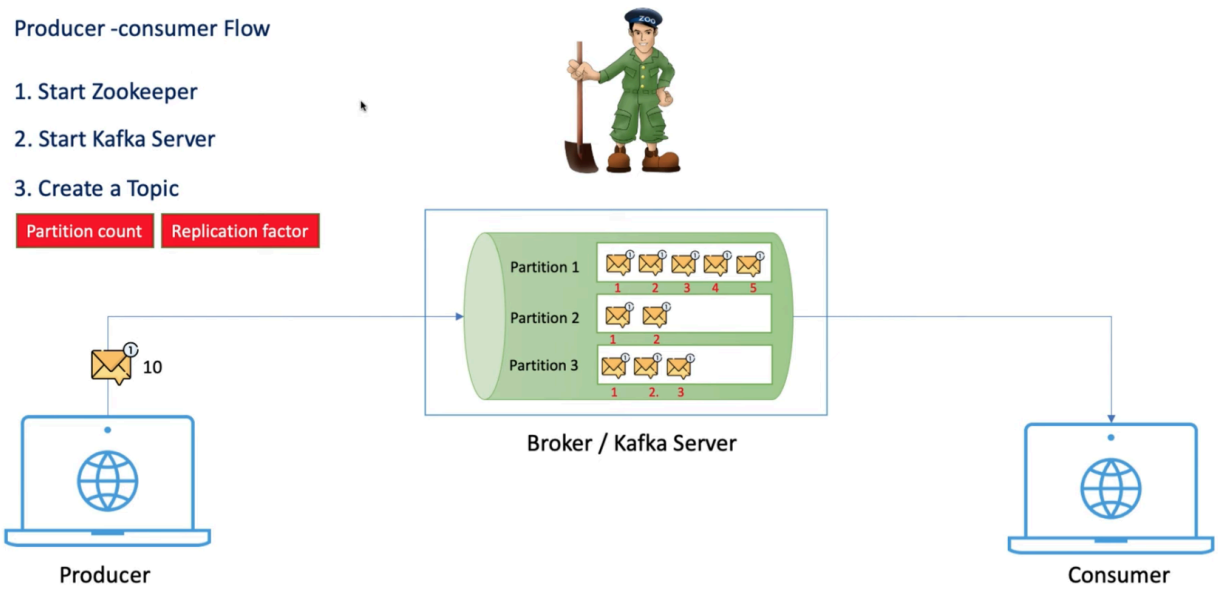
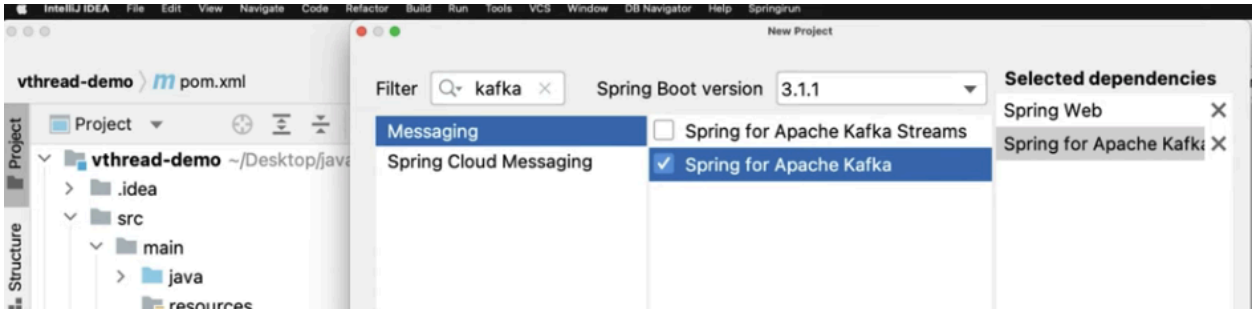
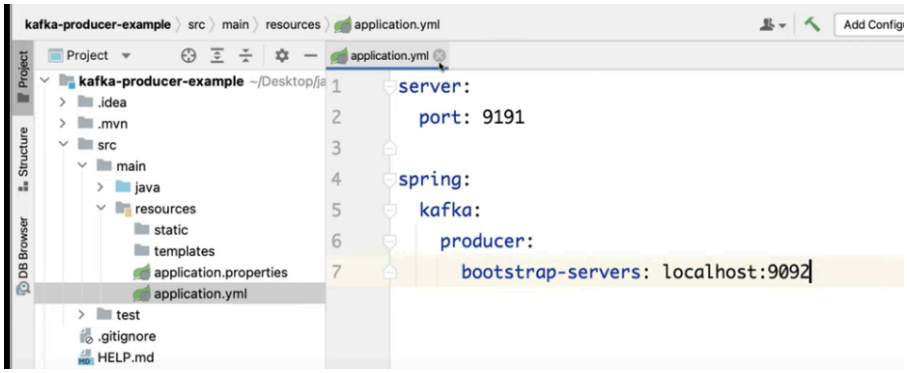


SpringBoot dependencies:



1st start zookeeper and kafka servers

We have to specify these kafka servers in spring boot application



```
KafkaMessagePublisher.java
11  ic class KafkaMessagePublisher {
12
13  @Autowired
14  private KafkaTemplate<String,Object> template;
15
16  public void sendMessageToTopic(String message){
17      CompletableFuture<SendResult<String, Object>> future = template.send( topic: "javc
18      future.get();
19  }
20
21
```

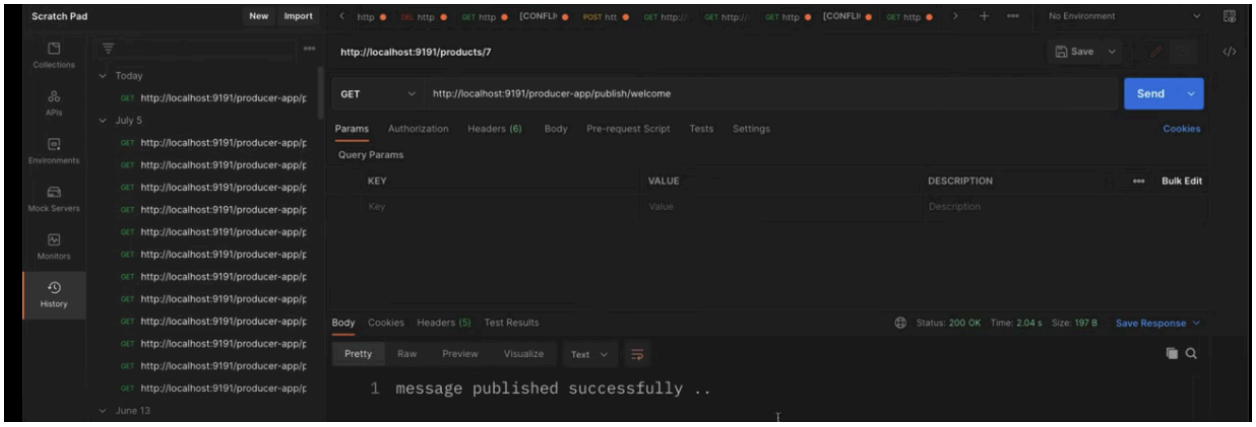
If template.send returns CompletableFuture. The CompletableFuture.get method blocks the thread => means it will wait for the results to come. It slows down the producer. So we need to handle results asynchronously. So the subsequent message does not wait for the result of the previous message. We can do it using call back implementation as shown below.

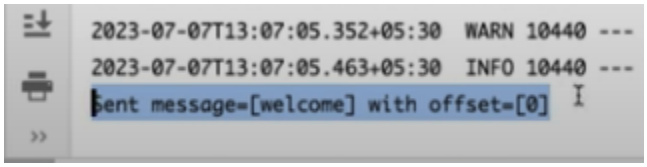
Service class

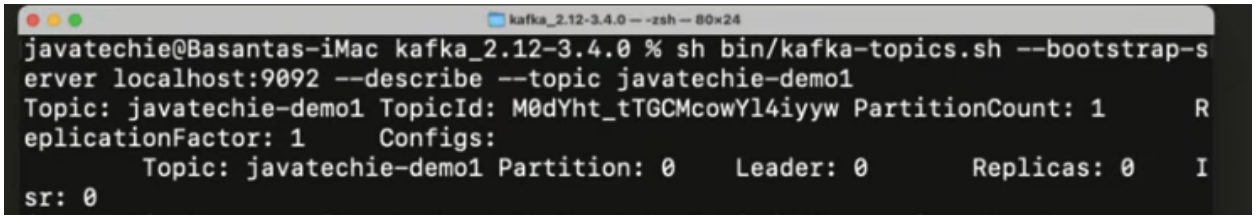
```
KafkaMessagePublisher.java
15
16  public void sendMessageToTopic(String message){
17      CompletableFuture<SendResult<String, Object>> future = template.send( topic: "jav
18      future.whenComplete((result,ex)->{
19          if (ex == null) {
20              System.out.println("Sent message=[" + message +
21                      "] with offset=[" + result.getRecordMetadata().offset() + "]");
22          } else {
23              System.out.println("Unable to send message=[" +
24                      message + "] due to : " + ex.getMessage());
25          }
26      });
27
Problems  Terminal  Jenkins Linter  TODO  Services  Build  Dependencies  Event Log
```

