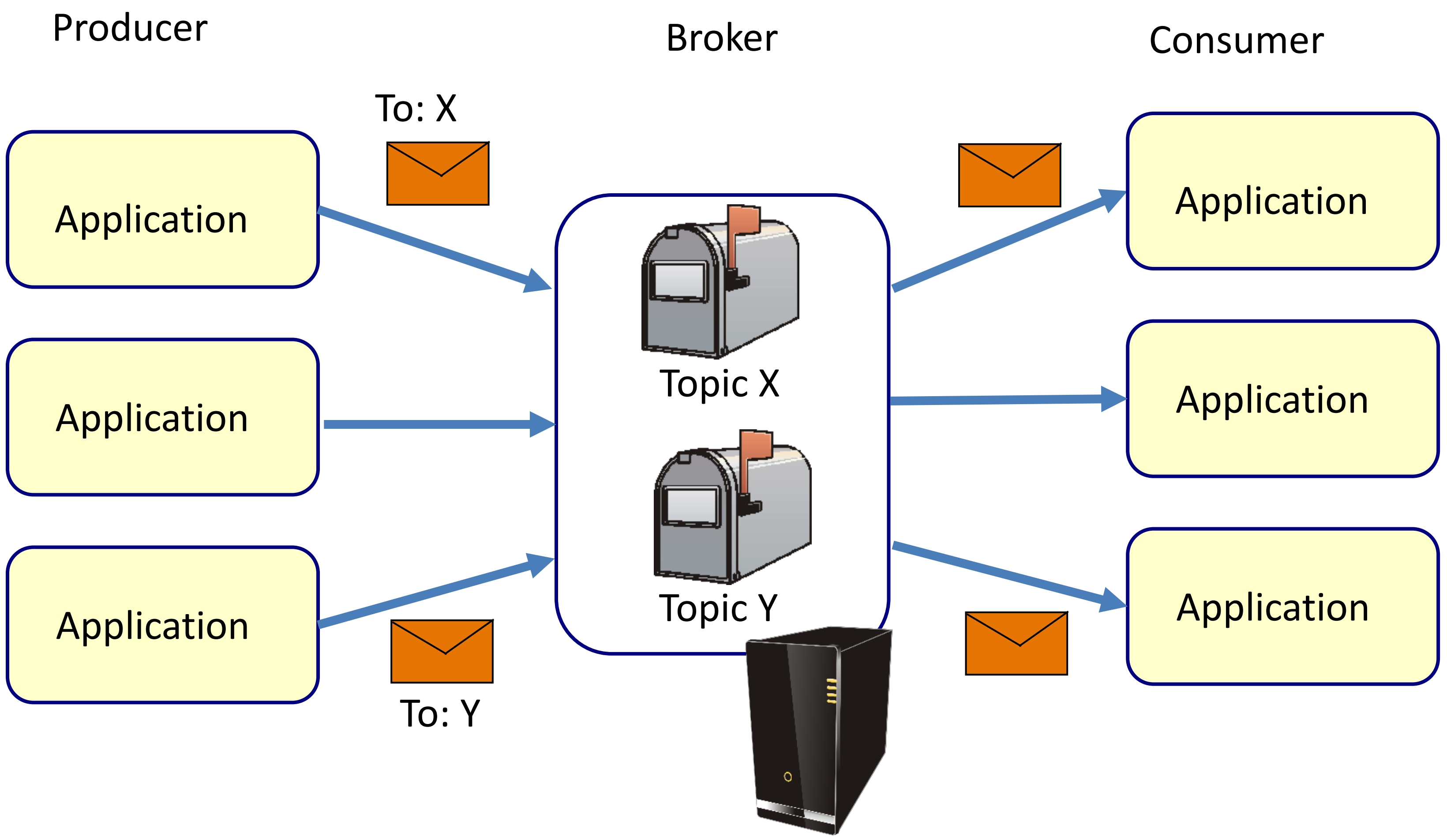
Lesson 11

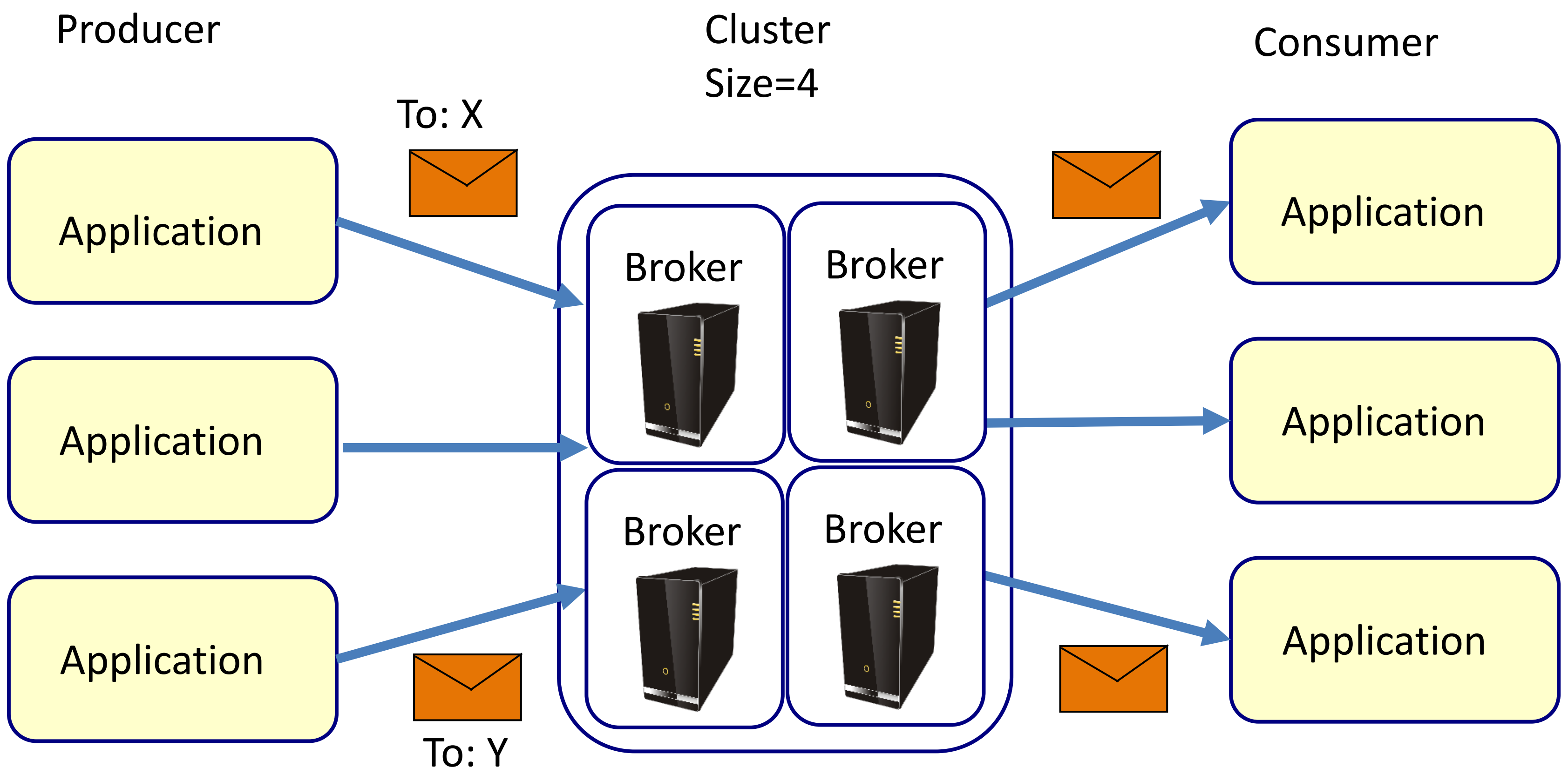
KAFKA

KAFKA BASICS

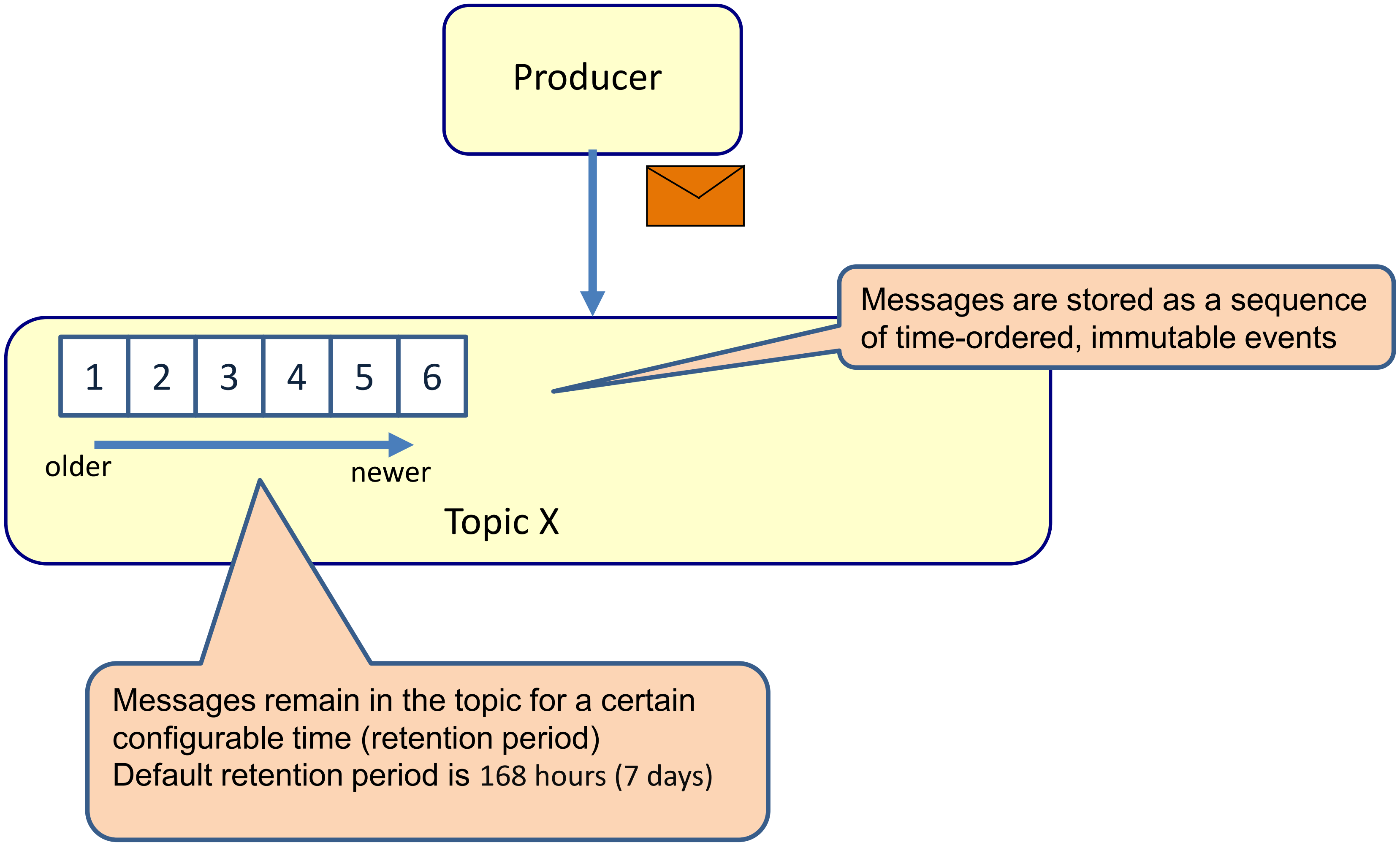
Kafka


Producer
Broker
Consumer
To: X
Application
Application
Application
Topic X
Topic Y
To: Y
Application
Application
Application

Cluster of Brokers



Event sourcing



Messages are stored as a sequence of time-ordered, immutable events

Messages remain in the topic for a certain configurable time (retention period)
Default retention period is 168 hours (7 days)

Message

```
{
    "Timestamp": "2025-09-02T09:36:14.532+00:00",
    "Topic": "topicZ",
    "Partition": 0,
    "Offset": 1,
    "SchemaId": null,
    "SchemaType": null,
    "Key": [
      107,
      101,
      121
    ],
    "Headers": null,
    "Message": "Hello World"
}
```

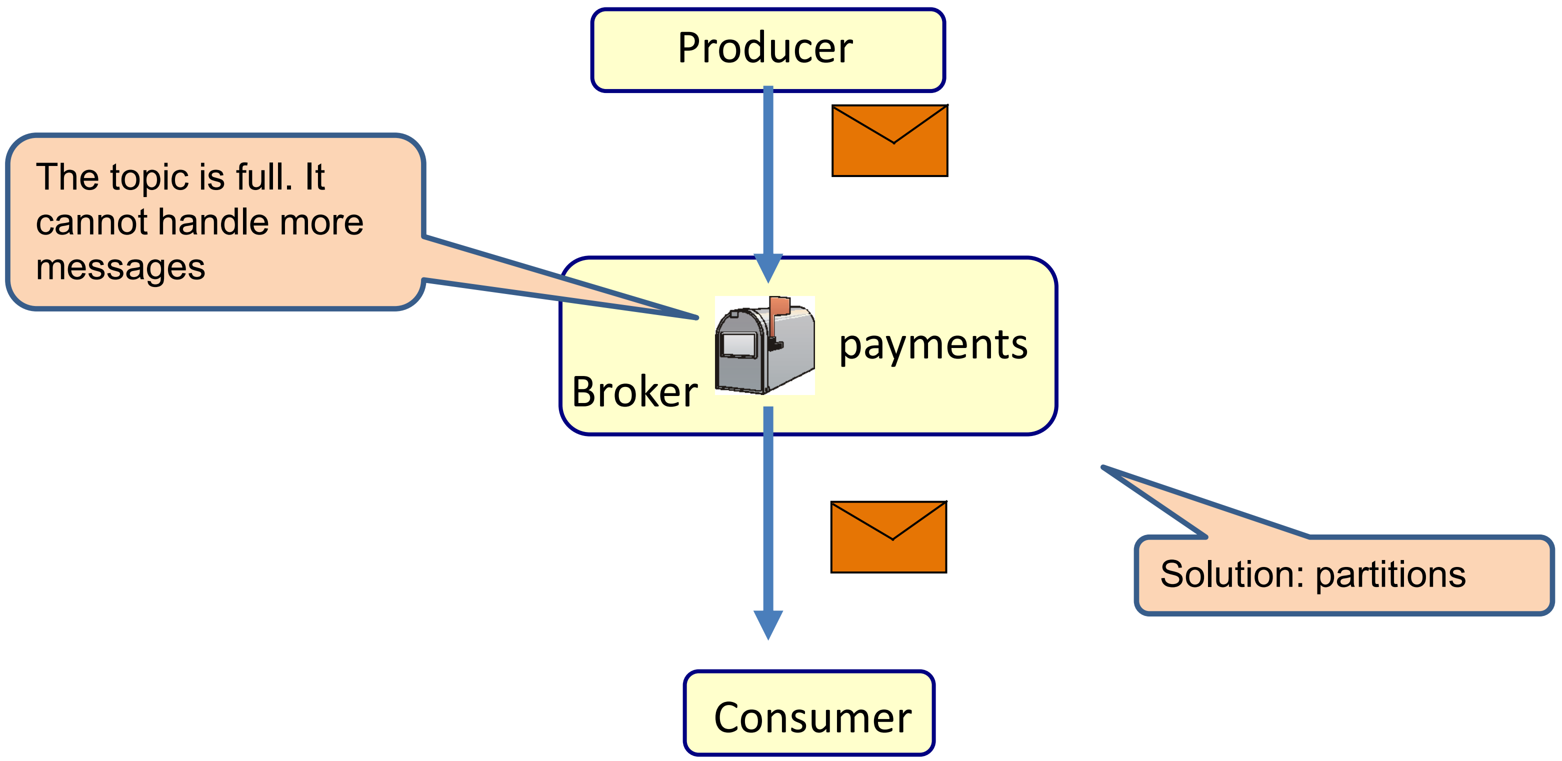
Set when Kafka receives the message

Key (optional)

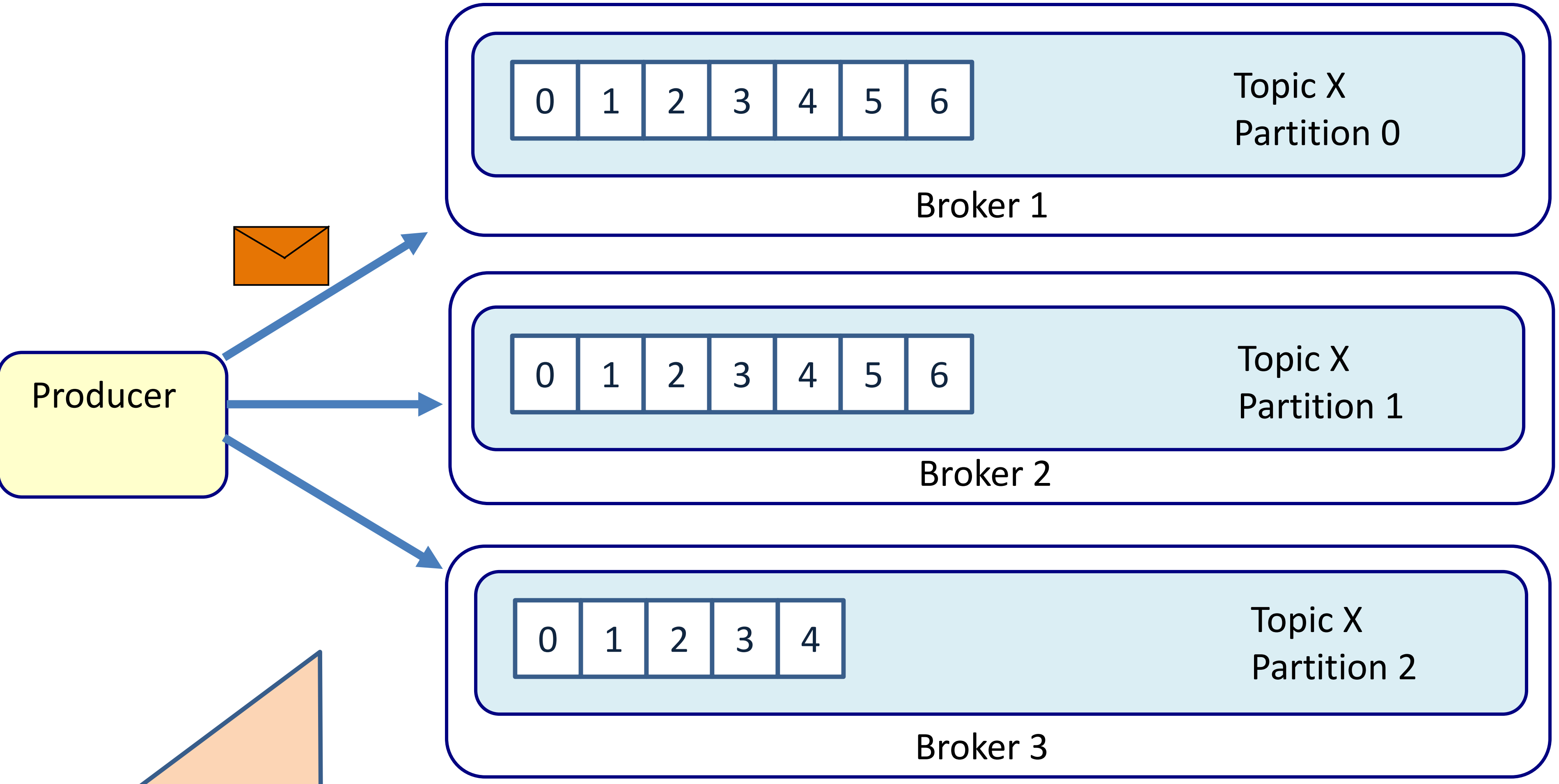
Headers(optional)

Content of the message

What if the topic gets too full?

