesProducer(KafkaTemplate<String, String> kafkaTemplate) {  
 *this*.kafkaTemplate = kafkaTemplate;  
 }  
  
 *public void* sendMesage() *throws* InterruptedException {  
 String topic = "wikimedia\_recentchange";  
  
 *//To read real time stream data from wikimedia , we use event source .* EventHandler eventHandler = *new* WikimediaChangesHandler(kafkaTemplate, topic );  
 String url= "https://stream.wikimedia.org/v2/stream/recentchange";  
  
 *//Create and event source to connect to wikimedia* EventSource.Builder builder = *new* EventSource.Builder(eventHandler , URI.create(url));  
 EventSource eventSource = builder.build();  
  
 eventSource.start();  
 TimeUnit.MINUTES.sleep(10);  
 }  
}

Now we need to call the producer in application

*@SpringBootApplication  
public class* SpringBootProducerApplication *implements* CommandLineRunner {  
 *public static void* main(String[] args){  
 SpringApplication.run(SpringBootProducerApplication.*class*);  
 }  
 *@Autowired  
 private* WikimediaChangesProducer wikimediaChangesProducer;  
  
 *//runs wheenver applicaiton is started  
 @Override  
 public void* run(String... args) *throws* Exception {  
 wikimediaChangesProducer.sendMesage();  
 }  
}

When you run , you should see data .

To verify if Kafak producer has sent actual realtime event data to kafka topic or not , open terminal and type following command and trigger command from cli.

bin/kafka-console-consumer.sh --topic wikimedia\_recentchange --from-beginning --bootstrap-server localhost:9092;

A white background with black text

Description automatically generated

We create application.properties file and add localhost:9092 and a kafka comusmer group name

And an offset.

spring.kafka.consumer.bootstrap-servers:localhost:9092  
spring.kafka.comsumer.group-id:myGroup  
spring.kafka.consumr.auto-offset-reset:earliest  
spring.kafka.consumer.key-serializer:org.apache.kafka.common.serialization.StringSerializer  
spring.kafka.consumer.value-serializer:org.apache.kafka.common.serialization.StringSerializer

A white background with black text

Description automatically generated

Create KafkaDatabaseConsume class

*@Service  
public class* KafkaDatabaseConsumer {  
 *private static final* Logger LOGGER = LoggerFactory.getLogger(KafkaDatabaseConsumer.*class*);

*@KafkaListener*(topics = "wikimedia\_recentchange" , groupId = "myGroup")  
 *public void* consume(String eventMessage){  
 LOGGER.info(String.format("Event Message received ->%s", eventMessage));  
 }  
}

A close-up of a sign

Description automatically generated

Createna dtabase and add Spring dtat jpa and mydql jdbc driver depenedcy ot comsuemr project.

N consumer paplicaiton.properties file add below

spring.datasource.url = jdbc:mysql://localhost:3306/wikimedia  
spring.datasource.username=test  
spring.datasource.password=adminadmin  
  
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect  
spring.jpa.hibernate.ddl-auto=update  
  
spring.jpa.properties.hibernate.show\_sql=true  
spring.jpa.properties.hibernate.use\_sql\_comments=true  
spring.jpa.properties.hibernate.format\_sql=true

A white background with black text

Description automatically generated

Creat and Enitty class WikimediaData

likebelow

*@Entity  
@Table*(name="wikimedia\_recentchange")  
*@Getter  
@Setter  
public class* WikimediaData {  
 *@Id  
 @GeneratedValue*(strategy = GenerationType.IDENTITY)  
 *private* Long id;  
 *@Lob  
 private* String wikiEventData;  
  
}

And for CRUD methods create and use below interface

*public interface* WikimediaDataRepository *extends* JpaRepository<WikimediaData,Long> {  
}

And use this wikimediaDataRepository in consume class,ince we have only one constructor with one param , we no need to use Autowired

*@Service  
public class* KafkaDatabaseConsumer {  
 *private static final* Logger LOGGER = LoggerFactory.getLogger(KafkaDatabaseConsumer.*class*);  
  
 *private* WikimediaDataRepository wikimediaDataRepository;  
  
 *public* KafkaDatabaseConsumer(WikimediaDataRepository wikimediaDataRepository) {  
 *this*.wikimediaDataRepository = wikimediaDataRepository;  
 }  
  
 *@KafkaListener*(topics = "wikimedia\_recentchange" , groupId = "myGroup")  
 *public void* consume(String eventMessage){  
 LOGGER.info(String.format("Event Message received ->%s", eventMessage));  
 }  
}