Corresponding Controller class

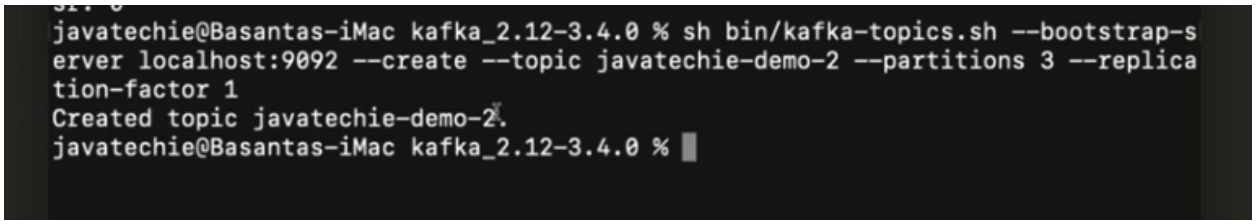
```
KafkaMessagePublisher.java  EventController.java
18
19      @GetMapping("/publish/{message}")
20      public ResponseEntity<?> publishMessage(@PathVariable String message){
21          try {
22              publisher.sendMessageToTopic(message);
23              return ResponseEntity.ok("message published successfully ..");
24          }catch (Exception ex){
25              return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
26                      .build();
27          }
28      }
29  }
```

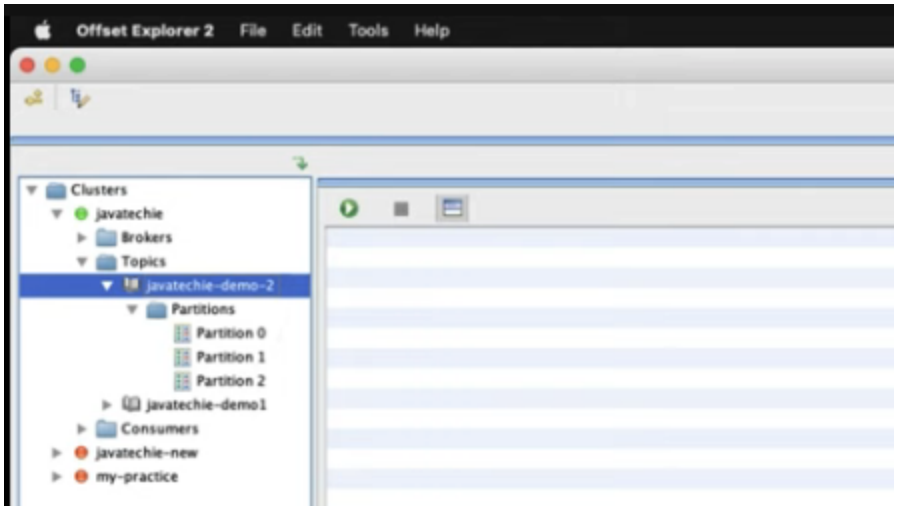






By default java will create only one partition dynamically if you have not taken care of creation of topic and partitions. See the above screenshot. See the below screenshot how to create topic with more number of partitions.

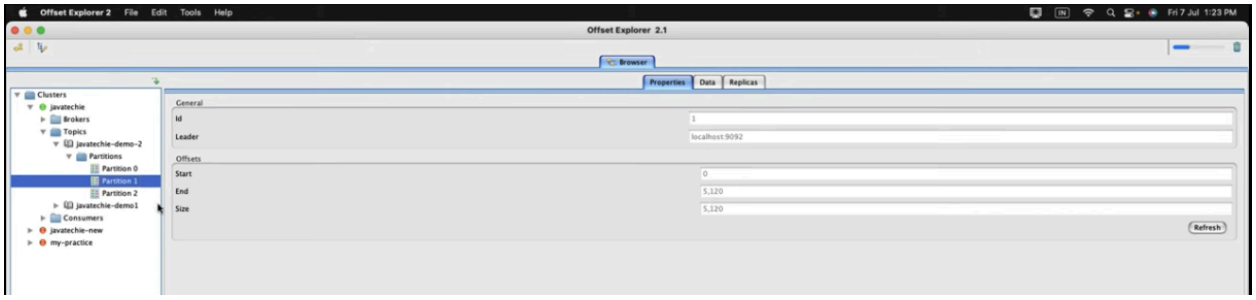




Let us publish 10000 records with small tweak.

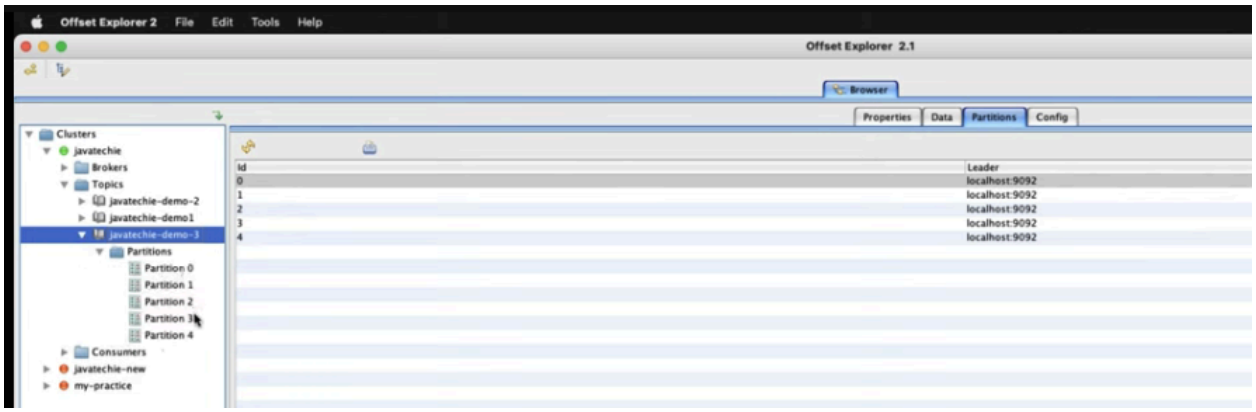
```
@GetMapping("/publish/{message}")
public ResponseEntity<?> publishMessage(@PathVariable String message) {
    try {
        for (int i = 0; i <= 10000; i++) {
            publisher.sendMessageToTopic(message + " : " + i);
        }
        return ResponseEntity.ok("message published successfully ..");
    } catch (Exception ex) {
```

After hitting the endpoint the load will be distributed between partitions by Kafka scheduler based on availability

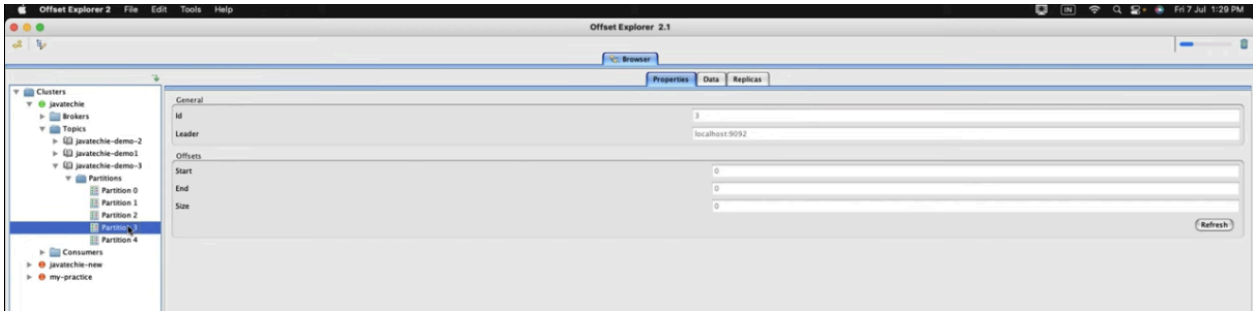


In real time we use the kafka portal for creating topics , partitions => this is industry practice and recommended.