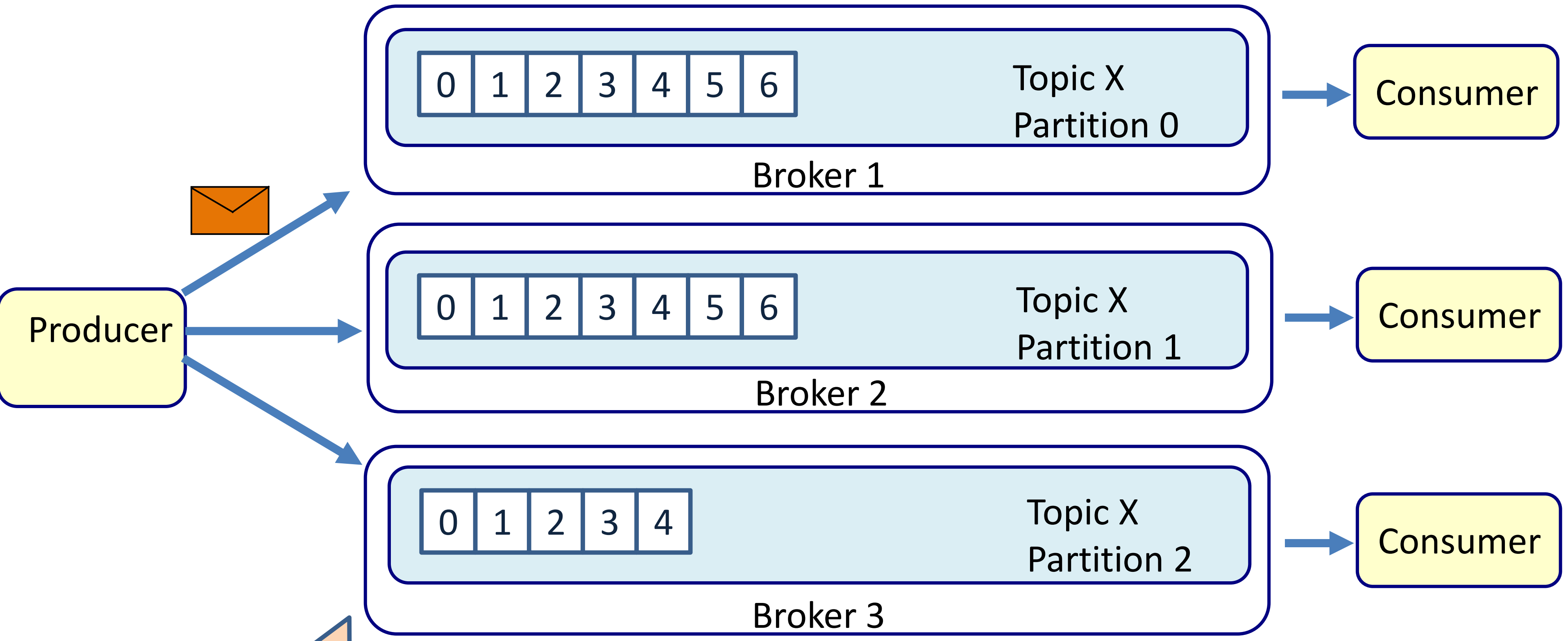


Scale out partitions



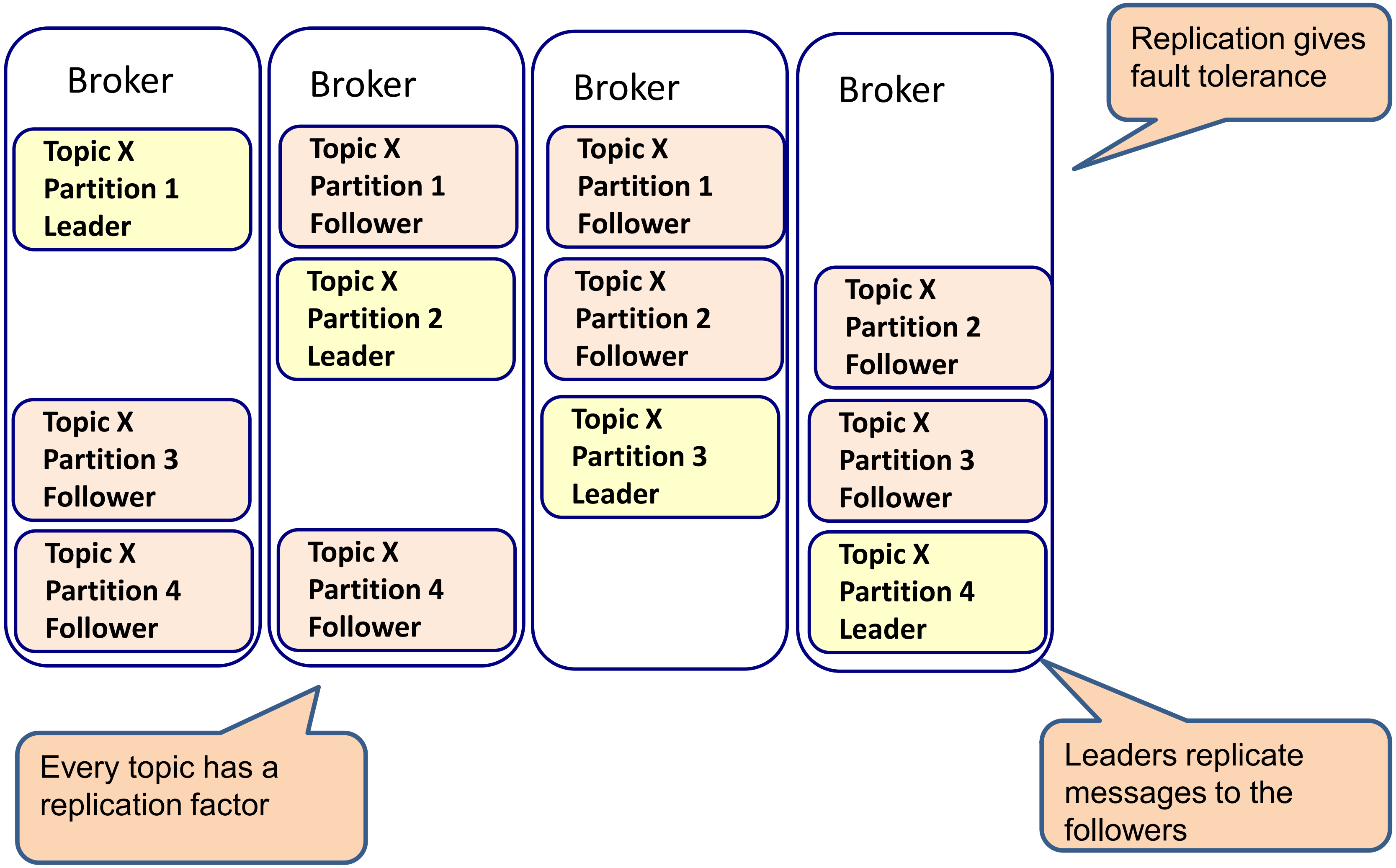
Increases
- Scalability (number of messages and their size)
- Higher levels of throughput (parallelism)

Scale out partitions



Increases
- Scalability (number of messages and their size)
- Higher levels of throughput (parallelism)

Replication


Broker
Topic X Partition 1 Leader
Topic X Partition 3 Follower
Topic X Partition 4 Follower
Broker
Topic X Partition 1 Follower
Topic X Partition 2 Leader
Topic X Partition 4 Follower
Broker
Topic X Partition 1 Follower
Topic X Partition 2 Follower
Topic X Partition 3 Leader
Broker
Topic X Partition 2 Follower
Topic X Partition 3 Follower
Topic X Partition 4 Leader
Replication gives fault tolerance
Every topic has a replication factor
Leaders replicate messages to the followers

SPRING BOOT KAFKA

Kafka producer

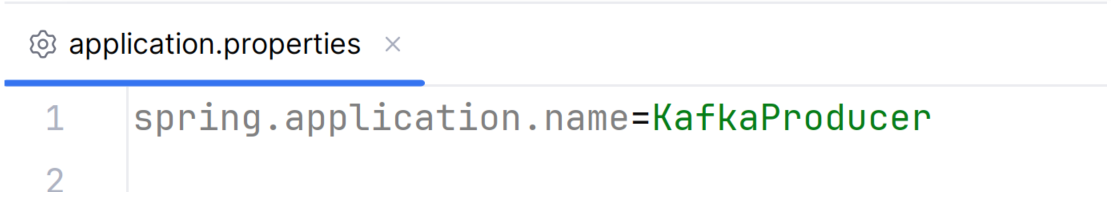
```
@Service
public class KafkaProducer {

  @Autowired
  private KafkaTemplate<String, String> kafkaTemplate;

  public void sendMessage() {
    kafkaTemplate.send("topic1", "Hello World");
  }
}
```

The topic will automatically be created if it does not exist

Spring makes a KafkaTemplate with the following defaults:
Server = localhost:9092
Offset = latest

Kafka producer

```java
@SpringBootApplication
public class KafkaProducerApplication implements CommandLineRunner {
    @Autowired
    KafkaProducer kafkaProducer;

    public static void main(String[] args) {
      SpringApplication.run(KafkaProducerApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
      kafkaProducer.sendMessage();
    }
}
```

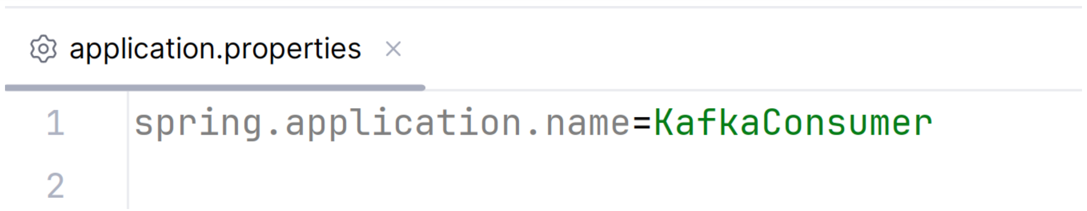


```
spring.application.name=KafkaProducer

```

Kafka consumer

```java
@Service
public class KafkaConsumer {

  @KafkaListener(topics = "topic1", groupId = "gid1")
  public void consume(String message) {
    System.out.println(message);
  }
}
```

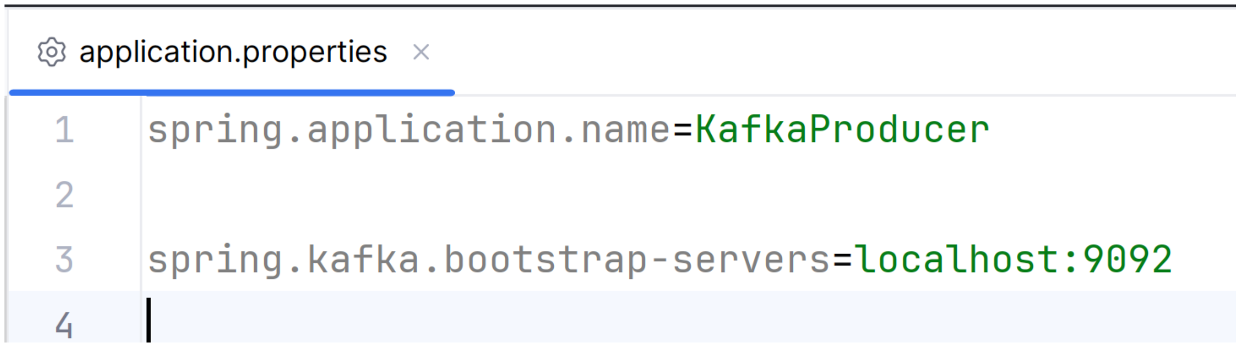
```java
@SpringBootApplication
public class KafkaConsumerApplication {

  public static void main(String[] args) {
    SpringApplication.run(KafkaConsumerApplication.class, args);
  }
}
```



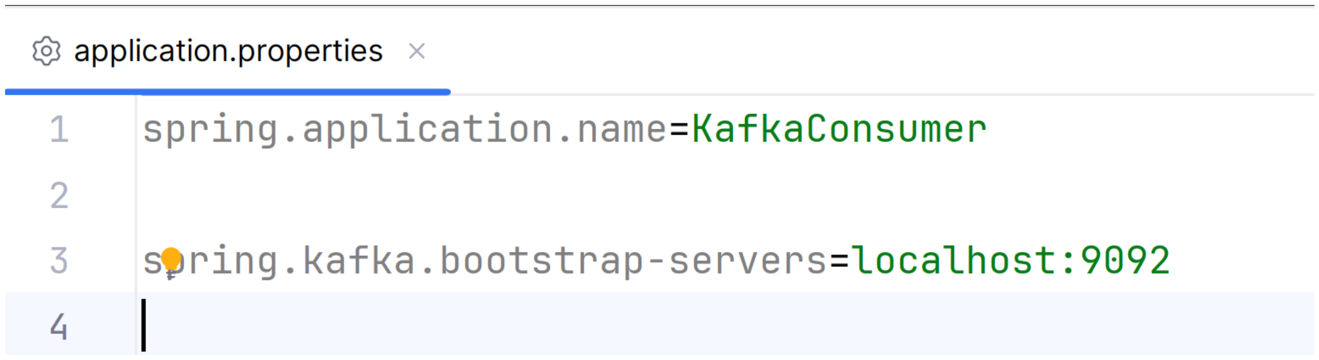
application.properties

```
spring.application.name=KafkaConsumer

```

Set the Kafka server


application.properties

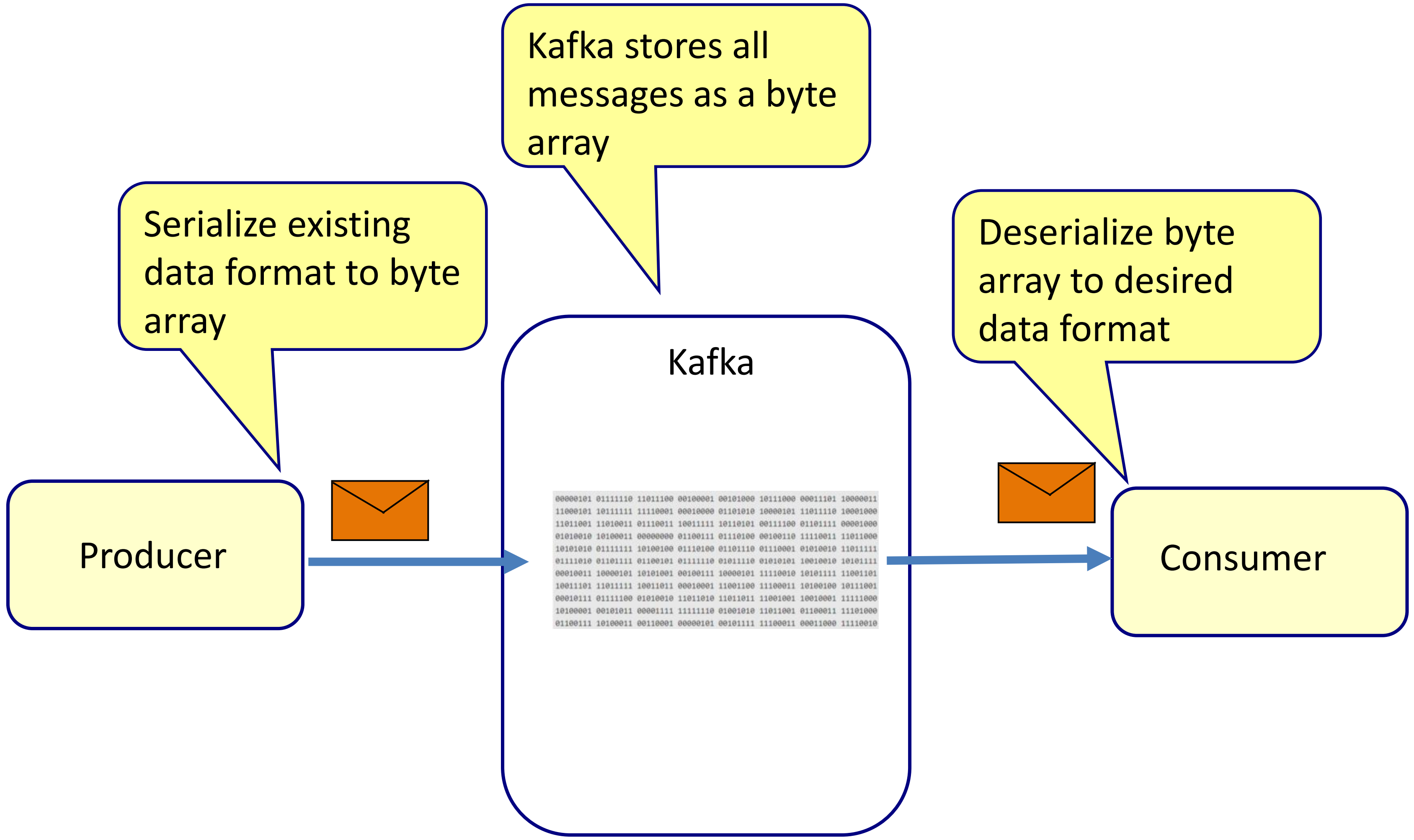
```
spring.application.name=KafkaProducer

spring.kafka.bootstrap-servers=localhost:9092

```


application.properties

```
spring.application.name=KafkaConsumer

spring.kafka.bootstrap-servers=localhost:9092

```

SERIALIZATION AND DESERIALIZATION

Serialization/deserialization


Kafka stores all messages as a byte array
Serialize existing data format to byte array
Deserialize byte array to desired data format
Kafka
Producer
Consumer

Sending an Object

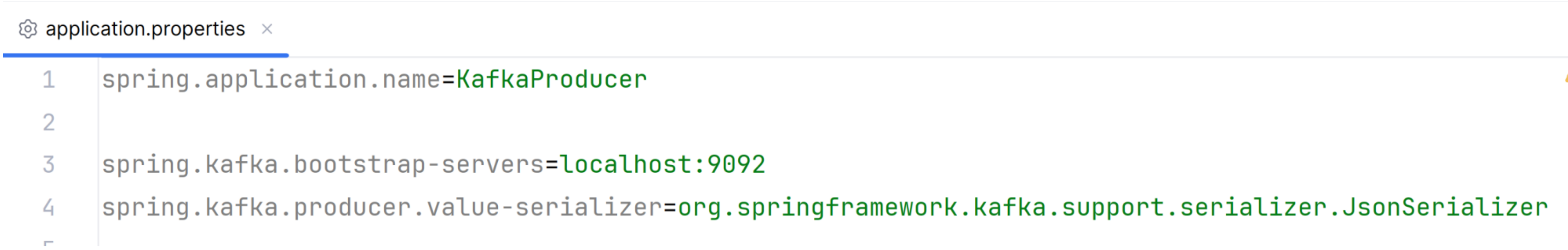
```java
package producer;

public class Product {
    private String productNumber;
    private String name;
    private double price;
```