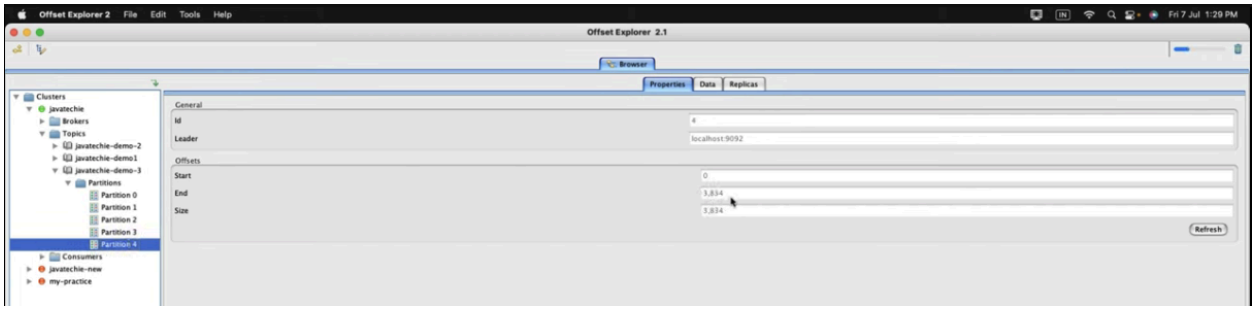
Using springBoot configuration also we can create topic and partition as shown below



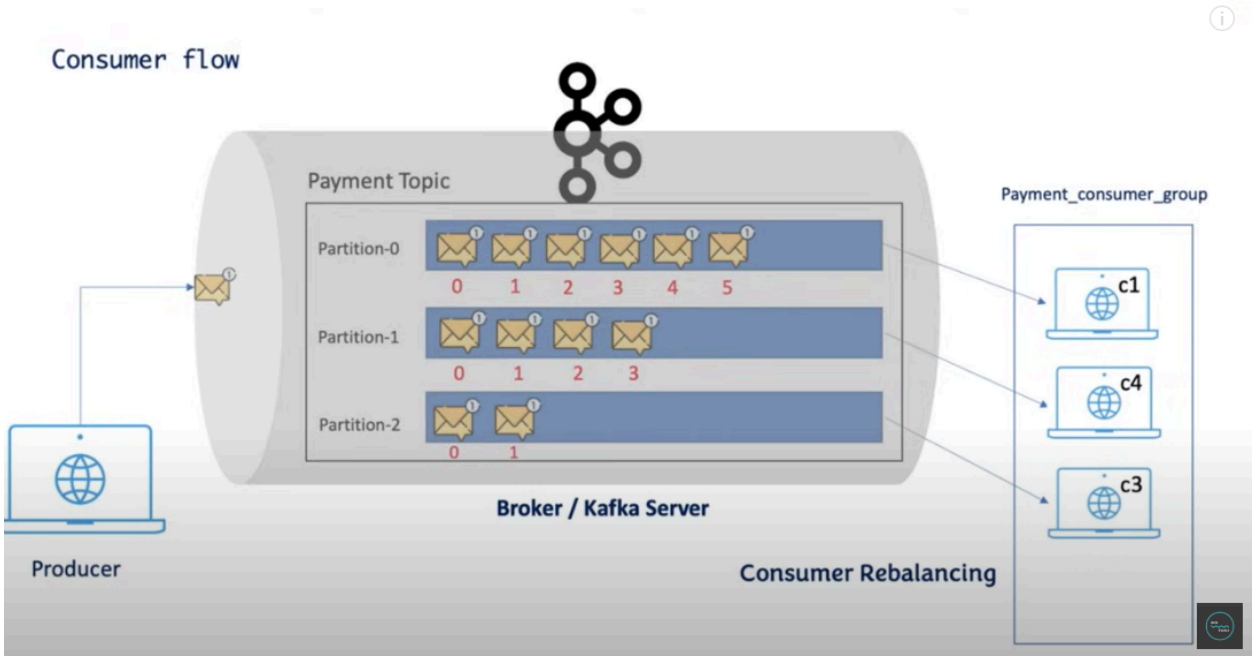
There is a possibility the partition may not receive any messages. That is not in our control. It is in the hands of kafka scheduler



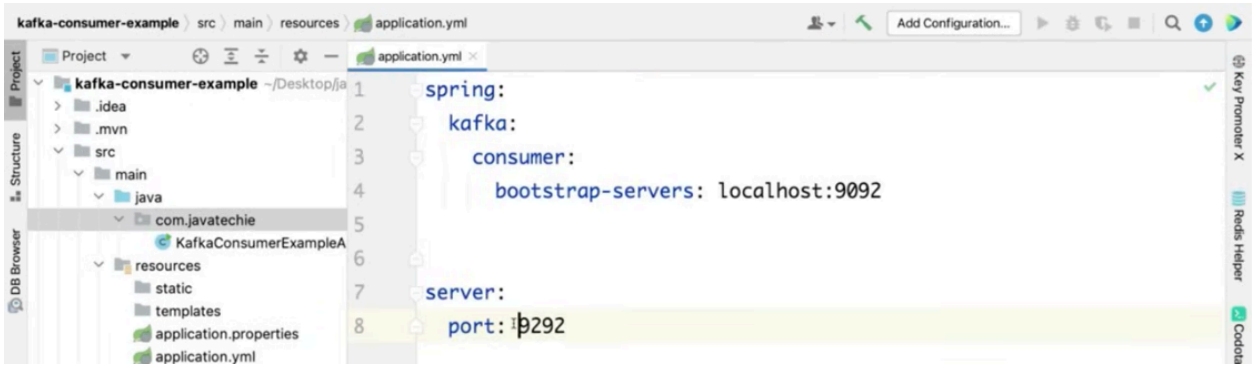
Here partition 3 did not receive any messages but partition 4 received messages.



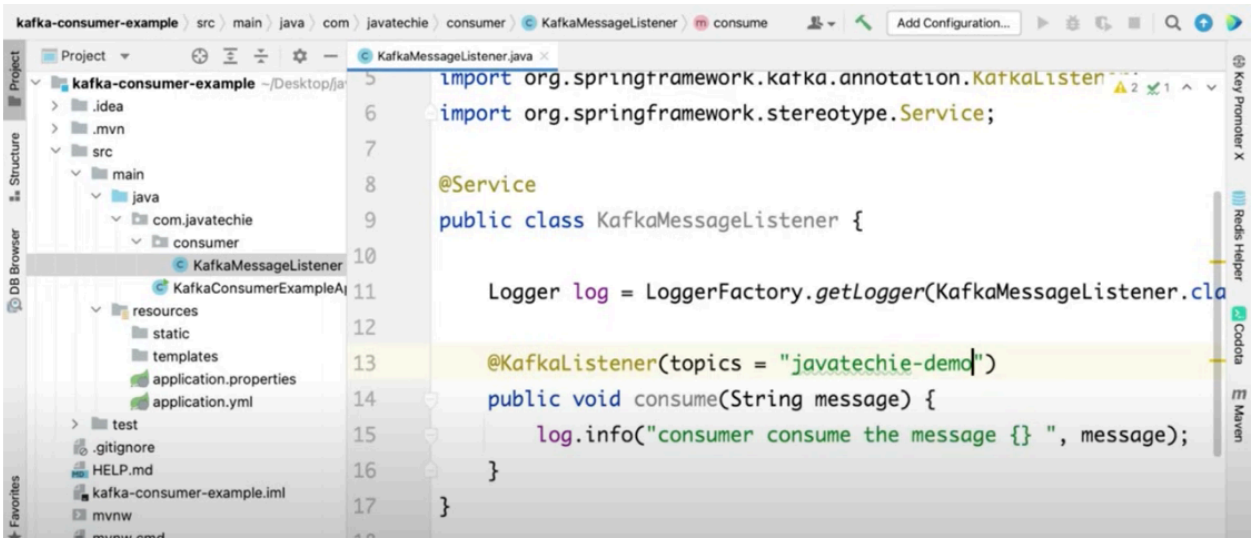
Now let us see the consumer side



Consumer properties

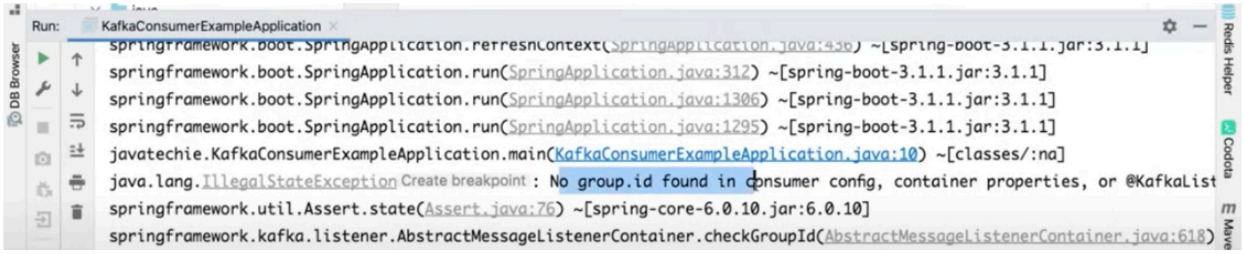


create a listener method



Now start the producer

Now start the consumer. It gave an error as no group id found in the consumer config.