Package producer

```java
@Service
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, Object> kafkaTemplate;

    public void sendMessage() {
        Product product = new Product("A158", "IPhone13", 180.0);
        kafkaTemplate.send("topic1", product);
    }
}
```

Object

Sending an Object

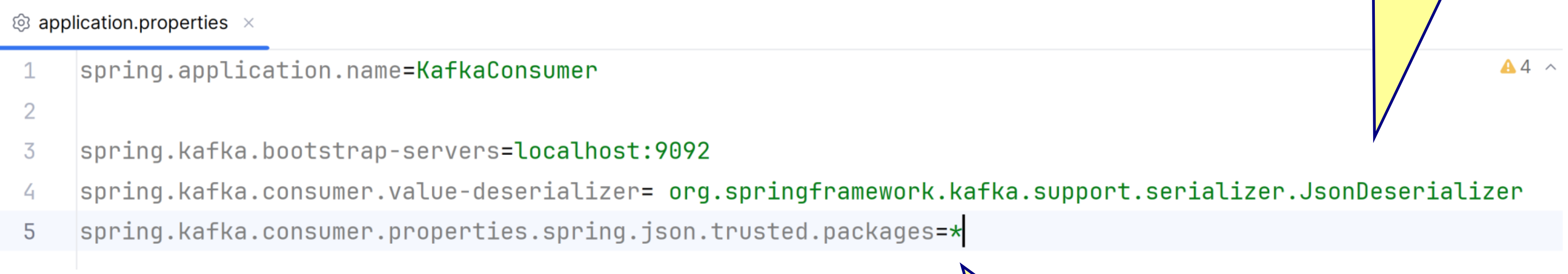

application.properties

```
spring.application.name=KafkaProducer

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
```

```
{
  "Timestamp": "2025-08-23T10:42:42.199+00:00",
  "Topic": "topic1",
  "Partition": 0,
  "Offset": 0,
  "SchemaId": null,
  "SchemaType": null,
  "Key": null,
  "Headers": {
   "__TypeId__": "producer.Product"
  },
  "Message": {
   "productNumber": "A159",
   "name": "IPhone13",
   "price": 180
  }
 }
```

Kafka consumer

JsonDeserializer


application.properties

```
spring.application.name=KafkaConsumer

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.trusted.packages=*
```

Only classes in trusted packages can be used to deserialize the received JSON to an object

Kafka consumer

```
package consumer;

public class Product {
   private String productNumber;
   private String name;
   private double price;
```

Package **consumer**

```
@Service
public class KafkaConsumer {

   @KafkaListener(topics = "topic1", groupId = "gid1")
   public void consume(Product product) {
      System.out.println(product);
   }
}
```

The consume() method is not called because we receive a **producer**.Product

Kafka consumer

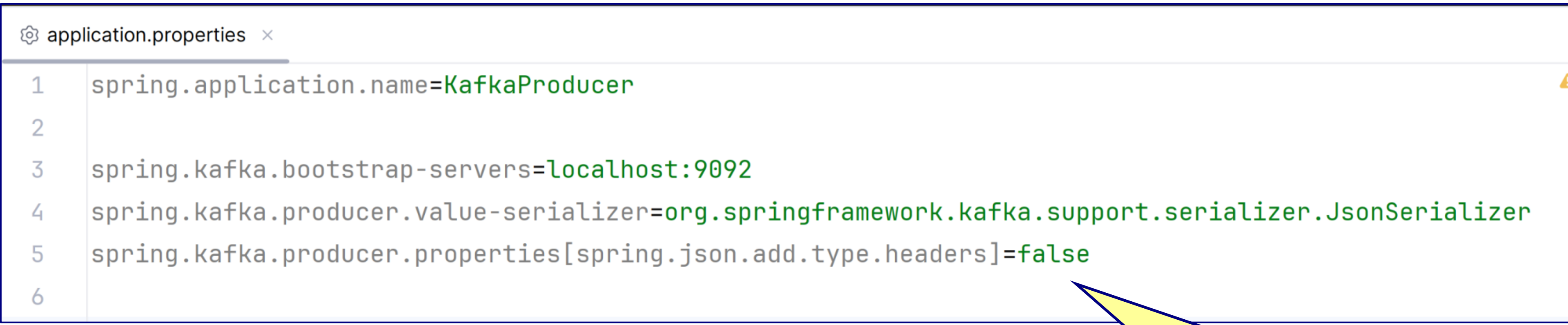
```
package consumer;

public class Product {
   private String productNumber;
   private String name;
   private double price;
```

Package **consumer**

```
{
   "Timestamp": "2025-08-23T10:42:42.199+00:00",
   "Topic": "topic1",
   "Partition": 0,
   "Offset": 0,
   "SchemaId": null,
   "SchemaType": null,
   "Key": null,
   "Headers": {
     "__TypeId__": "producer.Product"
   },
   "Message": {
     "productNumber": "A159",
     "name": "IPhone13",
     "price": 180
   }
 }
```

We receive a **producer**.Product but we have only a **consumer**.Product. This means Spring will not call the listener method.

Possible solution: Producer



application.properties

```
spring.application.name=KafkaProducer

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.producer.properties[spring.json.add.type.headers]=false

```

Do not add the class type in the message.

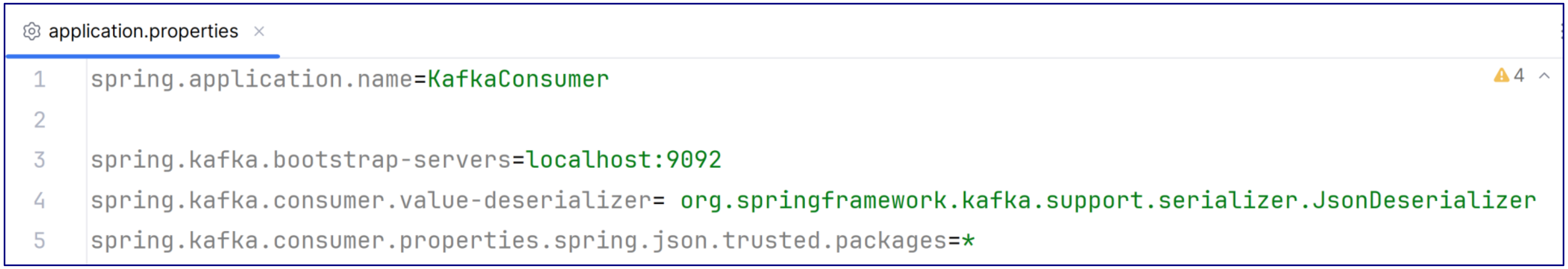
```
{
  "Timestamp": "2025-08-23T10:57:35.607+00:00",
  "Topic": "topic1",
  "Partition": 0,
  "Offset": 1,
  "SchemaId": null,
  "SchemaType": null,
  "Key": null,
  "Headers": null,
  "Message": {
   "productNumber": "A159",
   "name": "IPhone13",
   "price": 180
  }
 }
```

No class type in the message.

Possible solution: Consumer

Specify the type Spring needs to deserialize the received JSON to

```
@Service
public class KafkaConsumer {

  @KafkaListener(topics = "topic1", groupId = "gid1", properties =
                  {"spring.json.value.default.type=consumer.Product"})
  public void consume(Product product) {
    System.out.println(product);
  }
}
```


application.properties

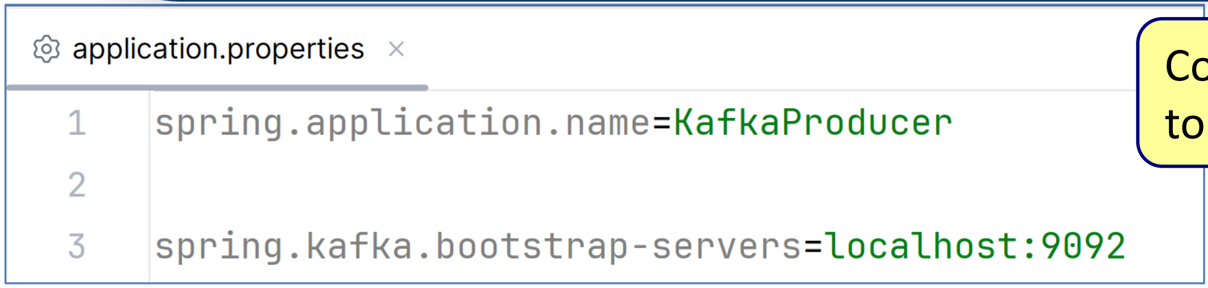
```
spring.application.name=KafkaConsumer

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.value-deserializer= org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.properties.spring.json.trusted.packages=*
```

Better solution: Producer

```
@Service
public class KafkaProducer {

   @Autowired
   private KafkaTemplate<String, String> kafkaTemplate;

   public void sendMessage() throws JsonProcessingException {
      Product product = new Product("A159", "IPhone13", 180.0);
      ObjectMapper objectMapper = new ObjectMapper();
      String productAsString = objectMapper.writeValueAsString(product);
      kafkaTemplate.send("topic1", productAsString);
   }
}
```

Always send a String

Convert the object to a JSON string



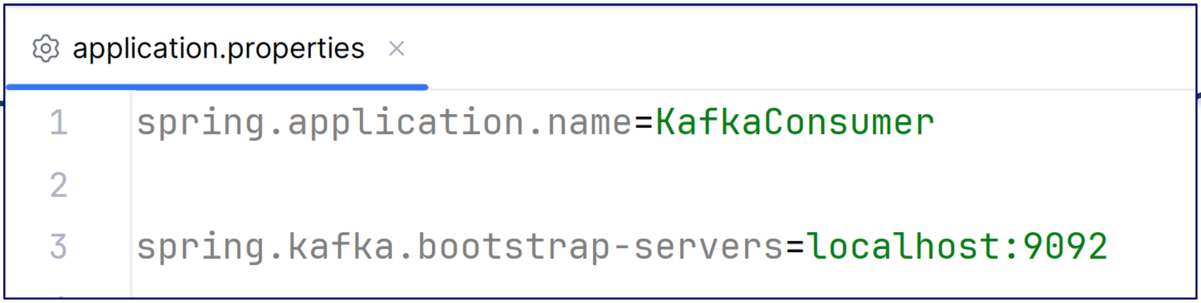
```
spring.application.name=KafkaProducer

spring.kafka.bootstrap-servers=localhost:9092
```

Better solution: Consumer

```
@Service
public class KafkaConsumer {

  @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(String productAsString) {
      ObjectMapper objectMapper = new ObjectMapper();
      try {
        Product product = objectMapper.readValue(productAsString, Product.class);
        System.out.println("Kafka receiver received message:" + product);
      } catch (IOException e) {
        System.out.println("Kafka receiver: Cannot convert : " + productAsString+" to a
            Product object");
      }
    }
}
```

Always receive a String

Convert the JSON string to an object



application.properties

```
spring.application.name=KafkaConsumer

spring.kafka.bootstrap-servers=localhost:9092
```

JAVA CONFIG

Producer configuration

Configure Kafka in Java instead of applications.properties

```
@Configuration
public class KafkaProducerConfig {

   @Bean
   public ProducerFactory<String, String> producerFactory() {
      Map<String, Object> configProps = new HashMap<>();
      configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
      configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
      configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                StringSerializer.class);
      return new DefaultKafkaProducerFactory<>(configProps);
   }

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

Consumer configuration

```
@Configuration
public class KafkaConsumerConfig {

   @Bean
   public ConsumerFactory<String, String> consumerFactory() {
      Map<String, Object> configProps = new HashMap<>();
      configProps.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
      configProps.put(ConsumerConfig.GROUP_ID_CONFIG, "group_id");
      configProps.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
      configProps.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
      return new DefaultKafkaConsumerFactory<>(configProps);
   }

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

Configure Kafka in Java instead of applications.properties

HEADERS

Custom headers: producer

```
public class KafkaProducer {

  @Autowired
  private KafkaTemplate<String, String> kafkaTemplate;

  public void sendMessage() {
    Message<String> message = MessageBuilder.withPayload("Hello World")
          .setHeader(KafkaHeaders.TOPIC, "topic1")
          .setHeader("MyCustomHeader","HeaderValue")
          .build();
    kafkaTemplate.send(message);
  }
}
```

Add a custom header

Custom header

```
{
    "Timestamp": "2025-08-25T08:58:34.829+00:00",
    "Topic": "topic1",
    "Partition": 0,
    "Offset": 9,
    "SchemaId": null,
    "SchemaType": null,
    "Key": null,
    "Headers": {
      "MyCustomHeader": "HeaderValue",
      "spring_json_header_types": "{\"MyCustomHeader\":\"java.lang.String\"}"
    },
    "Message": "Hello World"
 }
```

Custom header.

Custom headers: consumer

```
@Service
public class KafkaConsumer {

  @KafkaListener(topics = "topic1", groupId = "gid1")
  public void consume(@Payload String message, @Header(name =
                  "MyCustomHeader", required = false) String customHeaderValue) {
    System.out.println("Message header MyCustomHeader = "+customHeaderValue);
    System.out.println("Message payload = "+message);
  }
}
```

```
Message header MyCustomHeader = HeaderValue
Message payload = Hello World
```

Receiving a Message

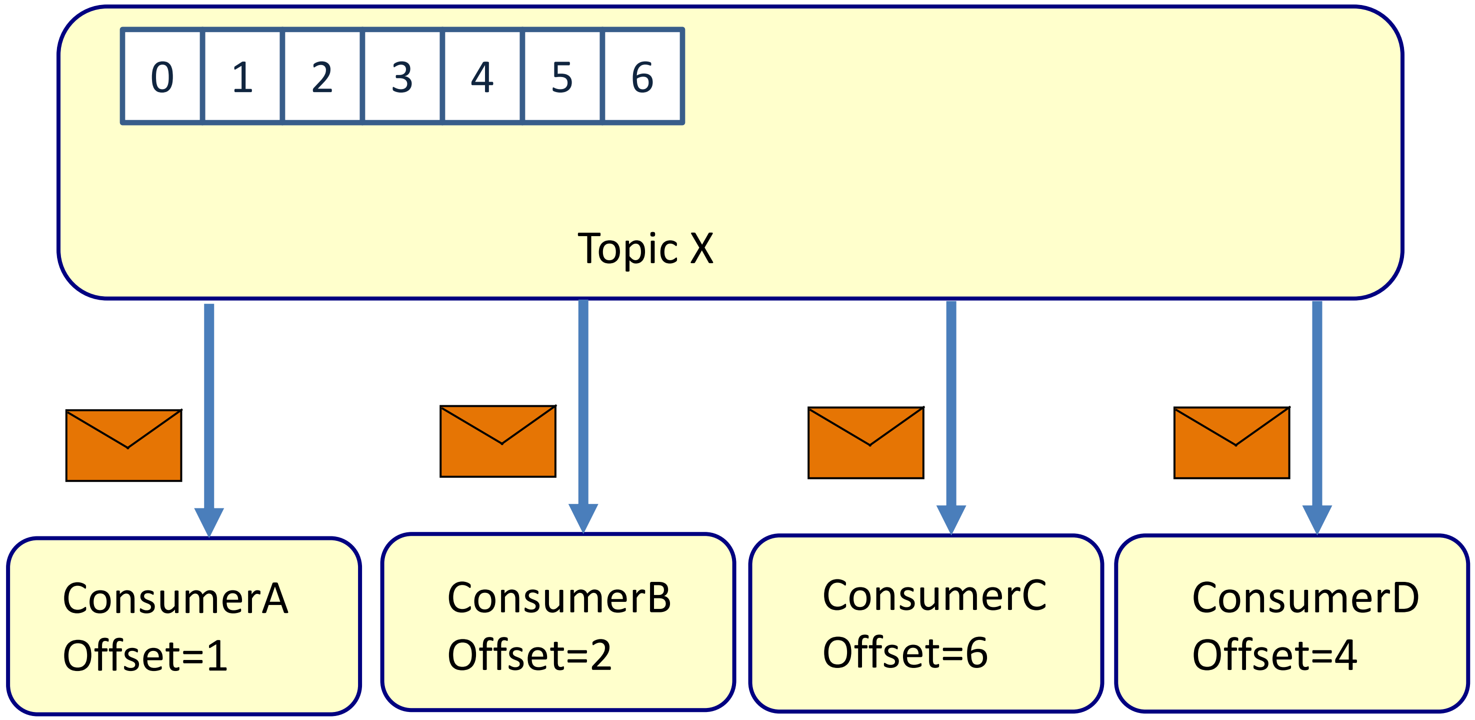
```
@Service
public class KafkaConsumer {

  @KafkaListener(topics = "topic1", groupId = "gid1")
  public void consume(Message<String> message) {
    MessageHeaders headers = message.getHeaders();
    System.out.println("Message header MyCustomHeader = "
                                   +headers.get("MyCustomHeader"));
    System.out.println("Message payload = "+message.getPayload());
  }
}
```

Receive a Message

```
Message header MyCustomHeader = HeaderValue
Message payload = Hello World
```

OFFSET

Offset



Every consumer has its own offset in the topic

For Kafka, it does not matter

1. How many consumers want to access the topic
2. If the consumer(s) have bugs
3. If the consumer(s) are down

Offset

```
{
   "Timestamp": "2025-08-25T09:34:11.608+00:00",
   "Topic": "topic1",
   "Partition": 0,
   "Offset": 13,
   "SchemaId": null,
   "SchemaType": null,
   "Key": null,
   "Headers": null,
   "Message": "Hello World"
  },
  {
   "Timestamp": "2025-08-25T09:35:21.386+00:00",
   "Topic": "topic1",
   "Partition": 0,
   "Offset": 14,
   "SchemaId": null,
   "SchemaType": null,
   "Key": null,
   "Headers": null,
   "Message": "Hello World"
  }
```

The offset header

The offset header

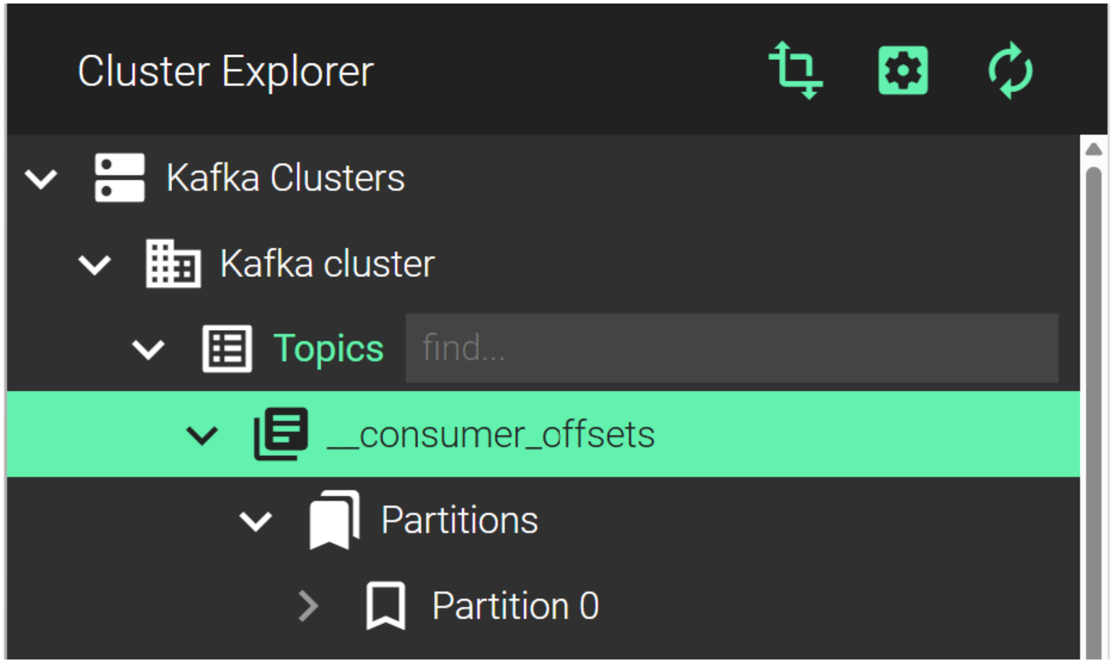
Offset

```
@Service
public class KafkaConsumer {

    @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(@Payload String message,
                               @Header(KafkaHeaders.OFFSET) long offset) {
        System.out.println("Message offset = "+offset);
        System.out.println("Message payload = "+message);
    }
}
```

Get the offset header

```
Message offset = 13
Message payload = Hello World
Message offset = 14
Message payload = Hello World
```

Offset is stored within Kafka

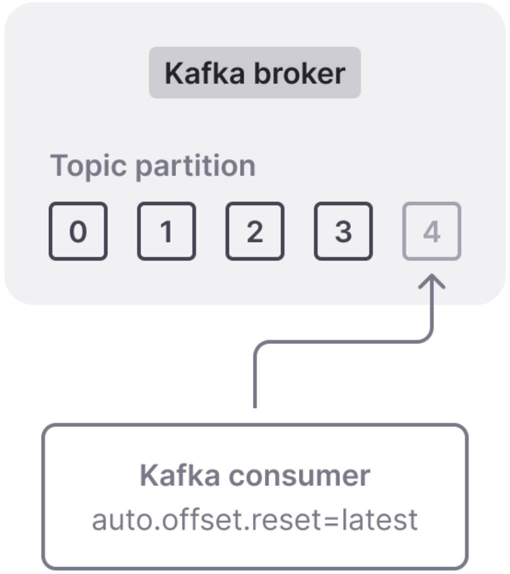


Offset: latest message

Get the latest message

```
@Service
public class KafkaConsumer {

  @KafkaListener(topics = "topic1", groupId = "gid1", properties = {"auto.offset.reset:latest"} )
  public void consume(String message, @Header(KafkaHeaders.OFFSET) long offset) {
    System.out.println("Message received = "+message+ ", offset= "+offset);
  }
}
```

```
Message received = Hello World, offset= 16
Message received = Hello World, offset= 17
```

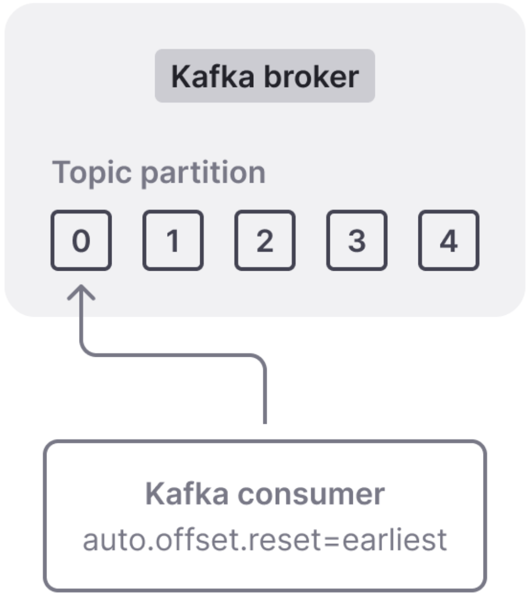


Offset: earliest message

Get the earliest message

```
@Service
public class KafkaConsumer {

  @KafkaListener(topics = "topic1", groupId = "gid2", properties = {"auto.offset.reset:earliest"}
  public void consume(String message, @Header(KafkaHeaders.OFFSET) long offset) {
    System.out.println("Message received = "+message+ ", offset= "+offset);
  }
}
```

```
Message received = Hello World, offset= 1
Message received = Hello World, offset= 2
```



Set auto.offset.reset in application.properties

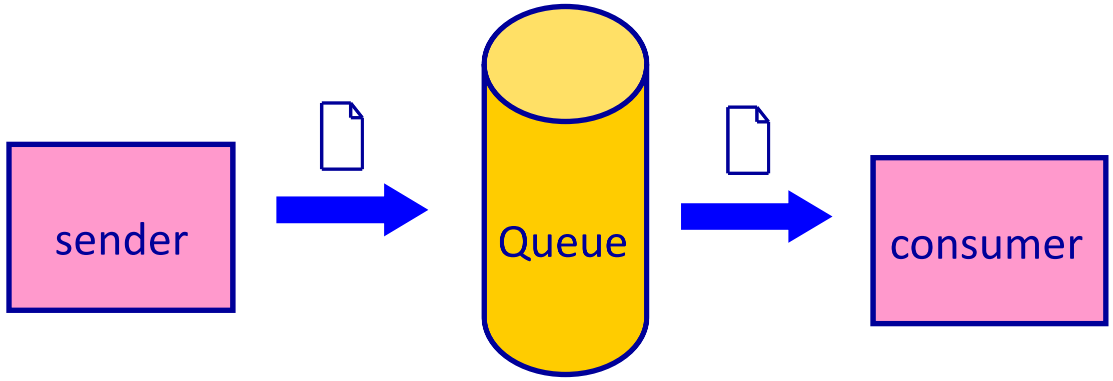
application.properties

```
spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.group-id= gid
spring.kafka.consumer.auto-offset-reset= earliest
```

GROUPID

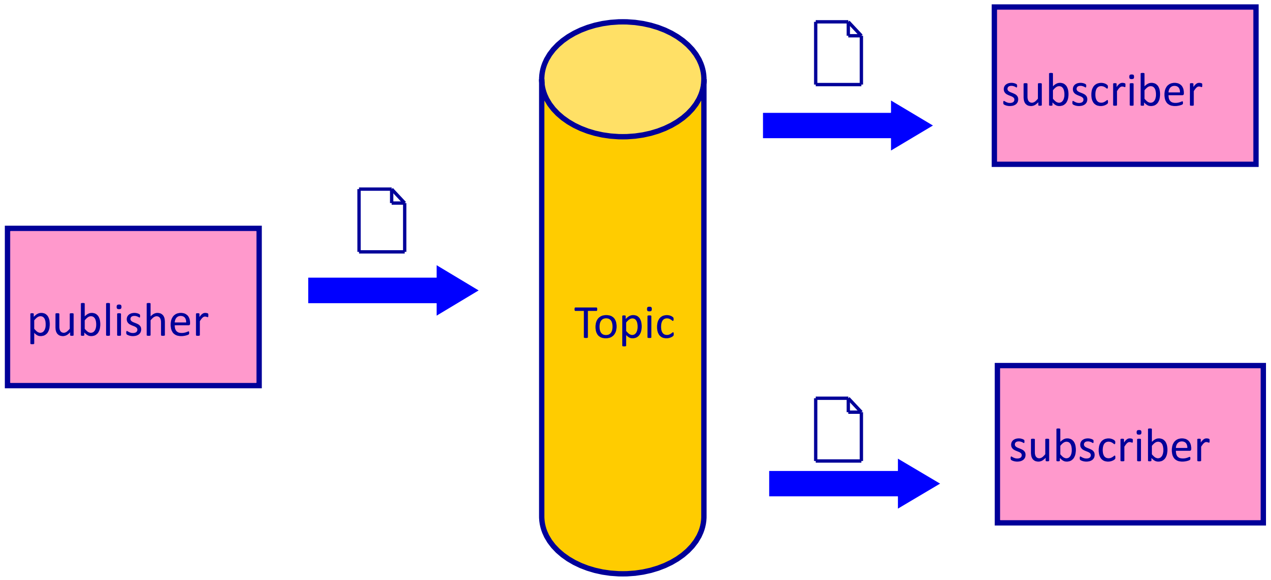
Point-To-Point (PTP)

- A dedicated consumer per Queue message



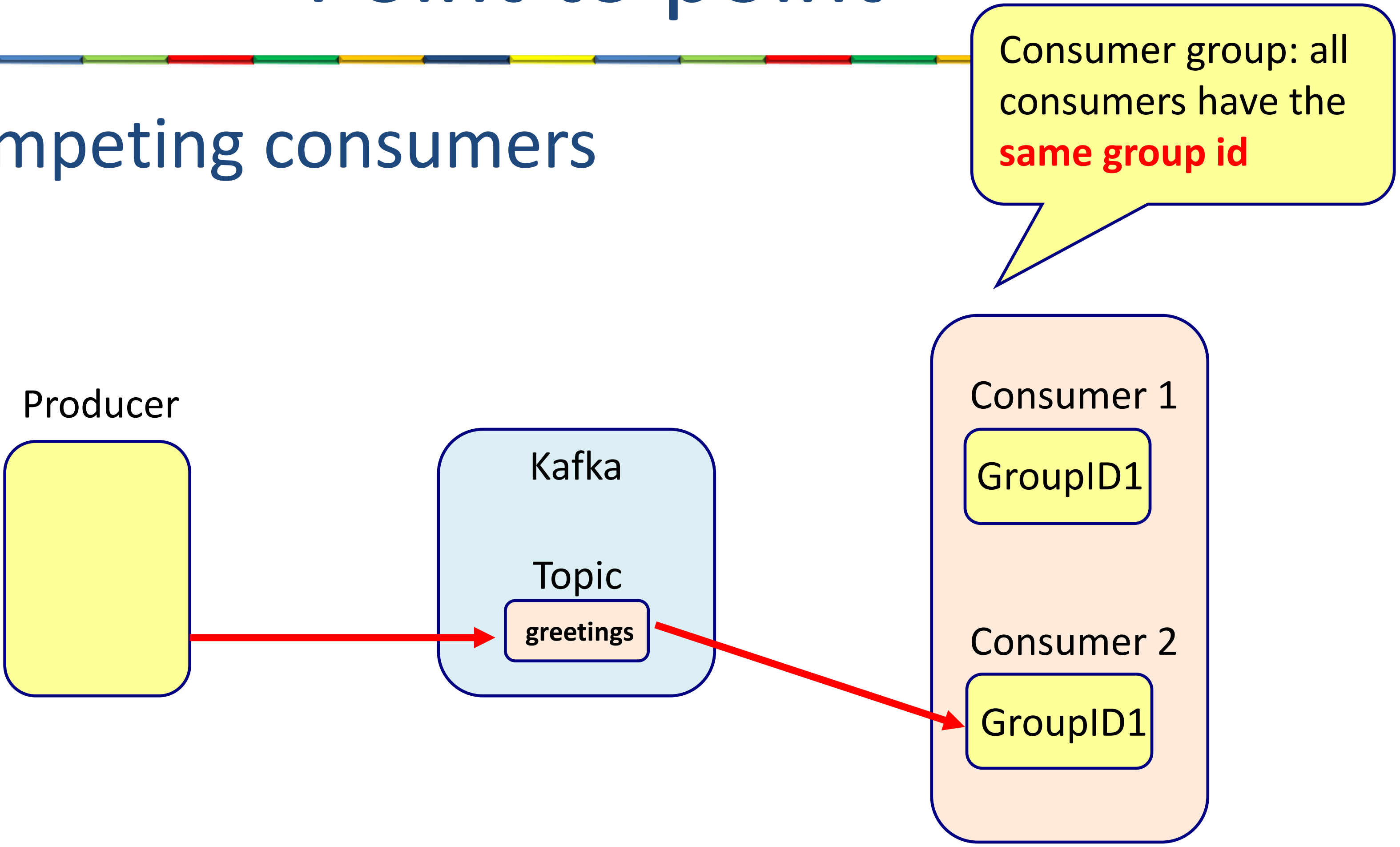
Publish-Subscribe (Pub-Sub)

- A message channel can have more than one ‘*consumer*’
  - Ideal for broadcasting



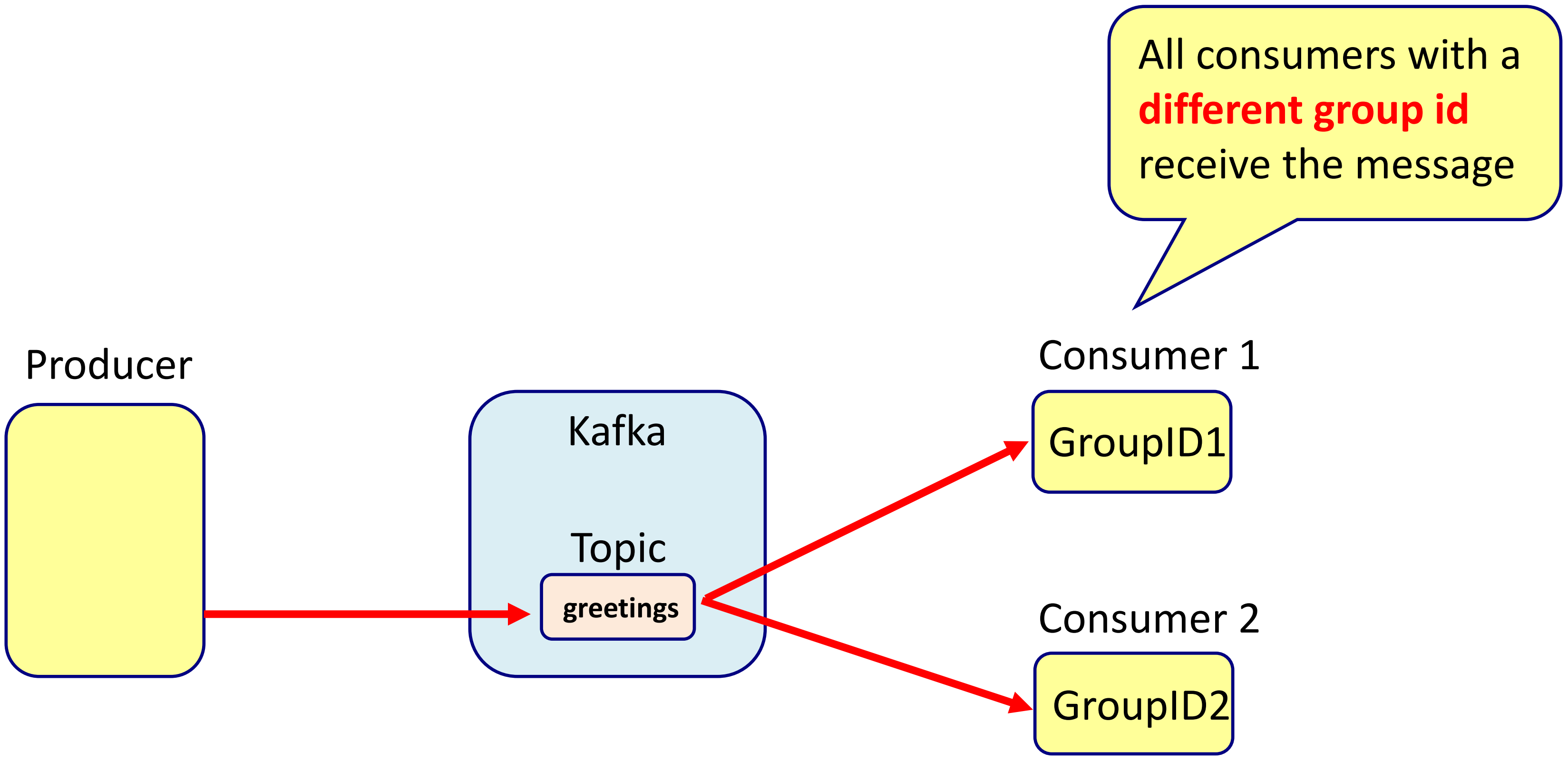
Point to point

- Competing consumers



- Only one consumers receives the message

Publish-Subscribe

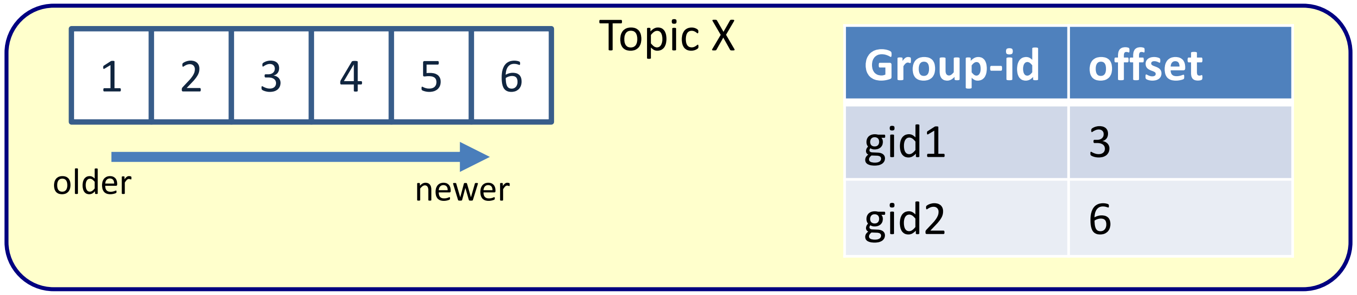

All consumers with a **different group id** receive the message
Producer
Kafka
Topic
greetings
Consumer 1
GroupID1
Consumer 2
GroupID2

Multiple listeners

```
@Service
public class KafkaConsumer {

  @KafkaListener(topics = "topic1", groupId = "gid1", properties = {"auto.offset.reset:earliest"}
  public void consume(String message, @Header(KafkaHeaders.OFFSET) long offset) {
    System.out.println("Message received 1= "+message+ ", offset= "+offset);
  }

  @KafkaListener(topics = "topic1", groupId = "gid1", properties = {"auto.offset.reset:earliest"}
  public void consume2(String message, @Header(KafkaHeaders.OFFSET) long offset) {
    System.out.println("Message received 2= "+message+ ", offset= "+offset);
  }

  @KafkaListener(topics = "topic1", groupId = "gid5", properties = {"auto.offset.reset:earliest"}
  public void consume3(String message, @Header(KafkaHeaders.OFFSET) long offset) {
    System.out.println("Message received 3= "+message+ ", offset= "+offset);
  }
}
```

Offset and group-id

- Offset is stored per group-id



Group-id	offset
gid1	3
gid2	6

A consumer connects to topic X in Kafka

Group-id	auto.offset.reset	Message received
gid1	earliest	4
gid1	latest	7
gid2	earliest	7
gid2	latest	7
gid3	earliest	1
gid3	latest	7