For single consumer instance also require group id. Add to listener

Also need to add in the property as well.

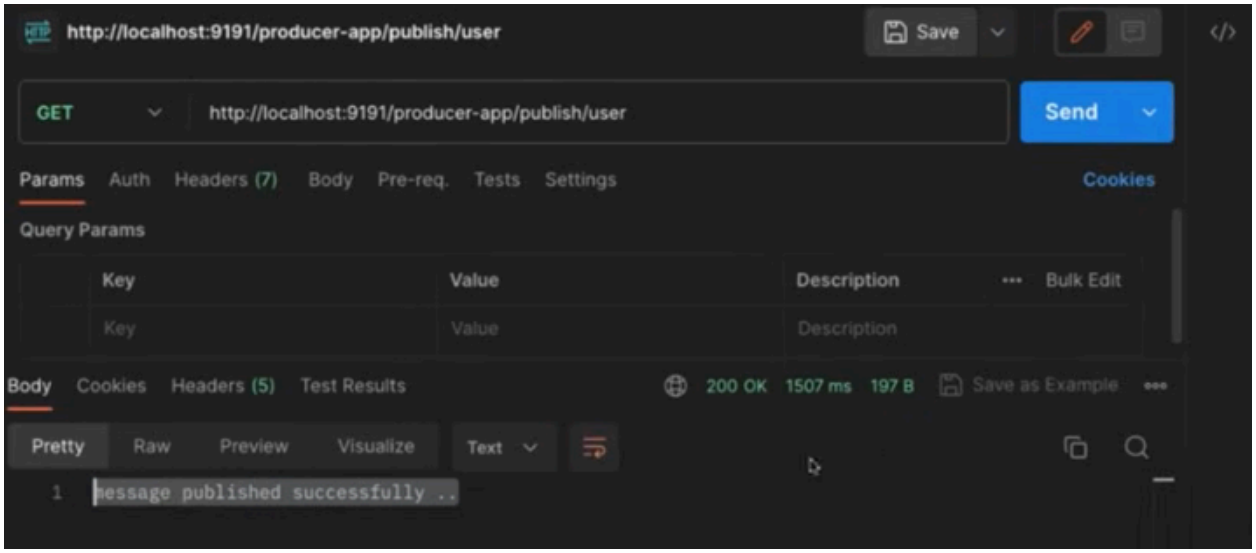
KafkaMessageListener.java | application.yml | KafkaConsumerExampleApplication.java

```yaml
spring:
  kafka:
    consumer:
      bootstrap-servers: localhost:9092
      group-id: jt-group-1

server:
  port: 9292
```

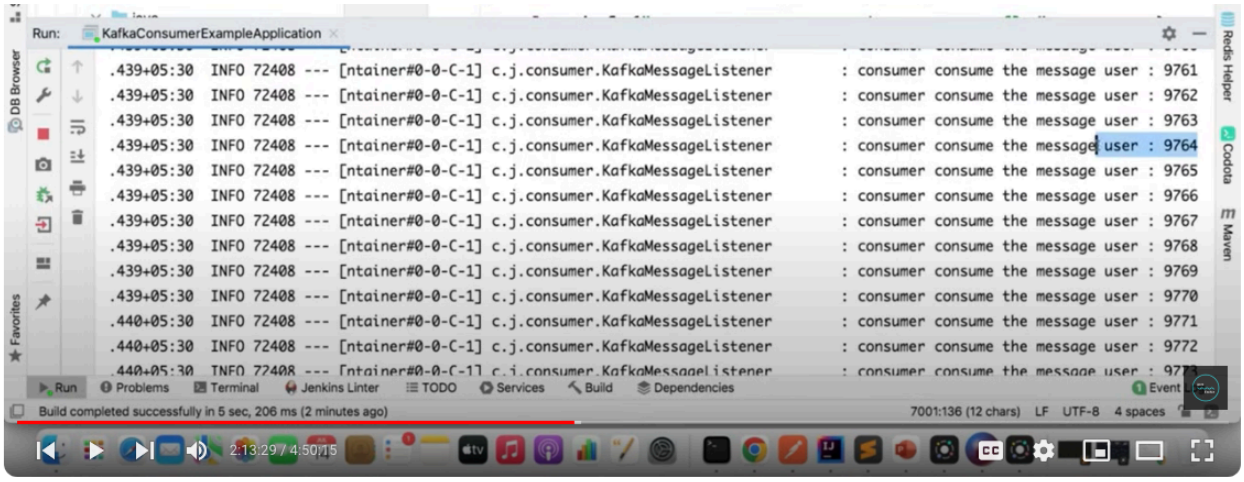
```
sageListenerContainer    : jt-group-1: partitions assigned: [javatechie-de
titions assigned: [javatechie-demo-0, javatechie-demo-1, javatechie-demo-2]
```

All the topics are assigned to consumer group

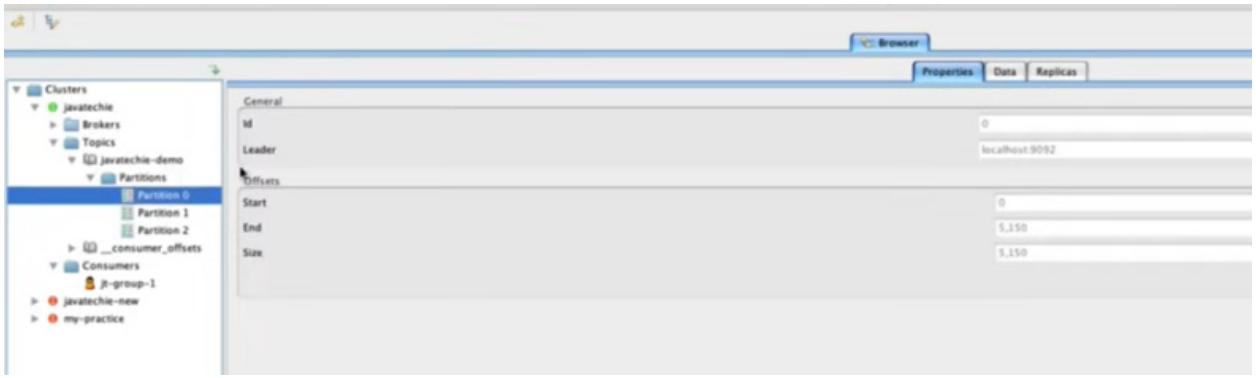
Just hit the producer endpoint



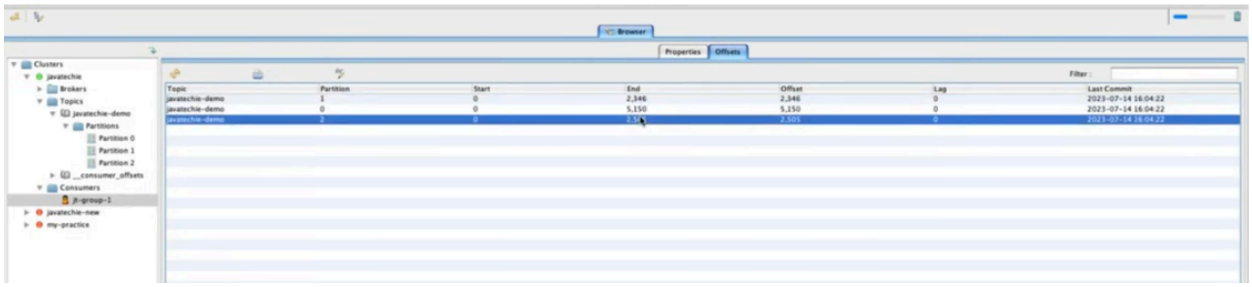
Consumer listens all the messages



Producer - summary



Consumer summary



Notice Log is zero. Which means all the messages were successfully consumed.