MESSAGES

Sending a message with KafkaTemplate

```
CompletableFuture<SendResult<K, V>> sendDefault(V data);

CompletableFuture<SendResult<K, V>> sendDefault(K key, V data);

CompletableFuture<SendResult<K, V>> sendDefault(Integer partition, K key, V data);

CompletableFuture<SendResult<K, V>> sendDefault(Integer partition, Long timestamp, K key, V data);

CompletableFuture<SendResult<K, V>> send(String topic, V data);

CompletableFuture<SendResult<K, V>> send(String topic, K key, V data);

CompletableFuture<SendResult<K, V>> send(String topic, Integer partition, K key, V data);

CompletableFuture<SendResult<K, V>> send(String topic, Integer partition, Long timestamp, K key, V data);

CompletableFuture<SendResult<K, V>> send(ProducerRecord<K, V> record);

CompletableFuture<SendResult<K, V>> send(Message<?> message);
```

Sending a message

```
public void sendMessage() {
   Product product = new Product("A158", "IPhone13", 180.0);
   kafkaTemplate.send("product_topic", product);
}
```

Sending an object

```
public void sendMessage() {
   ProducerRecord<String,String> producerRecord = new ProducerRecord<>("topic1","key",
                     "Hello World");
   producerRecord.headers().add("MyCustomHeader","HeaderValue".getBytes());
   kafkaTemplate.send(producerRecord);
}
```

Sending a ProducerRecord

```
public void sendMessage() {
   Product product = new Product("A158", "IPhone13", 180.0);
   Message<Product> message = MessageBuilder.withPayload(product)
         .setHeader("kafka_topic", "product_topic")  // Spring uses this header for the topic
         .build();
   kafkaTemplate.send(message);
}
```

Sending a Message

Receiving a message

```
@KafkaListener(topics = "product_topic", groupId = "gid1")
public void consume(Product product) {
   System.out.println(product);
}
```

Consume an object

```
@KafkaListener(topics = "topic1", groupId = "gid1")
public void consume(ConsumerRecord<String, String> consumerRecord) {
   System.out.println("Message received = "+consumerRecord.value()+ ", offset= "
          +consumerRecord.offset());
   Headers consumedHeaders = consumerRecord.headers();
   for (Header header : consumedHeaders) {
      System.out.println("Header key="+header.key()+" , value = "+new String(header.value()
   }
}
```

Consume a ConsumerRecord

Receiving a message

```
@KafkaListener(topics = "topic1", groupId = "gid1")
public void consume(Message<String> message) {
   System.out.println("Message received = "+message.getPayload());
   MessageHeaders consumedHeaders = message.getHeaders();
   System.out.println("offset="+consumedHeaders.get("kafka_offset"));
   System.out.println("group id="+consumedHeaders.get("kafka_groupId"));
}
```

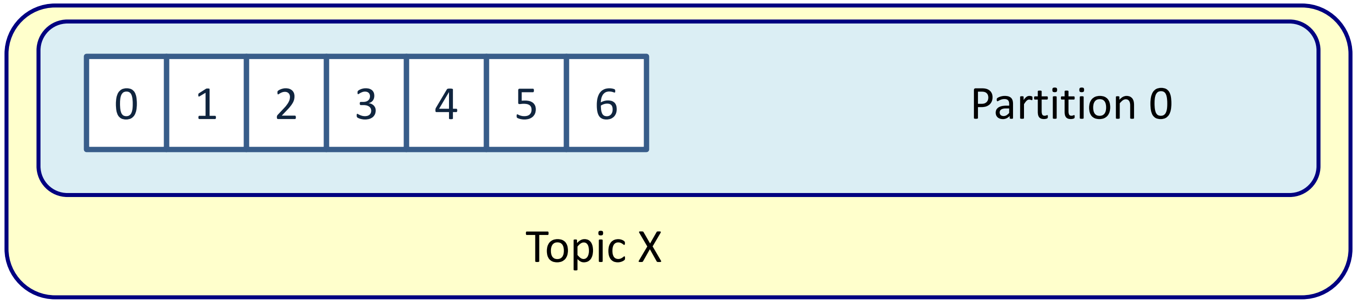
Consume a Message

PARTITIONS

Partition

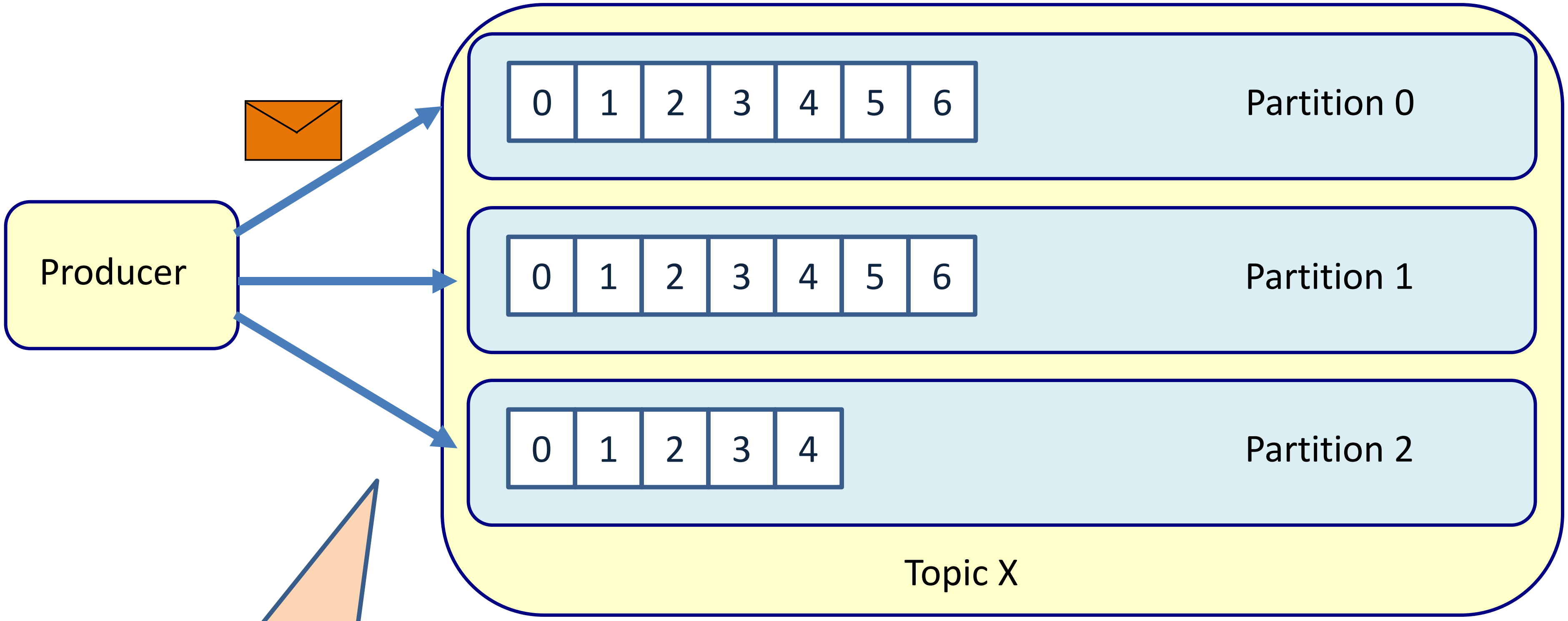
- Each topic has one or more partitions
  - This is configurable
- Each partition is maintained on 1 or more brokers

1 partition



- Each partition must fit on 1 broker

3 partitions

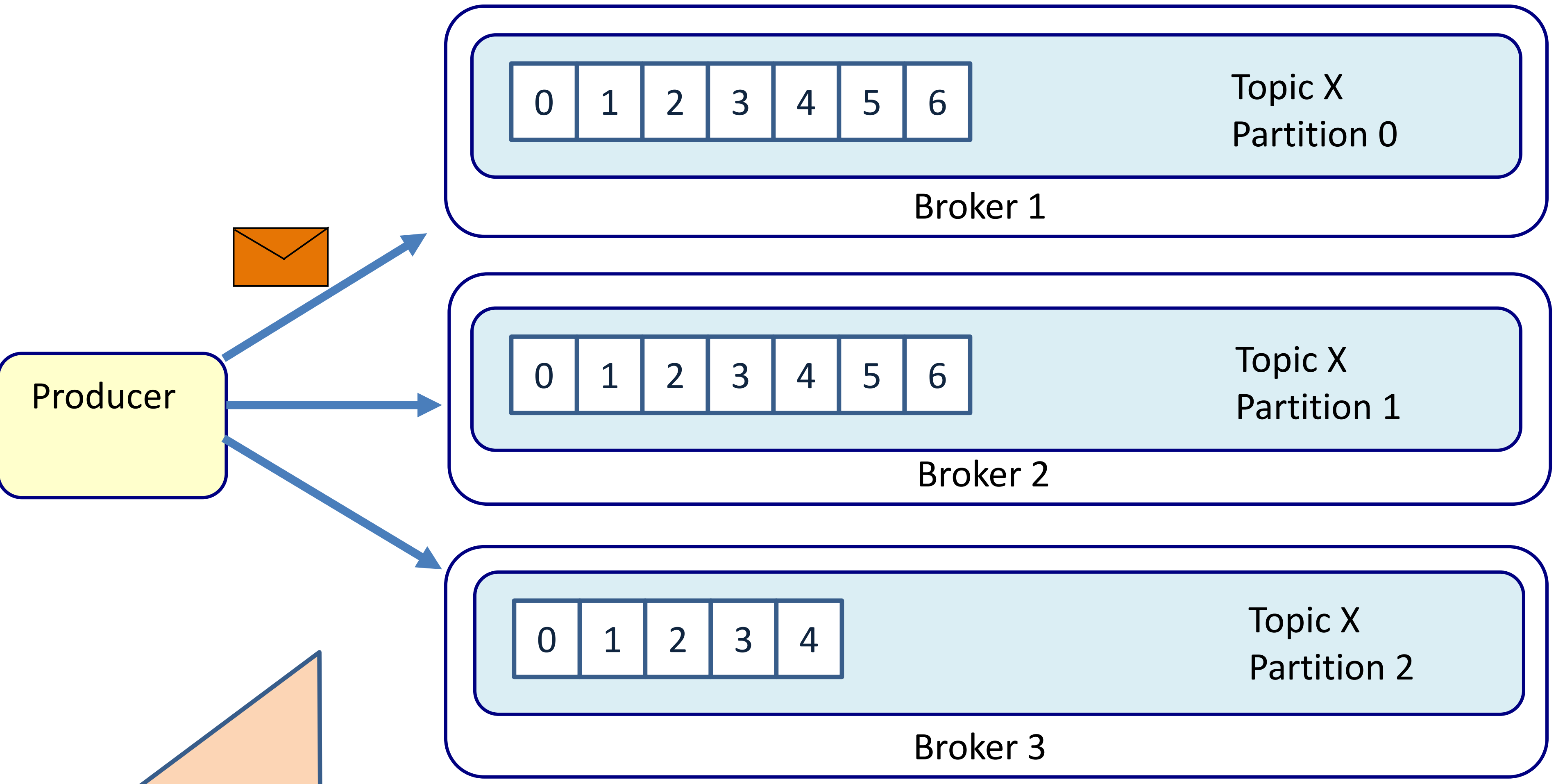


Each partition receives different messages

There is no message order between partitions
If this is important

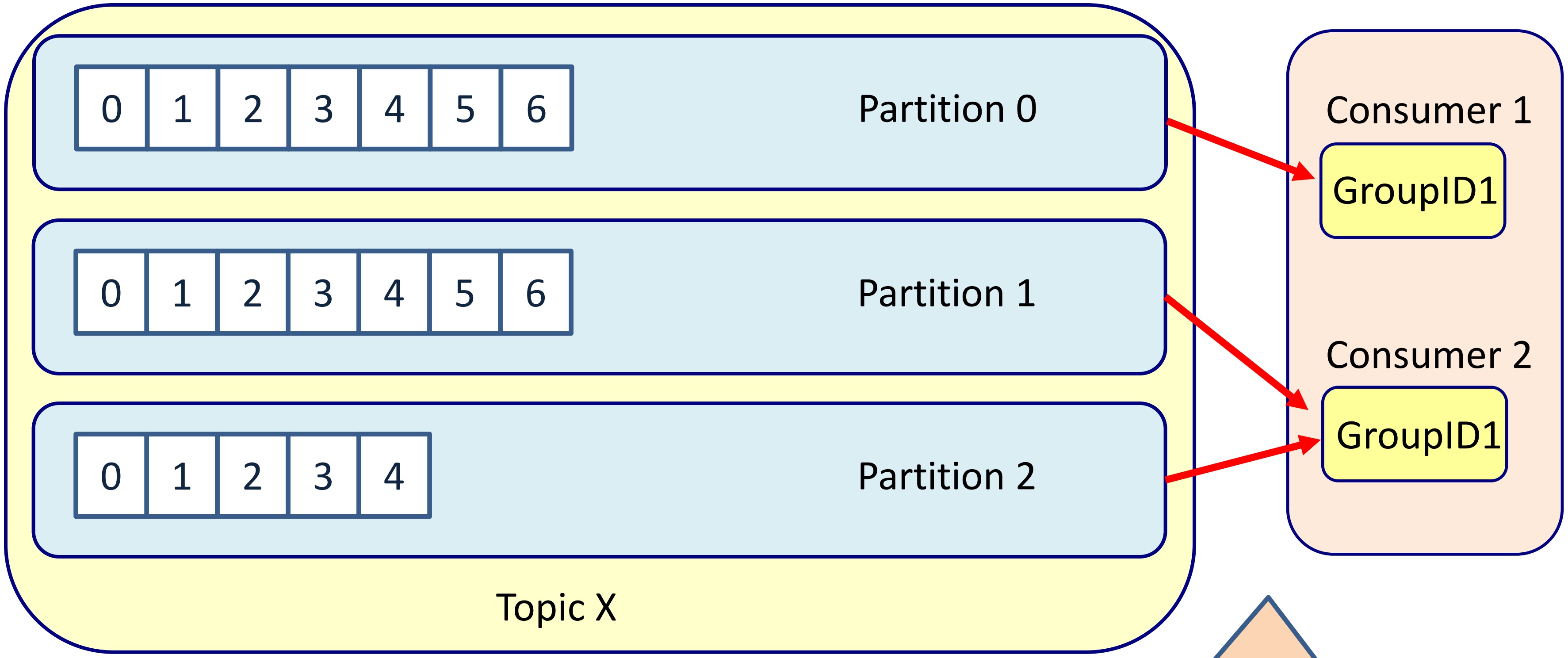
- Use a single partition
- Let the consumer manage the order

Scale out partitions



Increases
- Scalability (number of messages and their size)
- Higher levels of throughput (parallelism)

Consumer groups



Kafka automatically balances consumers in a group

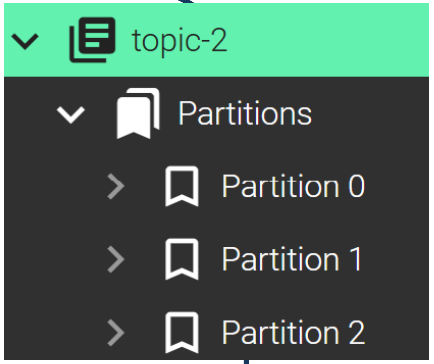
Create a topic with 3 partitions

```
@SpringBootApplication
public class KafkaProducerApplication implements CommandLineRunner {
   @Autowired
   KafkaProducer kafkaProducer;

   public static void main(String[] args) {
     SpringApplication.run(KafkaProducerApplication.class, args);
   }

   @Override
   public void run(String... args) throws Exception {
     kafkaProducer.sendMessage();
   }
   @Bean
   public NewTopic createtopic2() {
     return TopicBuilder.name("topic-2").partitions(3).build();
   }
}
```

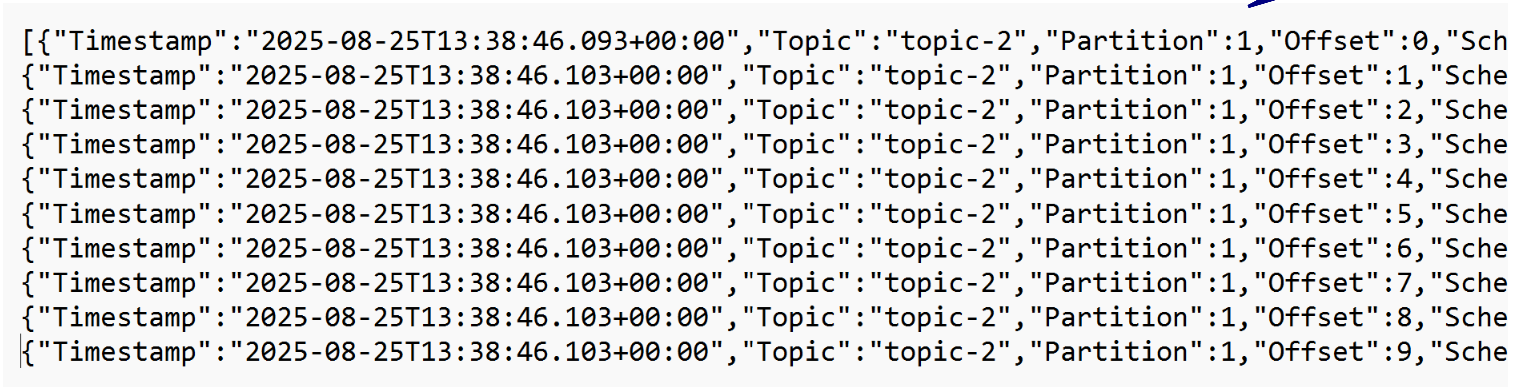
Create a topic with 3 partitions

Producer

```
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage() {
        for (int x=1; x<11 ; x++){
            kafkaTemplate.send("topic-2", "Message-"+x);
        }
    }
}
```



All messages go to 1 partition



```
[{"Timestamp":"2025-08-25T13:38:46.093+00:00","Topic":"topic-2","Partition":1,"Offset":0,"Sch
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":1,"Sche
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":2,"Sche
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":3,"Sche
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":4,"Sche
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":5,"Sche
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":6,"Sche
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":7,"Sche
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":8,"Sche
{"Timestamp":"2025-08-25T13:38:46.103+00:00","Topic":"topic-2","Partition":1,"Offset":9,"Sche
```

Consumer

```
@Service
public class KafkaConsumer {

   @KafkaListener(topics = "topic-2", groupId = "gid1", properties =
                                                  {"auto.offset.reset:earliest"} )
   public void consume(String message,
       @Header(KafkaHeaders.OFFSET) long offset,
       @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
     System.out.println("Message received 1= "+message+ ", offset= "+offset+ ", partition= "
       +partition);
   }
}
```