Just duplicate consumers. So we have 4 consumers. now

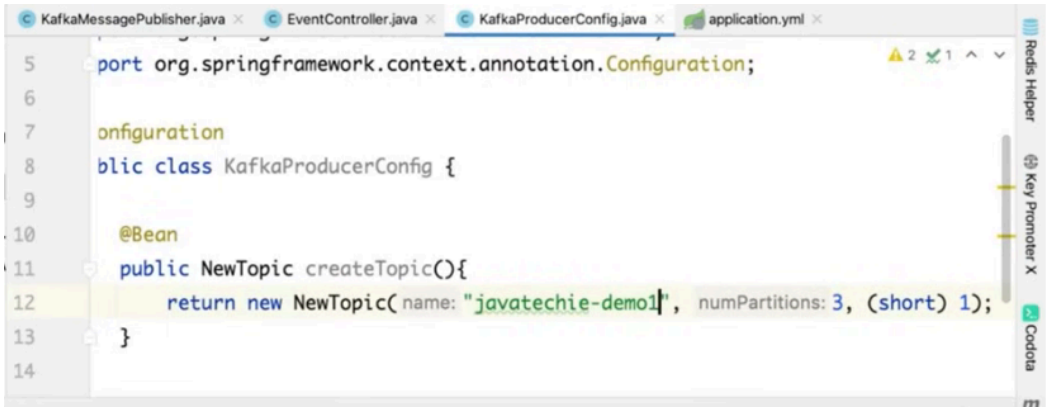
```java
Logger log = LoggerFactory.getLogger(KafkaMessageListener.class);

@KafkaListener(topics = "javatechie-demo",groupId = "jt-group-1")
public void consume1(String message) {
    log.info("consumer1 consume the message {} ", message);
}

@KafkaListener(topics = "javatechie-demo",groupId = "jt-group-1")
public void consume2(String message) {
    log.info("consumer2 consume the message {} ", message);
}
```

```java
}

@KafkaListener(topics = "javatechie-demo",groupId = "jt-group-1")
public void consume3(String message) {
    log.info("consumer3 consume the message {} ", message);
}

@KafkaListener(topics = "javatechie-demo",groupId = "jt-group-1")
public void consume4(String message) {
    log.info("consumer4 consume the message {} ", message);
}
}
```

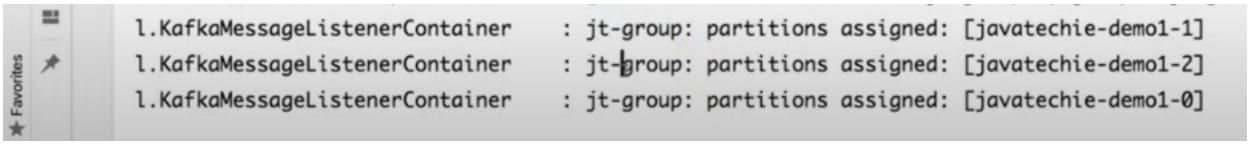
Rename the topics and groupId to clear understanding and start the consumer.

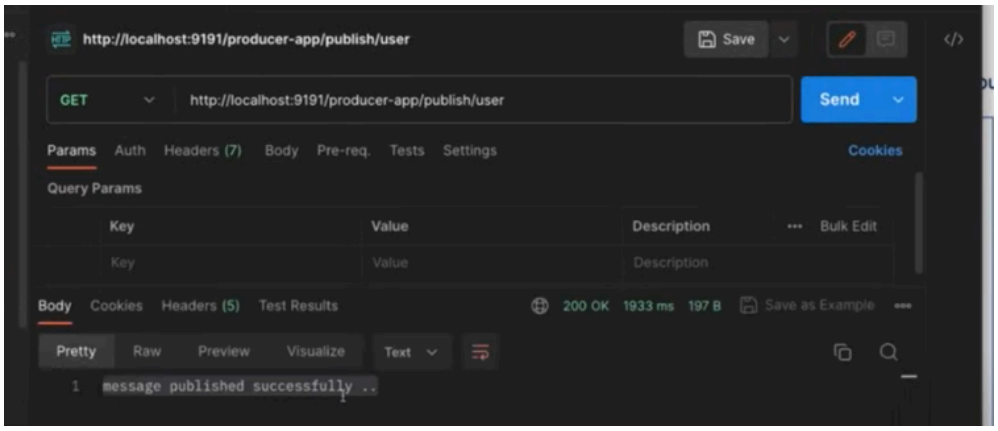
```java
@KafkaListener(topics = "javatechie-demo1",groupId = "jt-group")
public void consume1(String message) {
    log.info("consumer1 consume the message {} ", message);
}

@KafkaListener(topics = "javatechie-demo1",groupId = "jt-group")
public void consume2(String message) {
    log.info("consumer2 consume the message {} ", message);
}

@KafkaListener(topics = "javatechie-demo1",groupId = "jt-group")
public void consume3(String message) {
```

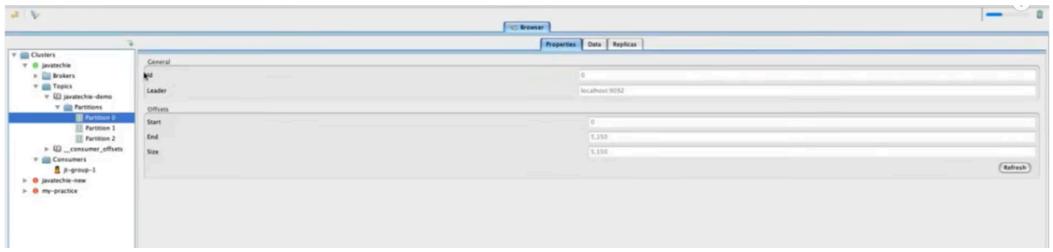
In the producer also updates the topic name to “javatechie-demo1 and start the producer”.

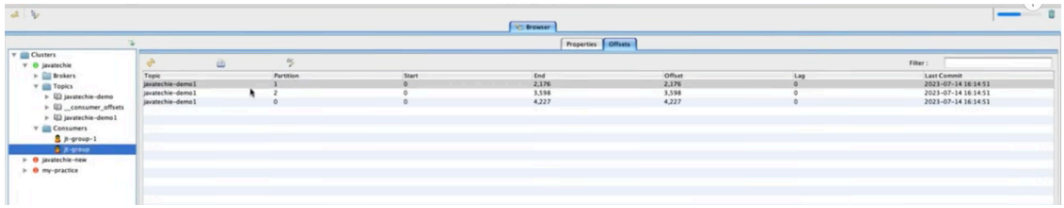


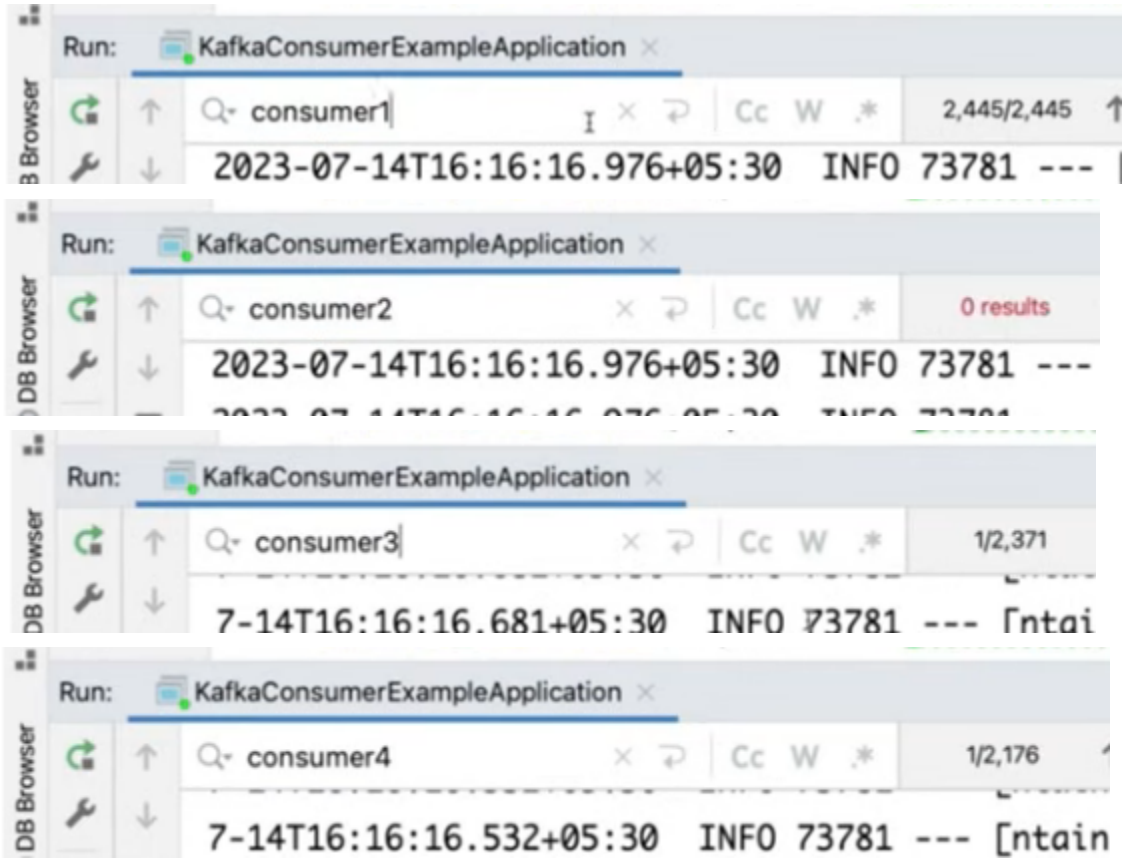
Now we can see each partition assigned to one consumer. Here consumer names are common.







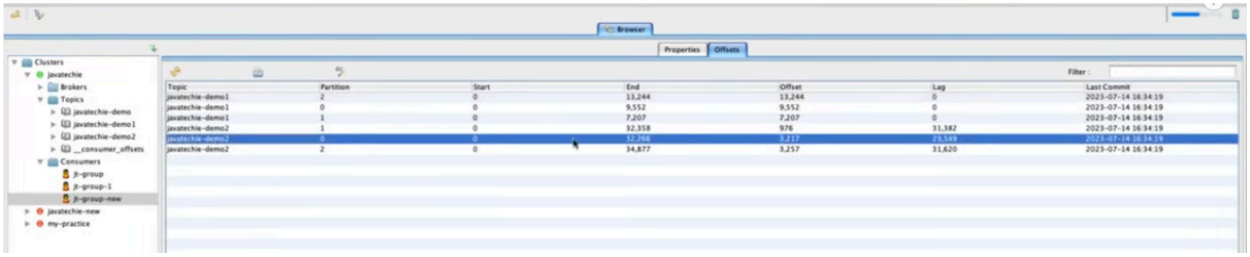




Each partition is assigned to each consumer instance.

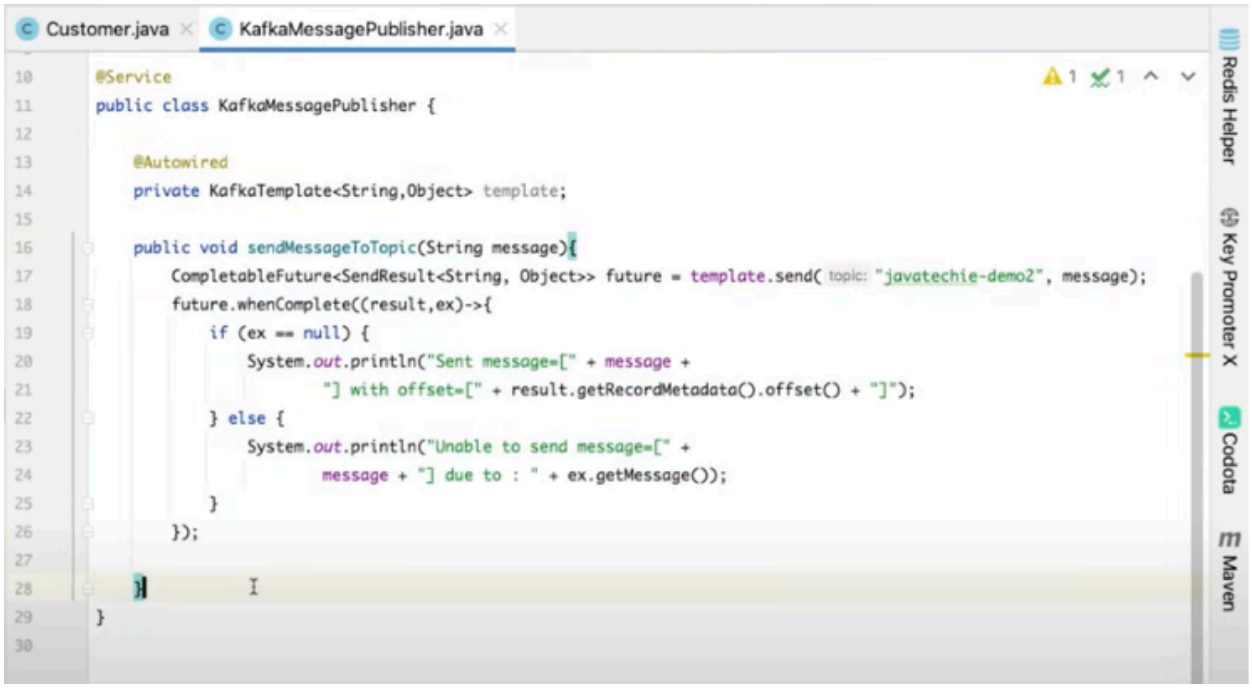
In real time we won't create duplicate consumer method instances
We will create duplicate instances of the application. Need to explore.

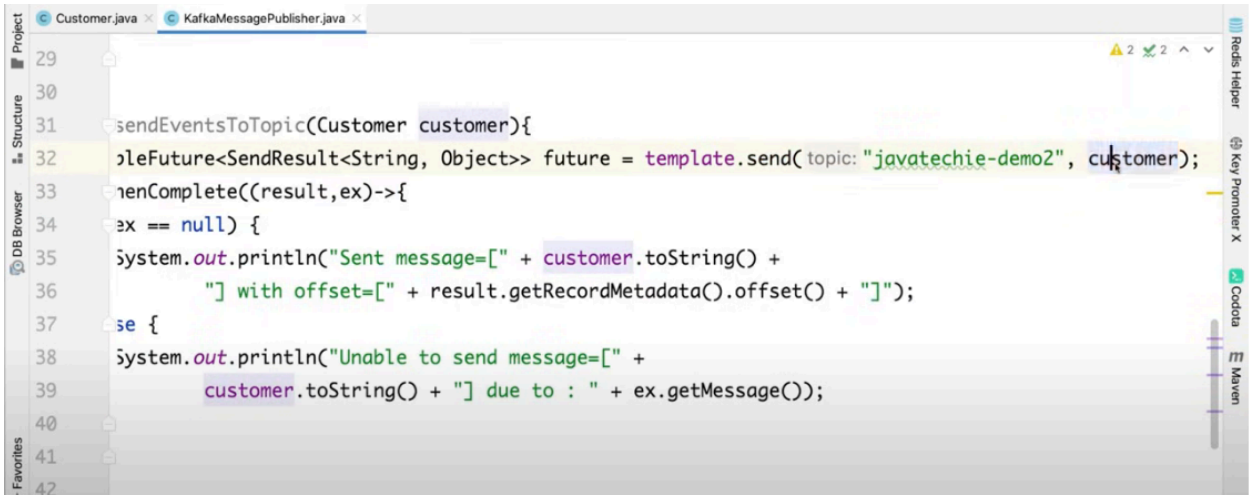
Lag messages.
If the consumer is stopped /went down while messages are flowing.



You may notice lag in consuming the messages as shown above.

Just copy our previous publisher and paste it and replace all String with Customer





And add it in the try catch block

```
Customer.java | KafkaMessagePublisher.java | KafkaProducerConfig.java | EventController.java

34          future.whenComplete((result, ex) -> {
35              if (ex == null) {
36                  System.out.println("Sent message=[" + customer.toString() +
37                          "] with offset=[" + result.getRecordMetadata().offset() + "]");
38              } else {
39                  System.out.println("Unable to send message=[" +
40                          customer.toString() + "] due to : " + ex.getMessage());
41              }
42          });
43
44      } catch (Exception ex) {
45          System.out.println("ERROR : "+ ex.getMessage());
46      }
47  }
```

Create controller

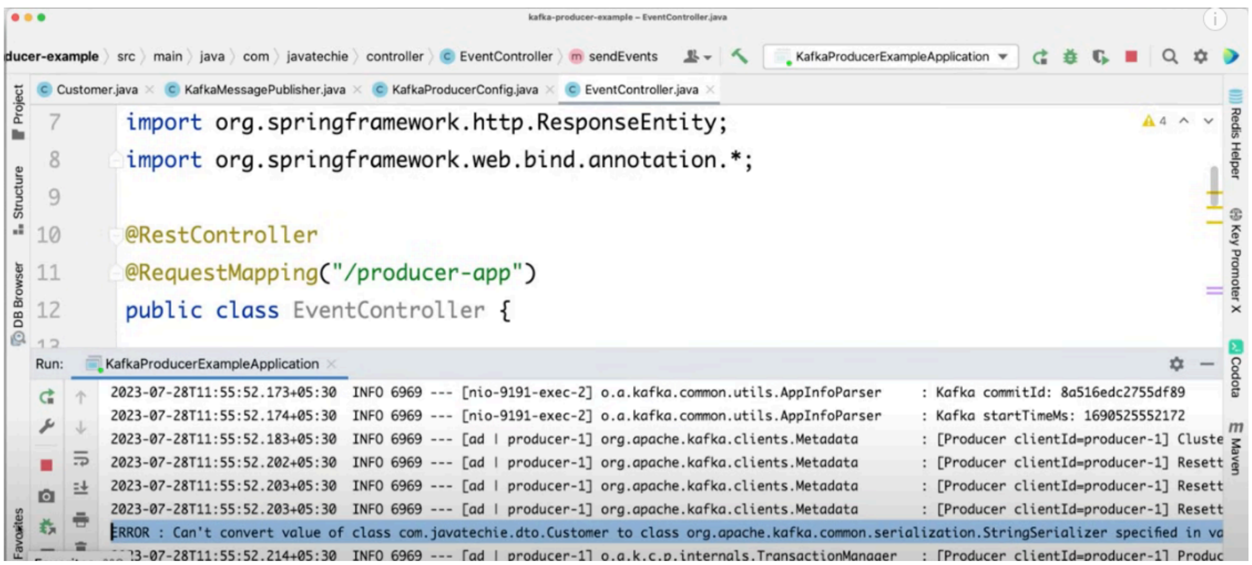
```
30      @PostMapping("/publish")
31      public void sendEvents(@RequestBody Customer customer) {
32          publisher.sendEventsToTopic(customer);
33      }
```