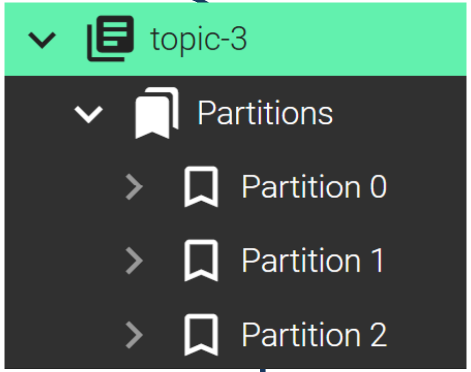
Consumer is connected to all 3 partitions

```
Message received 1= Message-1, offset= 12, partition= 1
Message received 1= Message-2, offset= 13, partition= 1
Message received 1= Message-3, offset= 14, partition= 1
Message received 1= Message-4, offset= 15, partition= 1
Message received 1= Message-5, offset= 16, partition= 1
Message received 1= Message-6, offset= 17, partition= 1
Message received 1= Message-7, offset= 18, partition= 1
Message received 1= Message-8, offset= 19, partition= 1
Message received 1= Message-9, offset= 20, partition= 1
Message received 1= Message-10, offset= 21, partition= 1
```

All messages come from the same partition

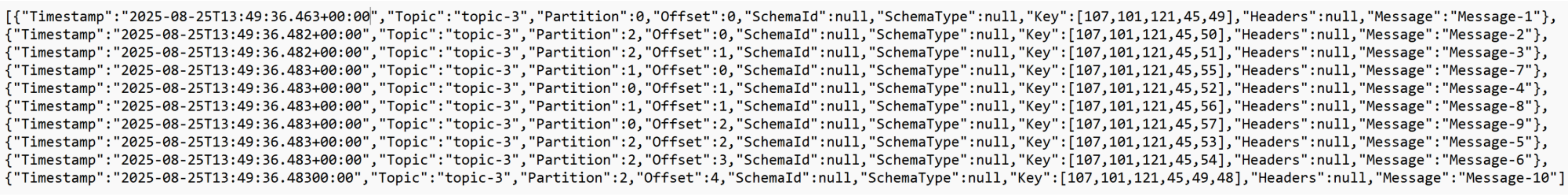
Producer

```java
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage() {
        for (int x=1; x<11 ; x++){
            kafkaTemplate.send("topic-3","key-"+x ,"Message-"+x);
        }
    }
}
```



Every message has an unique key

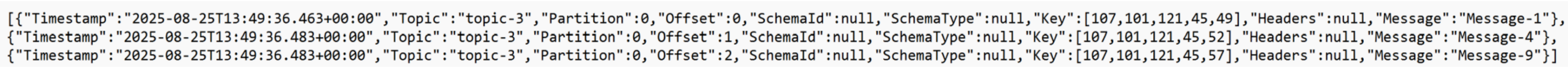
Messages are distributed over the 3 partitions



```
[{"Timestamp":"2025-08-25T13:49:36.463+00:00","Topic":"topic-3","Partition":0,"Offset":0,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,49],"Headers":null,"Message":"Message-1"},
{"Timestamp":"2025-08-25T13:49:36.482+00:00","Topic":"topic-3","Partition":2,"Offset":0,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,50],"Headers":null,"Message":"Message-2"},
{"Timestamp":"2025-08-25T13:49:36.482+00:00","Topic":"topic-3","Partition":2,"Offset":1,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,51],"Headers":null,"Message":"Message-3"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":1,"Offset":0,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,55],"Headers":null,"Message":"Message-7"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":0,"Offset":1,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,52],"Headers":null,"Message":"Message-4"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":1,"Offset":1,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,56],"Headers":null,"Message":"Message-8"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":0,"Offset":2,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,57],"Headers":null,"Message":"Message-9"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":2,"Offset":2,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,53],"Headers":null,"Message":"Message-5"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":2,"Offset":3,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,54],"Headers":null,"Message":"Message-6"},
{"Timestamp":"2025-08-25T13:49:36.48300:00","Topic":"topic-3","Partition":2,"Offset":4,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,49,48],"Headers":null,"Message":"Message-10"]
```

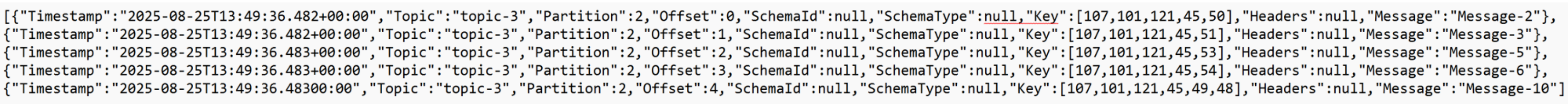
All messages have an unique key

- Partition 0



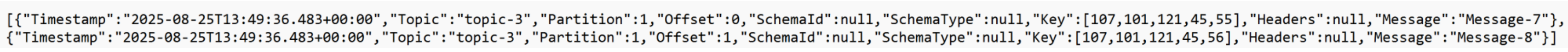
```
[{"Timestamp":"2025-08-25T13:49:36.463+00:00","Topic":"topic-3","Partition":0,"Offset":0,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,49],"Headers":null,"Message":"Message-1"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":0,"Offset":1,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,52],"Headers":null,"Message":"Message-4"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":0,"Offset":2,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,57],"Headers":null,"Message":"Message-9"}]
```

- Partition 1



```
[{"Timestamp":"2025-08-25T13:49:36.482+00:00","Topic":"topic-3","Partition":2,"Offset":0,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,50],"Headers":null,"Message":"Message-2"},
{"Timestamp":"2025-08-25T13:49:36.482+00:00","Topic":"topic-3","Partition":2,"Offset":1,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,51],"Headers":null,"Message":"Message-3"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":2,"Offset":2,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,53],"Headers":null,"Message":"Message-5"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":2,"Offset":3,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,54],"Headers":null,"Message":"Message-6"},
{"Timestamp":"2025-08-25T13:49:36.48300:00","Topic":"topic-3","Partition":2,"Offset":4,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,49,48],"Headers":null,"Message":"Message-10"]
```

- Partition 2



```
[{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":1,"Offset":0,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,55],"Headers":null,"Message":"Message-7"},
{"Timestamp":"2025-08-25T13:49:36.483+00:00","Topic":"topic-3","Partition":1,"Offset":1,"SchemaId":null,"SchemaType":null,"Key":[107,101,121,45,56],"Headers":null,"Message":"Message-8"}]
```

Consumer

Consumer is connected to all 3 partitions

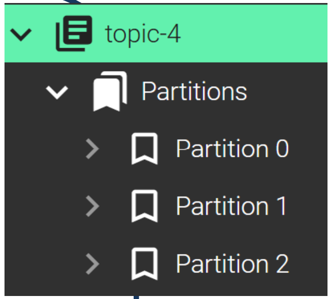
```
@Service
public class KafkaConsumer {

   @KafkaListener(topics = "topic-3", groupId = "gid1", properties =
                                                      {"auto.offset.reset:latest"} )
   public void consume(String message,
            @Header(KafkaHeaders.OFFSET) long offset,
            @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
     System.out.println("Message received 1= "+message+ ", offset= "+offset+ ", partition= "
            +partition);
   }
}
```

```
Message received 1= Message-1, offset= 5, partition= 2
Message received 1= Message-2, offset= 6, partition= 2
Message received 1= Message-9, offset= 7, partition= 2
Message received 1= Message-3, offset= 2, partition= 1
Message received 1= Message-5, offset= 3, partition= 1
Message received 1= Message-8, offset= 4, partition= 1
Message received 1= Message-4, offset= 3, partition= 0
Message received 1= Message-6, offset= 4, partition= 0
Message received 1= Message-7, offset= 5, partition= 0
Message received 1= Message-10, offset= 6, partition= 0
```

All messages come from different partition

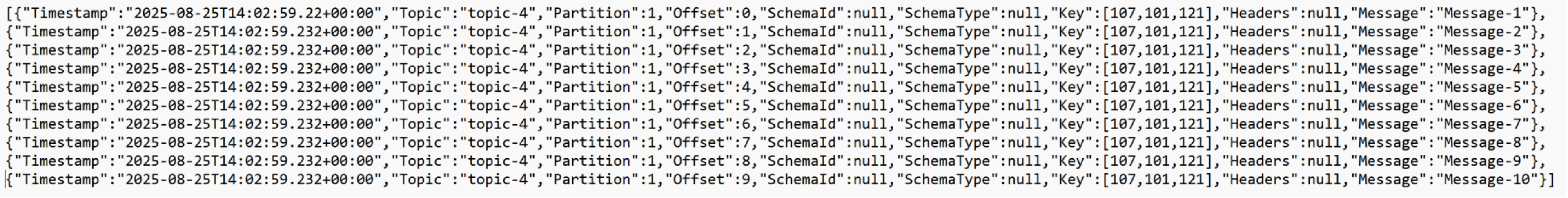
Producer

```
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage() {
        for (int x=1; x<11 ; x++){
            kafkaTemplate.send("topic-4","key" ,"Message-"+x);
        }
    }
}
```



All messages have the same key

All messages go to 1 partition

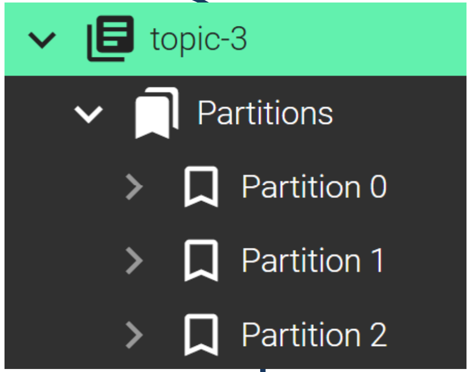


```
[{"Timestamp":"2025-08-25T14:02:59.22+00:00","Topic":"topic-4","Partition":1,"Offset":0,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-1"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":1,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-2"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":2,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-3"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":3,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-4"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":4,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-5"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":5,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-6"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":6,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-7"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":7,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-8"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":8,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-9"},
{"Timestamp":"2025-08-25T14:02:59.232+00:00","Topic":"topic-4","Partition":1,"Offset":9,"SchemaId":null,"SchemaType":null,"Key":[107,101,121],"Headers":null,"Message":"Message-10"}]
```

Producer

```
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage() {
        for (int x=1; x<11 ; x++){
            kafkaTemplate.send("topic-3","key-"+x ,"Message-"+x);
        }
    }
}
```

Every message has an unique key



Connect listener to different partitions

```
@Service
public class KafkaConsumer {

   @KafkaListener(groupId = "gid1", properties = {"auto.offset.reset:latest"}, topicPartitions =
         { @TopicPartition(topic = "topic-3", partitions = { "0" }) })
   public void consume(String message,
            @Header(KafkaHeaders.OFFSET) long offset,
            @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
      System.out.println("Listener 1: Message received = "+message+ ", offset= "+offset+ ",
partition= "+partition);
   }

   @KafkaListener(groupId = "gid1", properties = {"auto.offset.reset:latest"}, topicPartitions =
         { @TopicPartition(topic = "topic-3", partitions = { "1" }) })
   public void consume2(String message,
            @Header(KafkaHeaders.OFFSET) long offset,
            @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
      System.out.println("Listener 2: Message received = "+message+ ", offset= "+offset+ ",
partition= "+partition);
   }
```

Connect listener to different partitions

```
  @KafkaListener(groupId = "gid1", properties = {"auto.offset.reset:latest"}, topicPartitions =
          { @TopicPartition(topic = "topic-3", partitions = { "2" }) })
   public void consume3(String message,
                @Header(KafkaHeaders.OFFSET) long offset,
                @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
       System.out.println("Listener 3: Message received = "+message+ ", offset= "+offset+ ",
partition= "+partition);
   }
}
```

Consumer

```
Listener 1: Message received = Message-4, offset= 15, partition= 0
Listener 3: Message received = Message-1, offset= 14, partition= 2
Listener 2: Message received = Message-3, offset= 11, partition= 1
Listener 2: Message received = Message-5, offset= 12, partition= 1
Listener 1: Message received = Message-6, offset= 16, partition= 0
Listener 2: Message received = Message-8, offset= 13, partition= 1
Listener 3: Message received = Message-2, offset= 15, partition= 2
Listener 1: Message received = Message-7, offset= 17, partition= 0
Listener 3: Message received = Message-9, offset= 16, partition= 2
Listener 1: Message received = Message-10, offset= 18, partition= 0
```

Different listener for every partition

concurrency

```
@Service
public class KafkaConsumer {

    @KafkaListener(topics = "topic-3", groupId = "gid1", properties =
{"auto.offset.reset:latest"}, concurrency = "3")
    public void consume(String message,
                @Header(KafkaHeaders.OFFSET) long offset,
                @Header(KafkaHeaders.RECEIVED_PARTITION) int partition) {
        System.out.println(Thread.currentThread().getName()+"Message received =
"+message+ ", offset= "+offset+ ", partition= "+partition);
    }

}
```

concurrency = 3. Create 3 listeners, one for every partition

concurrency

```
org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1Message received = Message-4,
offset= 19, partition= 0
org.springframework.kafka.KafkaListenerEndpointContainer#0-2-C-1Message received = Message-1,
offset= 17, partition= 2
org.springframework.kafka.KafkaListenerEndpointContainer#0-1-C-1Message received = Message-3,
offset= 14, partition= 1
org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1Message received = Message-6,
offset= 20, partition= 0
org.springframework.kafka.KafkaListenerEndpointContainer#0-1-C-1Message received = Message-5,
offset= 15, partition= 1
org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1Message received = Message-7,
offset= 21, partition= 0
org.springframework.kafka.KafkaListenerEndpointContainer#0-2-C-1Message received = Message-2,
offset= 18, partition= 2
org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1Message received = Message-10,
offset= 22, partition= 0
org.springframework.kafka.KafkaListenerEndpointContainer#0-1-C-1Message received = Message-8,
offset= 16, partition= 1
org.springframework.kafka.KafkaListenerEndpointContainer#0-2-C-1Message received = Message-9,
offset= 19, partition= 2
```

3 different listeners, one for every partition

ERROR HANDLING

Producer errors

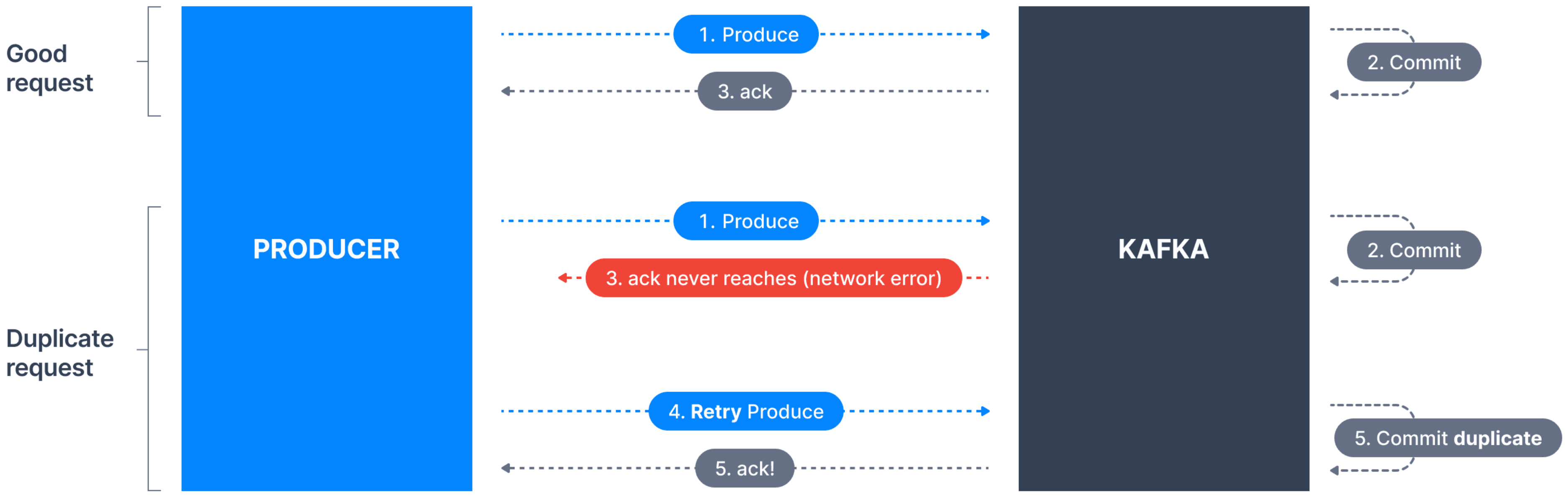
```
@Service
public class KafkaProducer {
  @Autowired
  private KafkaTemplate<String, String> kafkaTemplate;

  public void sendMessage() {

    CompletableFuture<SendResult<String, String>> future2 = kafkaTemplate.send("topic1",
                 "Hello");
    future2.whenComplete((result, sendExeption) -> {
      if (sendExeption == null) {
        System.out.println("Sent Message= Hello to topic= topic1 with offset= " +
                 result.getRecordMetadata().offset());
      } else {
        System.out.println("Unable to send Message= Hello World due to " +
                 sendExeption.getMessage());
      }
    });
  }
}
```

Check CompletableFuture if there was an exception

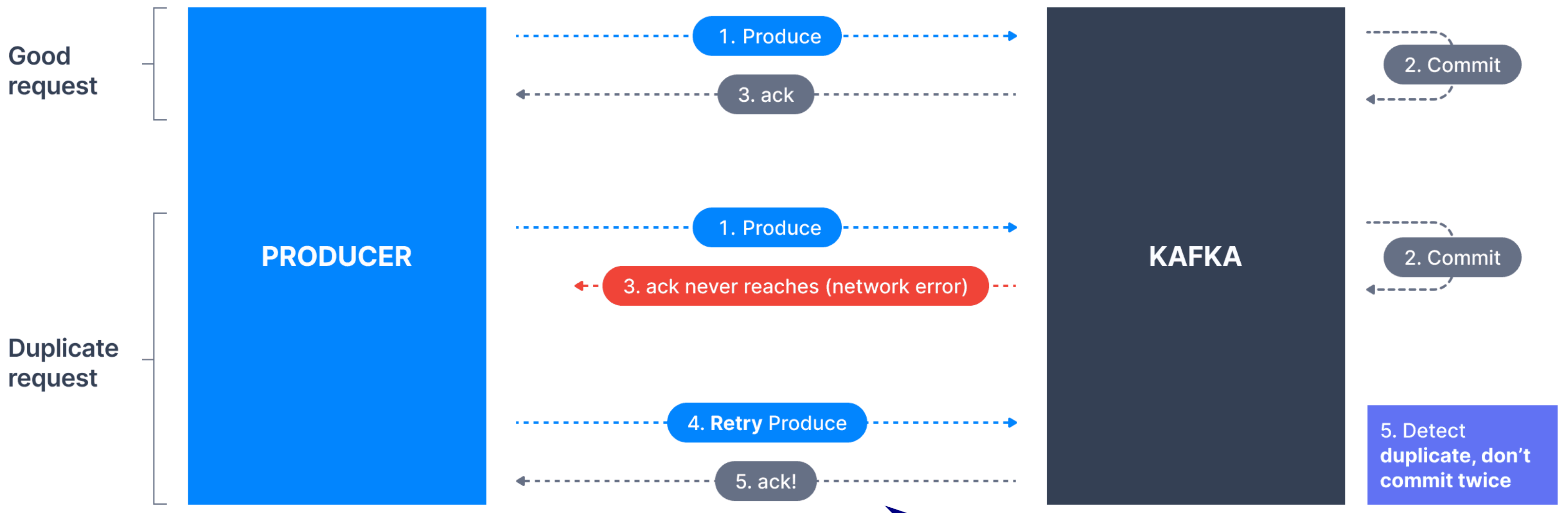
Producer retry

- If producer cannot send the message to Kafka, it retries multiple times and ultimately times out after 2 minutes.
- Default values
  - *retries* (defaults to *Integer.MAX_VALUE*): the maximum number of attempts to publish the message
  - *delivery.timeout.ms* (defaults to 120.000): the maximum time to wait for a message to be acknowledged before considering it failed
  - *retry.backoff.ms* (defaults to 100): the time to wait before retrying

Problem with producer retries

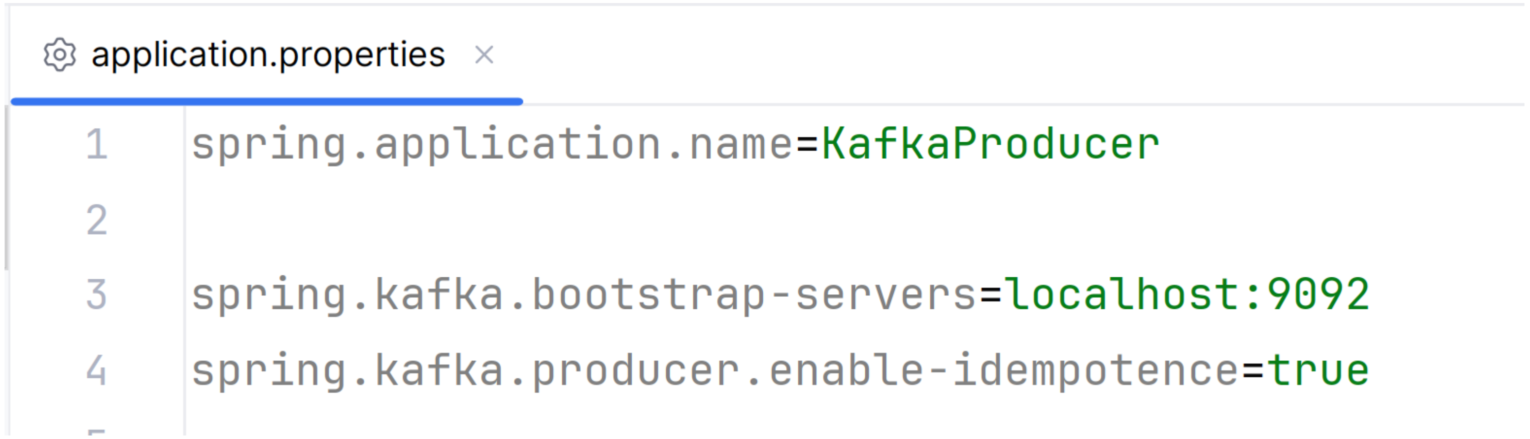


Idempotent producer



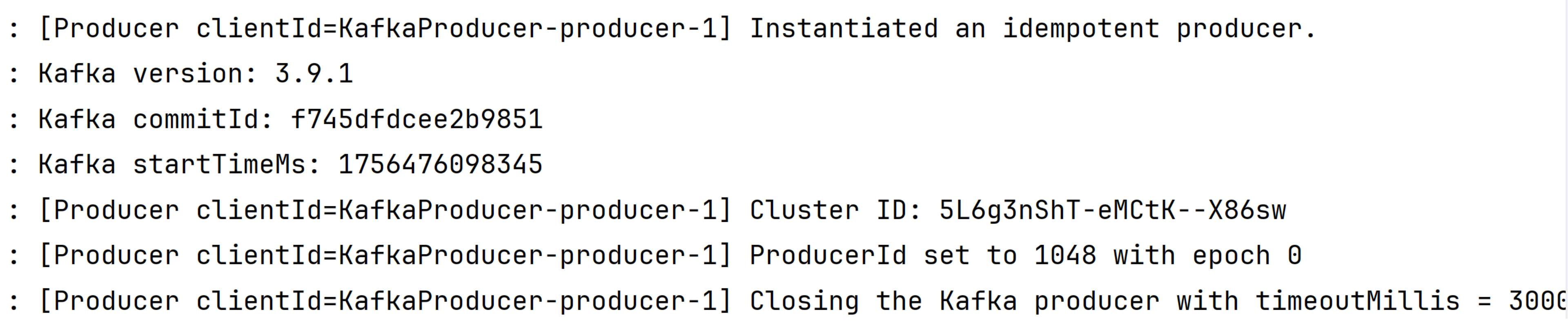
Every producer gets a Producer Id (PID)

Idempotent producer



application.properties

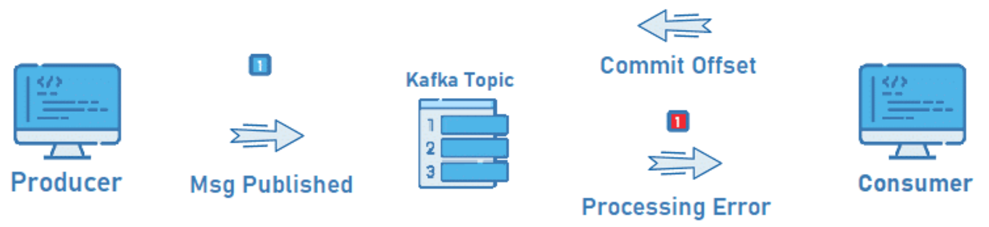
```
spring.application.name=KafkaProducer

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.producer.enable-idempotence=true
```



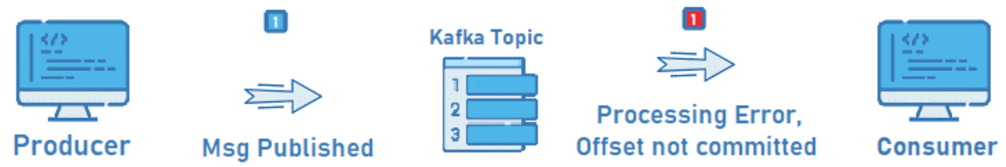
```
: [Producer clientId=KafkaProducer-producer-1] Instantiated an idempotent producer.
: Kafka version: 3.9.1
: Kafka commitId: f745dfdcee2b9851
: Kafka startTimeMs: 1756476098345
: [Producer clientId=KafkaProducer-producer-1] Cluster ID: 5L6g3nShT-eMCtK--X86sw
: [Producer clientId=KafkaProducer-producer-1] ProducerId set to 1048 with epoch 0
: [Producer clientId=KafkaProducer-producer-1] Closing the Kafka producer with timeoutMillis = 300
```

Consumer

- At most once



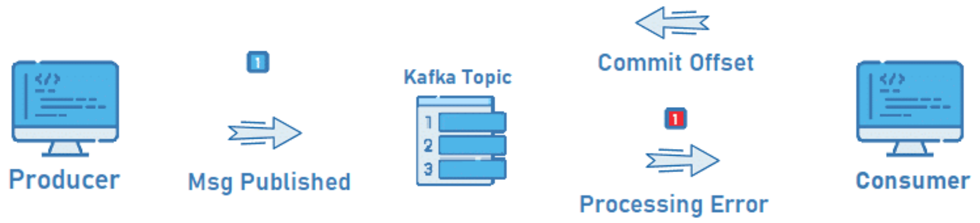
- At least once



Consumer needs to be idempotent

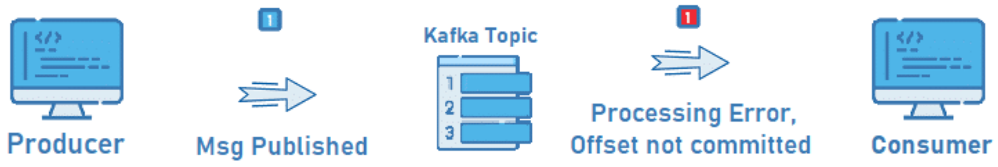
- Exactly once
  - You need a cache or database to store the messages you have already processed

At most once



- This is the default for Spring Boot Kafka
- The message is automatically acknowledged the moment it is received by the consumer
- Problem: what if we get an error during processing the message

At least once



- Retry when there is a processing error
- What if it still fails after a few retries?
  - Send the message to the Dead Letter Topic (DLT)

Simple producer

```
@Service
public class KafkaProducer2 {
  @Autowired
  private KafkaTemplate<String, String> kafkaTemplate;

  public void sendMessage() {
    kafkaTemplate.send("topic1", "Hello World");
    kafkaTemplate.send("topic1", "Hello");
    kafkaTemplate.send("topic1", "GoodBy");
  }
}
```

Handle error in consumer

```
@Service
public class KafkaConsumer2 {
  @KafkaListener(topics = "topic1", groupId = "gid1")
  public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
          topic, @Header(KafkaHeaders.OFFSET) long offset) {
    try {
      System.out.println("Listener: Message received = " + message + ", topic= " + topic + ",
         offset= " + offset);
      if (message.equals("Hello")) {
        throw new RuntimeException("Invalid message received");
      }
    } catch (Exception exception) {
      System.out.println("Exception in listener: exception = " + exception);
    }
  }
}
```

Exception in message handler

```
Listener: Message received = Hello World, topic= topic1, offset= 15
Listener: Message received = Hello, topic= topic1, offset= 16
Exception in listener: exception = java.lang.RuntimeException: Invalid message received
Listener: Message received = GoodBy, topic= topic1, offset= 17
```

Consumer error handling options

- **Blocking** retry
  - Do retry when retriable exceptions occur during consuming a message, and **block** the next message.
  - **DefaultErrorHandler**

Consumer thread is blocked during retry
Order is guaranteed

- **Non-blocking** retry
  - Send the message to another retry topic, when the message exceeds the retry max attempts limit.
  - **@RetryableTopic**

Consumer thread is not blocked during retry
Order is not guaranteed

- Dead letter queue and handler
  - Send the message to a dead letter topic.

Global exception handler (DefaultErrorHandler)

```
@Configuration
public class KafkaConfig {
    @Bean
    public DefaultErrorHandler errorHandler() {
        FixedBackOff noRetry = new FixedBackOff(0L, 0L);

        BiConsumer<ConsumerRecord<?, ?>, Exception> globalHandler = (record, ex) -> {
            System.err.println("Kafka error occurred:");
            System.err.println("Topic: " + record.topic());
            System.err.println("Value: " + record.value());
            System.err.println("Exception: " + ex.getMessage());
        };
        DefaultErrorHandler errorHandler = new DefaultErrorHandler(
                (record, exception) -> globalHandler.accept(record, exception),
                noRetry
        );
        return errorHandler;
    }
}
```

Global exception handler config

```
@Bean
public ConcurrentKafkaListenerContainerFactory<String, String>
            kafkaListenerContainerFactory(
      ConsumerFactory<String, String> consumerFactory,
      DefaultErrorHandler errorHandler) {

   ConcurrentKafkaListenerContainerFactory<String, String> factory =
         new ConcurrentKafkaListenerContainerFactory<>();
   factory.setConsumerFactory(consumerFactory);
   factory.setCommonErrorHandler(errorHandler);
   return factory;
}
```

Add global error handler to the ListernerContainer

Kafka listener with global exception handler

```
public class KafkaConsumer2 {
   @KafkaListener(topics = "topic1", groupId = "gid1")
   public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
            topic, @Header(KafkaHeaders.OFFSET) long offset) {

      System.out.println("Listener: Message received = " + message + ", topic= " + topic + ",
            offset= " + offset);
      if (message.equals("Hello")) {
         throw new RuntimeException("Invalid message received");
      }
   }
}
```

No exception handling in the Listener

Kafka listener with global exception handler

```
Listener: Message received = Hello World, topic= topic1, offset= 21
Listener: Message received = Hello, topic= topic1, offset= 22
Kafka error occurred:
Topic: topic1
Value: Hello
Exception: Listener method 'public void
consumer.KafkaConsumer2.consume(java.lang.String,java.lang.String,long)' threw exception
Listener: Message received = GoodBy, topic= topic1, offset= 23
```

Global error handler with retries

```
@Configuration
public class KafkaConfig2 {
  @Bean
  public DefaultErrorHandler errorHandler() {
    // Retry 2 times, wait 1 second between retries
    FixedBackOff fixedBackOff = new FixedBackOff(1000L, 2L);

    DefaultErrorHandler errorHandler = new DefaultErrorHandler(
        (ConsumerRecord<?, ?> record, Exception ex) -> {
          // Custom recovery logic after retries are exhausted
          System.err.println("Error after retries for record: " + record.value()
              + ", exception: " + ex.getMessage());
        },
        fixedBackOff
    );
    return errorHandler;
  }
```

Blocking retry

2 retries

Global error handler with retries and Dead Letter Topic

```
@Bean
public DefaultErrorHandler errorHandler(KafkaTemplate<Object, Object> kafkaTemplate) {
   // Sends failed messages to "<topic>.DLT"
   DeadLetterPublishingRecoverer recoverer = new
       DeadLetterPublishingRecoverer(kafkaTemplate,
         (record, exception) -> {
            return new TopicPartition(record.topic() + ".DLT",  record.partition());
         });

   // Retry twice with 1 second interval, then send to DLT
   FixedBackOff backOff = new FixedBackOff(1000L, 2L);

   DefaultErrorHandler errorHandler = new DefaultErrorHandler(recoverer, backOff);
   return errorHandler;
}
```

Send failed messages to DLT

2 retries

Global error handler with retries and Dead Letter Topic

```
@Service
public class KafkaConsumer2 {
  @KafkaListener(topics = "topic1", groupId = "gid1")
  public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
        topic, @Header(KafkaHeaders.OFFSET) long offset) {

    System.out.println("Listener: Message received = " + message + ", topic= " + topic + ",
    offset= " + offset);
    if (message.equals("Hello")) {
      throw new RuntimeException("Invalid message received");
    }
  }

  @KafkaListener(topics = "topic1.DLT", groupId = "dlt-group")
  public void handleDltMessage(String message) {
    System.err.println("Received from DLT: " + message);
  }
}
```

Listen on DLT

Global error handler with retries

```
Listener: Message received = Hello World, topic= topic1, offset= 33
Listener: Message received = Hello, topic= topic1, offset= 34
…
Listener: Message received = Hello, topic= topic1, offset= 34
…
Listener: Message received = Hello, topic= topic1, offset= 34
…
Received from DLT: Hello
…
Listener: Message received = GoodBy, topic= topic1, offset= 35
```

Retry 1

Retry 2

Send to DLT

Received from DLT

Consumer error handling options

- Blocking retry
  - Do retry when retriable exceptions occur during consuming a message, and **block** the next message.
  - **DefaultErrorHandler**
- Non-blocking retry
  - Send the message to another retry topic, when the message exceeds the blocking retry max attempts limit.
  - **@RetryableTopic**
- Dead letter queue and handler
  - Send the message to another dead letter topic.

Non blocking retries with DLT

```
@Service
public class KafkaConsumer {

   @RetryableTopic(attempts = "2")
   @KafkaListener(topics = "topic1", groupId = "gid1")
   public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
            topic, @Header(KafkaHeaders.OFFSET) long offset) {

       System.out.println("Listener: Message received = "+message+ ", topic= "+topic+ ",
          offset= "+offset);
       if (message.equals("Hello")){
          throw new RuntimeException("Invalid message received");
       }
   }
```

1 retry on exception

Non-blocking retry

Non blocking retries with DLT

```
@Service
public class KafkaConsumer {

   @RetryableTopic(attempts = "2")
   @KafkaListener(topics = "topic1", groupId = "gid1")
   public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
…

   }

   @DltHandler
   public void listenDLT(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
      System.out.println("DLT Received = "+message+ ", topic= "+topic+ ", offset= "+offset);
   }
}
```

After the retries failed, Spring will automatically place this message in the Dead Letter Topic (DLT)

Listener for DTL topic

Non blocking retries with DLT

```
Listener: Message received = Hello World, topic= topic1, offset= 36
Listener: Message received = Hello, topic= topic1, offset= 37
…
Listener: Message received = GoodBy, topic= topic1, offset= 38
Listener: Message received = Hello, topic= topic1-retry, offset= 0
…

DLT Received = Hello, topic= topic1-dlt, offset= 6
```

1 retry on topic1-retry

Non blocking retries with DLT

```
@Service
public class KafkaConsumer {

   @RetryableTopic(attempts = "3")
   @KafkaListener(topics = "topic1", groupId = "gid1")
   public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
…

   }

   @DltHandler
   public void listenDLT(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
      System.out.println("DLT Received = "+message+ ", topic= "+topic+ ", offset= "+offset);
   }
}
```

2 retries

Non blocking retries with DLT

```
Listener: Message received = Hello World, topic= topic1, offset= 39
Listener: Message received = Hello, topic= topic1, offset= 40
Listener: Message received = GoodBy, topic= topic1, offset= 41
...
Listener: Message received = Hello, topic= topic1-retry, offset= 1
...
Listener: Message received = Hello, topic= topic1-retry, offset= 2
...

DLT Received = Hello, topic= topic1-dlt, offset= 7
```

2 retries on topic1-retry

Non blocking retries with DLT

```
@Service
public class KafkaConsumer {
   @RetryableTopic(attempts = "3",
         backoff = @Backoff(delay = 1000, multiplier = 2))
   @KafkaListener(topics = "topic1", groupId = "gid1")
   public void consume(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
…

   }

   @DltHandler
   public void listenDLT(String message, @Header(KafkaHeaders.RECEIVED_TOPIC) String
topic, @Header(KafkaHeaders.OFFSET) long offset) {
      System.out.println("DLT Received = "+message+ ", topic= "+topic+ ", offset= "+offset);
   }
}
```

Backoff algorithm

Non blocking retries with DLT

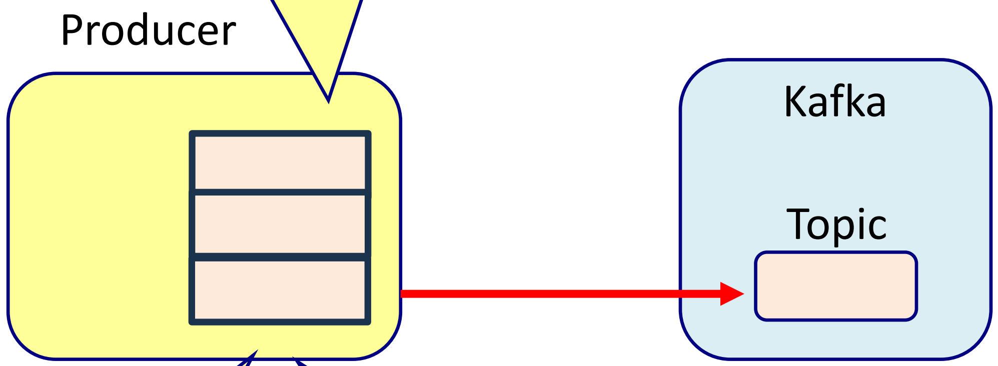
```
Listener: Message received = Hello World, topic= topic1, offset= 42
Listener: Message received = Hello, topic= topic1, offset= 43
...
Listener: Message received = GoodBy, topic= topic1, offset= 44
Listener: Message received = Hello, topic= topic1-retry-1000, offset= 0
..
Listener: Message received = Hello, topic= topic1-retry-2000, offset= 0
..
DLT Received = Hello, topic= topic1-dlt, offset= 8
```