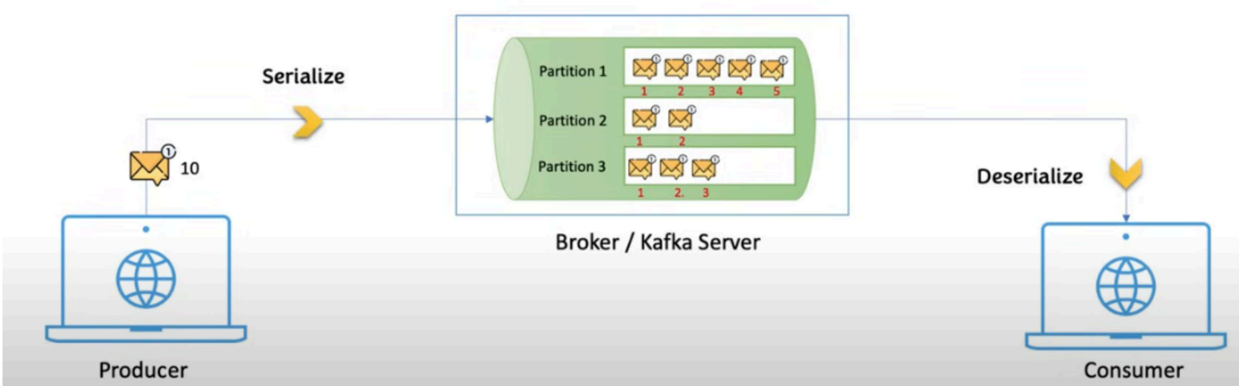
Create a consumer to listen to customer messages in the consumer app.
Create customer class in the consumer too and start .

```
KafkaMessageListener.java | Customer.java | pom.xml (kafka-consumer-example) | application.yml | KafkaConsumerExampleApplication.java

9   @Service
10  public class KafkaMessageListener {
11
12      Logger log = LoggerFactory.getLogger(KafkaMessageListener.class);
13
14      @KafkaListener(topics = "javatechie-demo",groupId = "jt-group-new")
15  @   public void consume(Customer customer) {
16          log.info("consumer consume the message {} ", customer.toString());
17      }
```

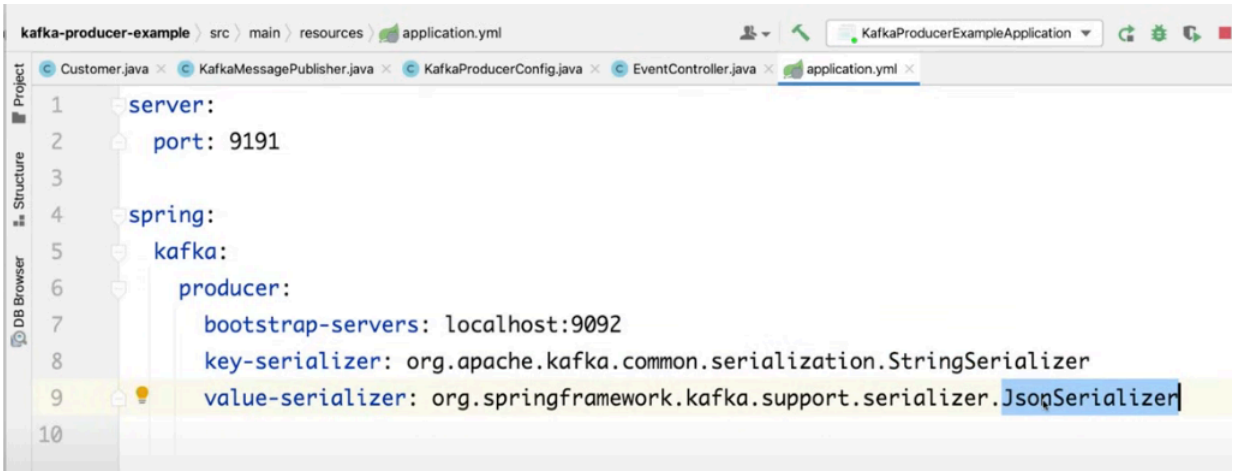
Rename group id to jt-group

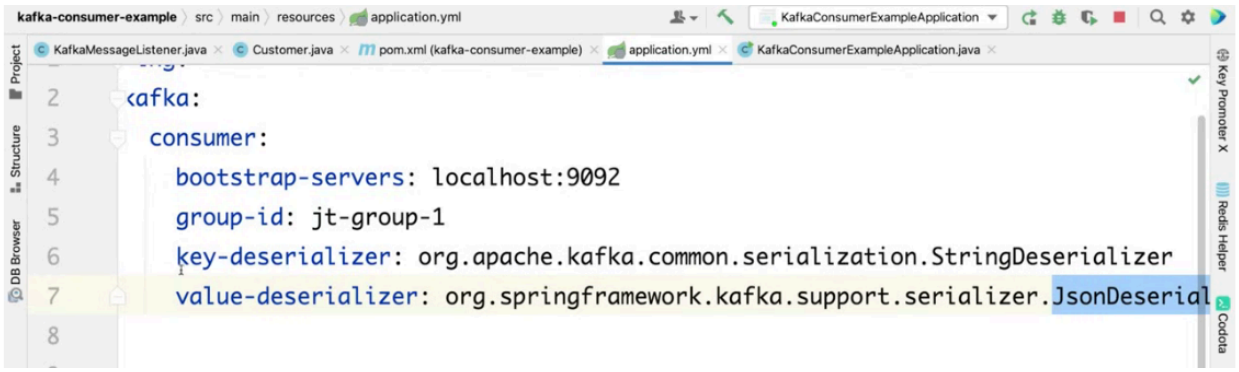
Up the producer and hit the endpoint. We will get serialization error.



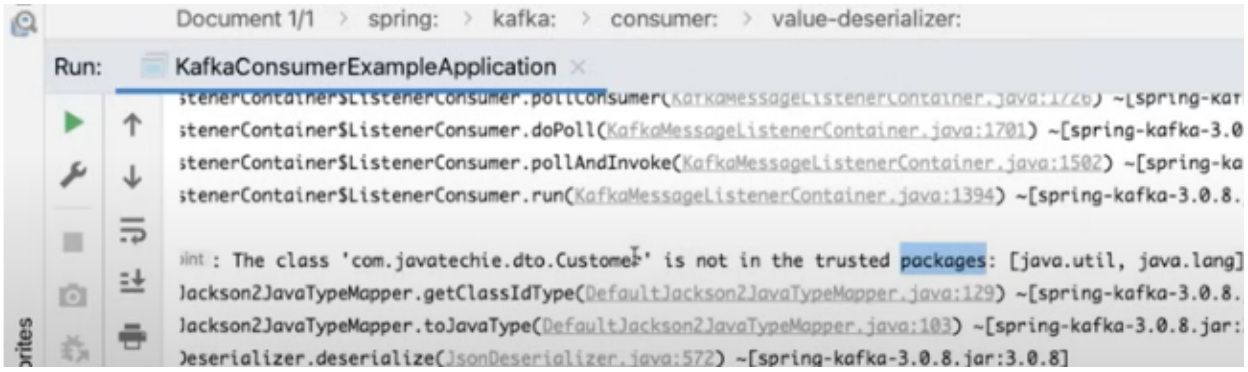


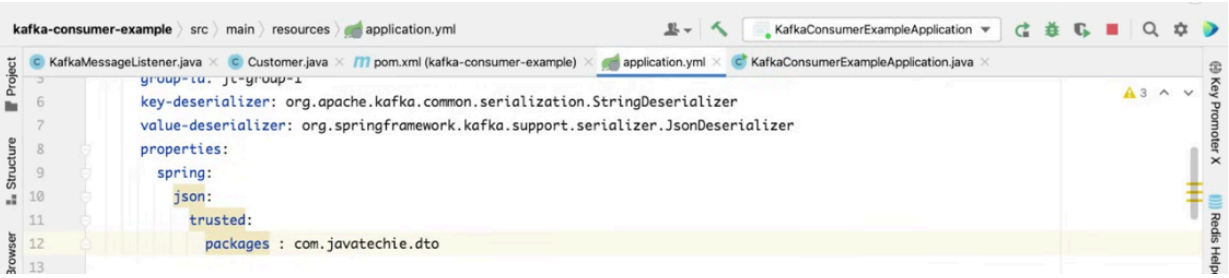
From the producer we have to serialize and the consumer side we have to deserialize data. It is a common practice we have to do when we send data across network.