Send message to new topic every retry

BATCH MESSAGES

Producer batching

Messages can be buffered before they are sent to Kafka as a batch of messages



Advantage:
Increase **throughput** because we have less calls to Kafka

Disadvantage:
Increased **latency**: small delay before messages are sent to Kafka

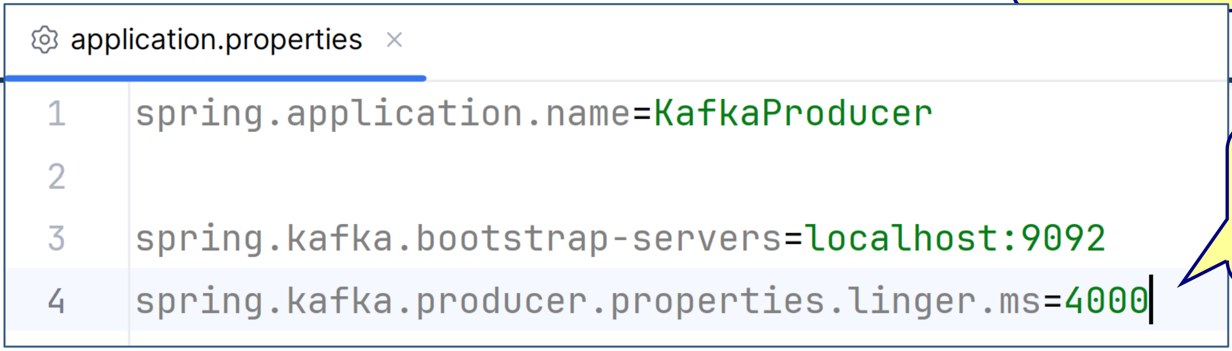
Producer batch settings

- **linger.ms**
  - Number of milliseconds a producer is willing to wait before sending a batch out.
  - Default value is 0, which means "send the messages right away".
- **batch.size**
  - Maximum number of bytes that will be included in a batch
  - Default value is 16KB

Producer

```java
@Service
public class KafkaProducer2 {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage() throws InterruptedException {
        for (int x=1; x<13 ; x++){
            kafkaTemplate.send("topic1" ,"Message-"+x);
            System.out.println("sending message-"+x+" at "+ LocalTime.now().getSecond());
            Thread.sleep(1000);
        }
    }
}
```

Send message every second



application.properties

```properties
spring.application.name=KafkaProducer

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.producer.properties.linger.ms=4000
```

Batch all messages for 4 seconds

Producer

```java
public void sendMessage() throws InterruptedException {
    for (int x=1; x<13 ; x++){
        kafkaTemplate.send("topic1" ,"Message-"+x);
        System.out.println("sending message-"+x+" at "+ LocalTime.now().getSecond());
        Thread.sleep(1000);
    }
}
```

```
sending message-1 at 29
sending message-2 at 30
sending message-3 at 31
sending message-4 at 32
sending message-5 at 33
sending message-6 at 34
sending message-7 at 35
sending message-8 at 36
sending message-9 at 37
sending message-10 at 38
sending message-11 at 39
sending message-12 at 40
```

Sending a message every second

Consumer

```
@Service
public class KafkaConsumer2 {

  @KafkaListener(topics = "topic1", groupId = "gid1")
  public void consume(Message<String> message) {
    System.out.println("Receiving message = "+message.getPayload()+" at "+
           LocalTime.now().getSecond());
  }
}
```

```
Receiving message = Message-1 at 33
Receiving message = Message-2 at 33
Receiving message = Message-3 at 33
Receiving message = Message-4 at 33
Receiving message = Message-5 at 37
Receiving message = Message-6 at 37
Receiving message = Message-7 at 37
Receiving message = Message-8 at 37
Receiving message = Message-9 at 41
Receiving message = Message-10 at 41
Receiving message = Message-11 at 41
Receiving message = Message-12 at 41
```

4 messages every 4 seconds

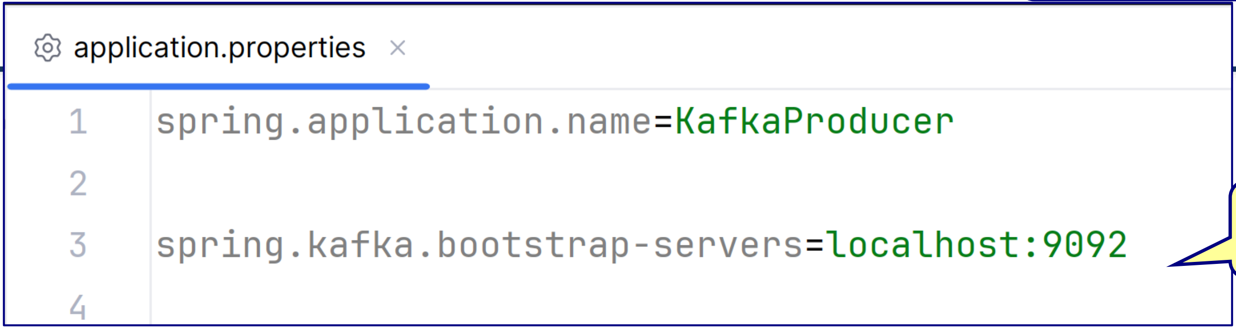
Consumer batch settings

- **Max.poll.records**
  - The maximum number of messages your consumer will receive in a single poll.
  - Default value is 500
- **fetch.min.bytes**
  - The broker will wait until at least this many bytes of messages are available before responding.
  - Default value is 1
- **fetch.max.wait.ms**
  - The maximum time the broker will wait if fetch.min.bytes isn’t satisfied.
  - Default value is 500 ms

Producer

```java
@Service
public class KafkaProducer2 {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage() throws InterruptedException {
        for (int x=1; x<13 ; x++){
            kafkaTemplate.send("topic1" ,"Message-"+x);
            System.out.println("sending message-"+x+" at "+ LocalTime.now().getSecond());
            Thread.sleep(1000);
        }
    }
}
```

Send message every second



application.properties

```
spring.application.name=KafkaProducer

spring.kafka.bootstrap-servers=localhost:9092
```

No batching

Producer

```
public void sendMessage() throws InterruptedException {
   for (int x=1; x<13 ; x++){
      kafkaTemplate.send("topic1" ,"Message-"+x);
      System.out.println("sending message-"+x+" at "+ LocalTime.now().getSecond());
      Thread.sleep(1000);
   }
}
```

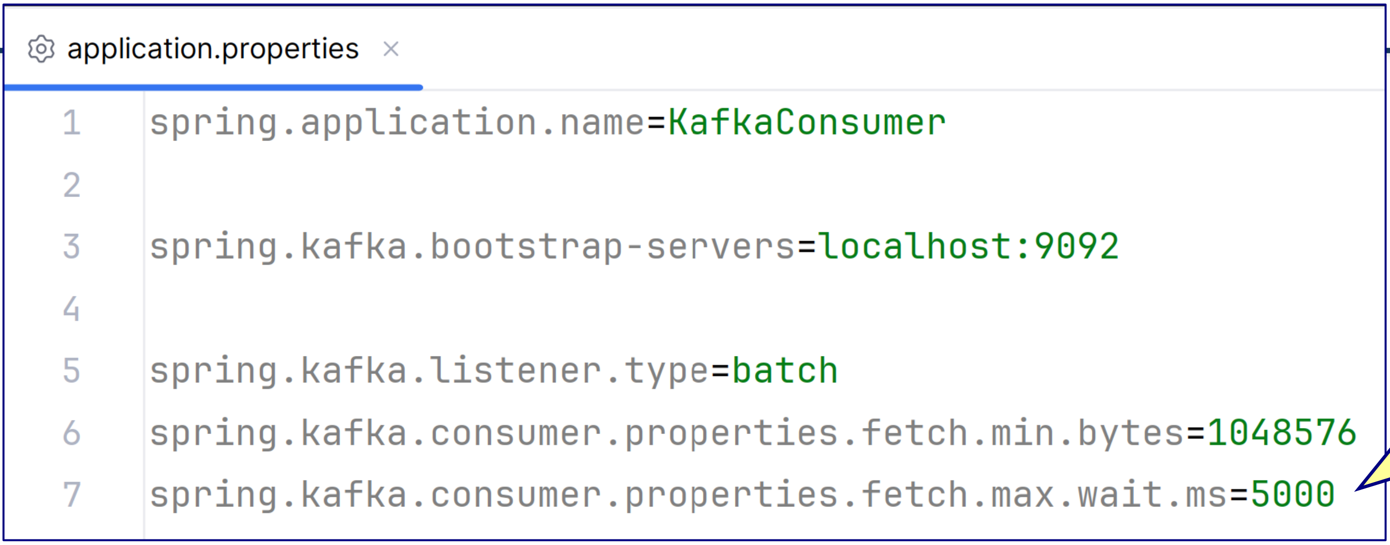
```
sending message-1 at 29
sending message-2 at 30
sending message-3 at 31
sending message-4 at 32
sending message-5 at 33
sending message-6 at 34
sending message-7 at 35
sending message-8 at 36
sending message-9 at 37
sending message-10 at 38
sending message-11 at 39
sending message-12 at 40
```

Sending a message every second

Consumer

```java
@Service
public class KafkaConsumer2 {

    @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(List<Message<String>> messages) {
        for(Message message: messages) {
            System.out.println("Receiving message = " + message.getPayload() + " at " +
                LocalTime.now().getSecond());
        }
    }
}
```

List of messages



application.properties

```
spring.application.name=KafkaConsumer

spring.kafka.bootstrap-servers=localhost:9092

spring.kafka.listener.type=batch
spring.kafka.consumer.properties.fetch.min.bytes=1048576
spring.kafka.consumer.properties.fetch.max.wait.ms=5000
```

Wait for a maximum of 5 seconds

Consumer

```
Receiving message = Message-1 at 19
Receiving message = Message-2 at 19
Receiving message = Message-3 at 19
Receiving message = Message-4 at 19
Receiving message = Message-5 at 24
Receiving message = Message-6 at 24
Receiving message = Message-7 at 24
Receiving message = Message-8 at 24
Receiving message = Message-9 at 24
Receiving message = Message-10 at 29
Receiving message = Message-11 at 29
Receiving message = Message-12 at 29
```

Receive messages in batches

TESTING

Testing Kafka applications

- Using the embedded Kafka broker
- Using Testcontainers

Test using embedded Kafka

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

Producer

```
@Service
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    public void sendMessage(String message) {
        kafkaTemplate.send("topic1", message);
    }
}
```

Consumer

```
@Service
public class KafkaConsumer {
    private String receivedMessage="";


    @KafkaListener(topics = "topic1", groupId = "gid1")
    public void consume(String message) {
        System.out.println("Receiving message = " + message);
        receivedMessage = message;
    }

    public String getReceivedMessage() {
        return receivedMessage;
    }
}
```

Test using embedded Kafka

Create the Spring context

Use the embedded kafka broker

```
@SpringBootTest
@EmbeddedKafka(topics = {"topic1"})
public class KafkaIntegrationTest {
    @Autowired
    private KafkaConsumer consumer;

    @Autowired
    private KafkaProducer producer;

    @Test
    void testKafka() {
        producer.sendMessage("Hello World");
        assertThat(consumer.getReceivedMessage().equals("Hello World"));
    }
}
```

Test using TestContainer

```
<dependency>
   <groupId>org.springframework.kafka</groupId>
   <artifactId>spring-kafka-test</artifactId>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>org.testcontainers</groupId>
   <artifactId>junit-jupiter</artifactId>
   <scope>test</scope>
</dependency>
<dependency>
   <groupId>org.testcontainers</groupId>
   <artifactId>kafka</artifactId>
   <scope>test</scope>
</dependency>
```

Test using TestContainer(1/2)

```
@SpringBootTest
@Testcontainers
public class KafkaIntegrationTest {
    @Container
    static final KafkaContainer kafka = new KafkaContainer(
            DockerImageName.parse("apache/kafka:latest")
    );

    @DynamicPropertySource
    static void overrideProperties(DynamicPropertyRegistry registry) {
        registry.add("spring.kafka.bootstrap-servers", kafka::getBootstrapServers);
    }
```

@Testcontainers

Create a test container based on a Docker image

Test using TestContainer(2/2)

```
    @Autowired
    private KafkaConsumer consumer;

    @Autowired
    private KafkaProducer producer;

    @Test
    void testKafka() {
        producer.sendMessage("Hello World");
        assertThat(consumer.getReceivedMessage().equals("Hello World"));
    }
}
```

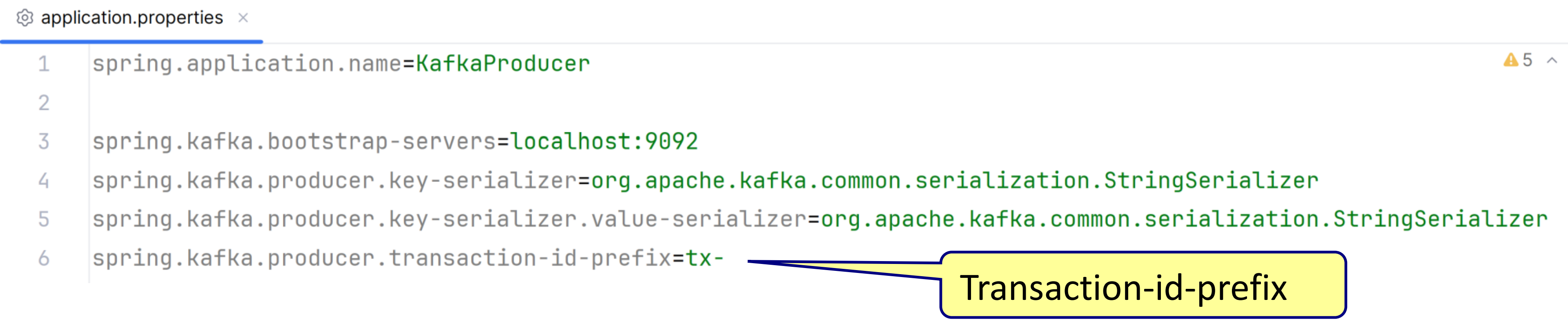
TRANSACTIONS

Producer

```java
@Service
public class KafkaProducer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @Transactional
    public void generateAndSendPackage()
            throws InterruptedException, TransactionException {
        for (long i = 0; i < 10; i++) {
            kafkaTemplate.send("topic8", "Message"+i+" Time ="+ LocalTime.now());
            System.out.println("sending "+"Message"+i+" Time created="+ LocalTime.now());
            Thread.sleep(1000);
        }
    }
}
```

@Transactional

Producer



application.properties

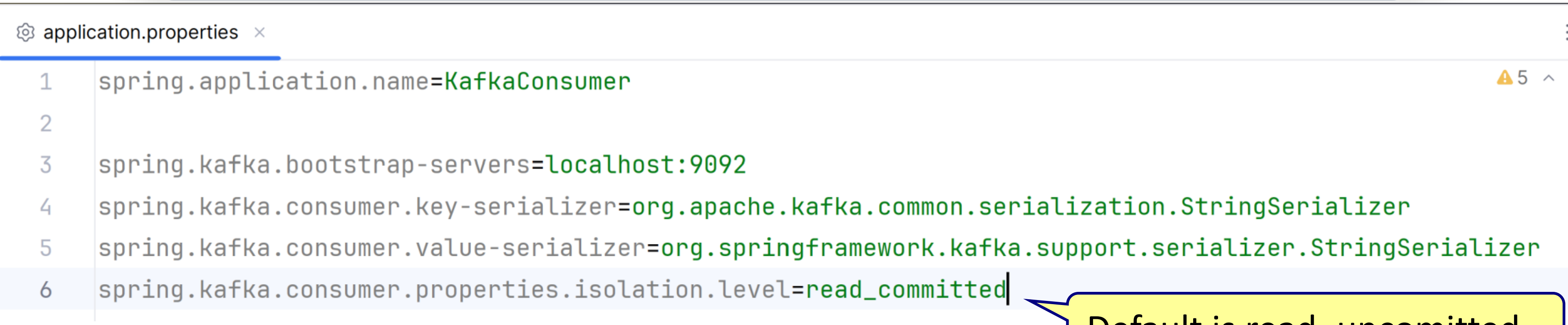
```
spring.application.name=KafkaProducer

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.key-serializer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.transaction-id-prefix=tx-
```

```
Instantiated a transactional producer.
Invoking InitProducerId for the first time in order to acquire a producer ID
Discovered transaction coordinator localhost:9092 (id: 1 rack: null)
ProducerId set to 3000 with epoch 4sending Message0 Time
created=08:22:08.007622300sending Message1 Time
created=08:22:09.023629900sending Message2 Time
…
created=08:22:15.075196200sending Message8 Time
created=08:22:16.077350600sending Message9 Time
created=08:22:17.090879600Closing the Kafka producer with timeoutMillis = 30000 ms.
```

Consumer

```java
@Service
public class KafkaConsumer {
    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;
    @Transactional
    @KafkaListener(topics = "topic8", groupId = "gid1")
    public void consume(String message) {
        System.out.println("Consumer receiving message = " + message+" Time
                received = "+ LocalTime.now());
    }
}
```

@Transactional



```properties
spring.application.name=KafkaConsumer

spring.kafka.bootstrap-servers=localhost:9092
spring.kafka.consumer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.consumer.value-serializer=org.springframework.kafka.support.serializer.StringSerializer
spring.kafka.consumer.properties.isolation.level=read_committed
```

Default is read_uncomitted

Consumer

```
Consumer receiving message = Message0 Time =08:33:05.944749900 Time received =
08:33:16.101232800
Consumer receiving message = Message1 Time =08:33:07.011881300 Time received =
08:33:16.104627700
Consumer receiving message = Message2 Time =08:33:08.018925500 Time received =
08:33:16.104627700
…
Consumer receiving message = Message8 Time =08:33:14.054309500 Time received =
08:33:16.106232900
Consumer receiving message = Message9 Time =08:33:15.060405900 Time received =
08:33:16.106232900
```

All messages received at the same time (when the sender committed the messages)

Producer

```
@Service
public class KafkaProducer {

  @Autowired
  private KafkaTemplate<String, String> kafkaTemplate;

  @Transactional
  public void generateAndSendPackage()
      throws InterruptedException, TransactionException {
    for (long i = 0; i < 10; i++) {
      kafkaTemplate.send("topic8", "Message"+i+" Time ="+ LocalTime.now());
      if ( i > 5)
        throw new RuntimeException();
      System.out.println("sending "+"Message"+i+" Time created="+ LocalTime.now());
      Thread.sleep(1000);
    }
  }
}
```

Exception within transaction: rollback

Producer

```
Instantiated a transactional producer
....
Discovered transaction coordinator localhost:9092 (id: 1 rack: null)
ProducerId set to 3000 with epoch 6
sending Message0 Time created=08:37:25.019418100sending Message1 Time
created=08:37:26.025871200sending Message2 Time
created=08:37:27.035123100sending Message3 Time
created=08:37:28.037490100sending Message4 Time
created=08:37:29.047949800sending Message5 Time
created=08:37:30.054130400Aborting incomplete transaction
...
Closing the Kafka producer with timeoutMillis = 30000 ms.
```

These messages are removed from Kafka

The consumer does not receive any message