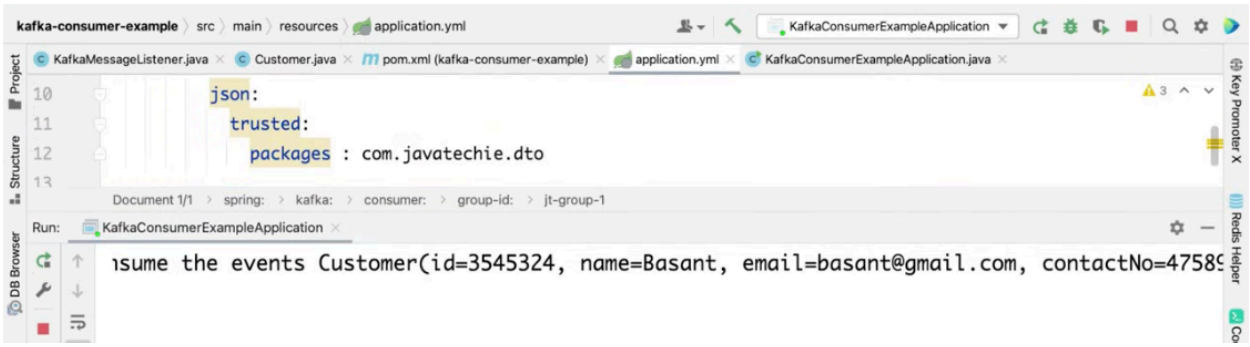


Still we are seeing error as the object is not from the trusted package.

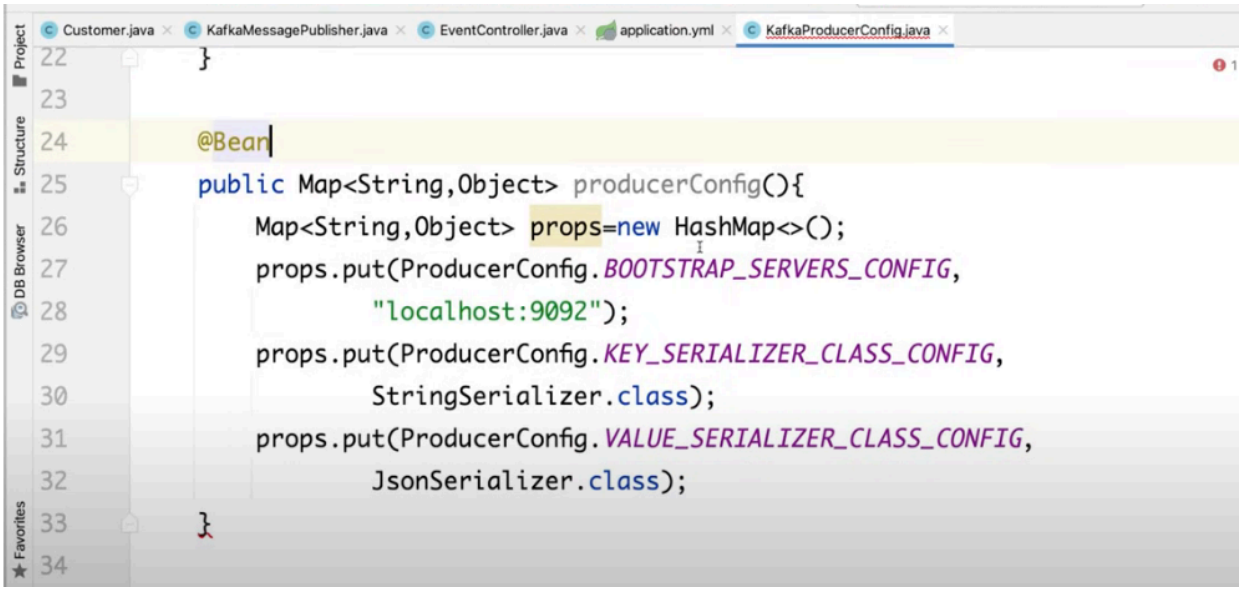




Now consumer



But this is not best practice. Comment those properties in both producer and consumer.
Refer to below java configuration for the best practice.
The below one is for producer



Same thing for consumer side as well.

```java
}

@Bean
public ProducerFactory<String,Object> producerFactory(){
    return new DefaultKafkaProducerFactory<>(producerConfig());
}

@Bean
public KafkaTemplate<String,Object> kafkaTemplate(){
    return new KafkaTemplate<>(producerFactory());
}
```

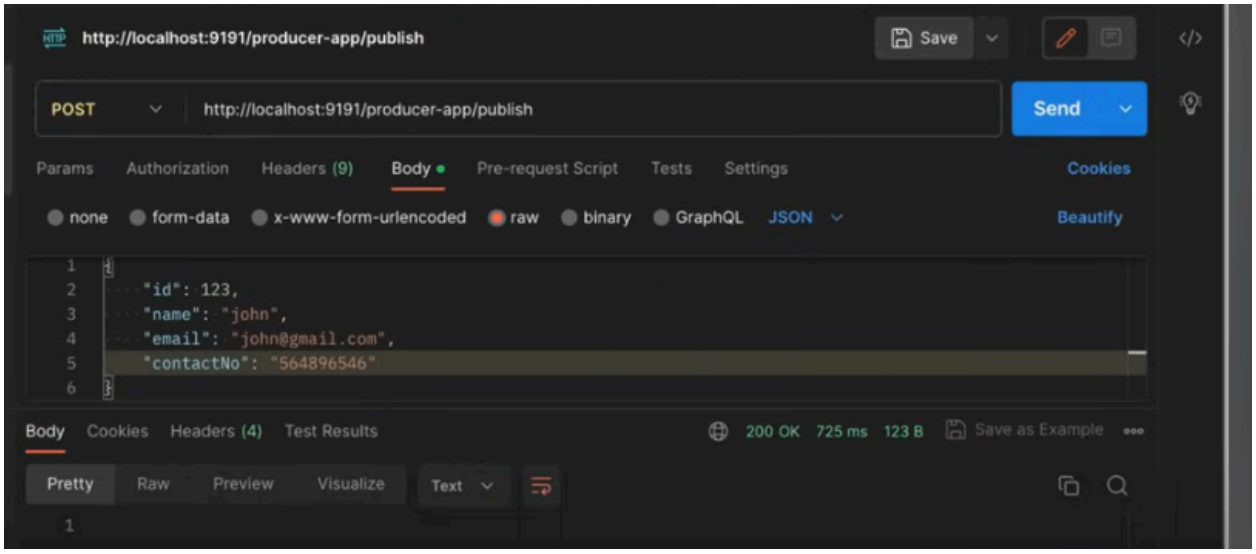
Same thing for consumer side

```java
@Bean
public Map<String, Object> consumerConfig() {
    Map<String, Object> props = new HashMap<>();
    props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
            "localhost:9092");
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
            StringDeserializer.class);
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
            JsonDeserializer.class);
    props.put(JsonDeserializer.TRUSTED_PACKAGES, "com.javatechie.dto");
    return props;
}
}
```

```java
@Bean
public ConsumerFactory<String,Object> consumerFactory(){
    return new DefaultKafkaConsumerFactory<>(consumerConfig());
}

@Bean
public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String, Object>> kafkaListenerContainer
    ConcurrentKafkaListenerContainerFactory<String, Object> factory =
            new ConcurrentKafkaListenerContainerFactory<>();
    factory.setConsumerFactory(consumerFactory());
    return factory;
}
```

Way java config :
If our approach is straight forward - push the event and consume it go for application properties or yaml file.

If you want more customization like to access secure kafka cluster which will be https in that case we have configure certificates using java based config.



producer

Consumer

Now refer

Control Kafka Partion.pdf file