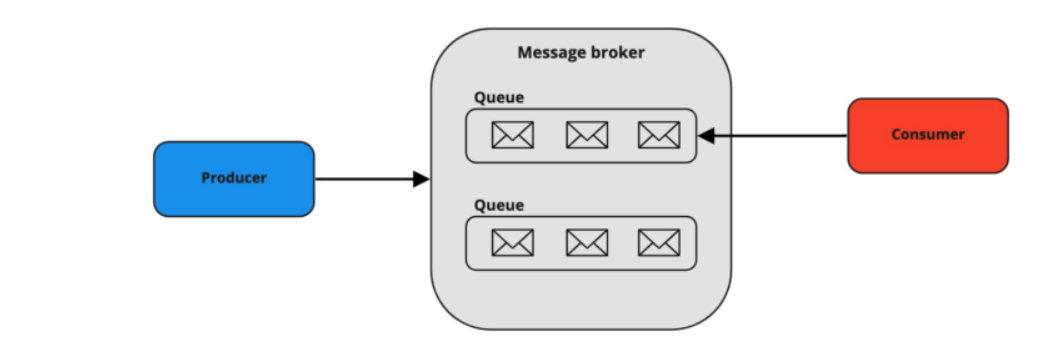
**Kafka:**

[Apache Kafka](https://kafka.apache.org/) is an open-source distributed event streaming service for high-performance data pipelines, streaming analytics and data integration applications. It is based on a *publish/subscribe messaging system* and is often called a “*distributed commit log “*.



It possesses three key capabilities:

* It enables applications to publish or subscribe data or event streams.
* Ensures accurate storage of records in a fault tolerant and durable manner.’
* Real time processing of records.

**Kafka Cluster:**

Since Kafka is a distributed system, it acts as a cluster. A Kafka cluster consists of a set of brokers.

**Kafka Cluster**



**Kafka Broker:**

The broker is the Kafka server. It's just a meaningful name given to the Kafka server. And this name makes sense as well because all that Kafka does is act as a message broker between producer and consumer.

The producer and consumer don't interact directly. They use the Kafka server as an agent or a broker to exchange messages.



**Kafka Producer:**

Producer is an application that sends messages. It does not send messages directly to the recipient. It sends messages only to the Kafka server.

**Kafka Consumer:**

Consumer is an application that reads messages from the Kafka server.

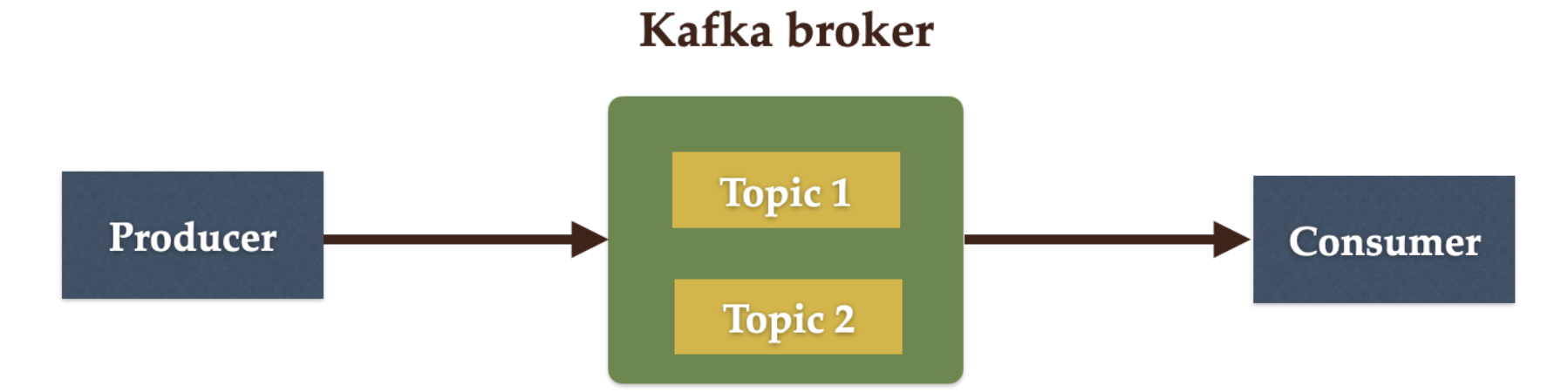
If producers are sending data, they must be sending it to someone, right? The consumers are the recipients. But remember that the producers don't send data to a recipient address. They just send it to the Kafka server.

Anyone who is interested in that data can come forward and take it from the Kafka server. So, any application that requests data from a Kafka server is a consumer, and they can ask for data sent by any producer provided they have permission to read it.

**Kafka Topic:**

Topic is like a table in a database or folder in a file system.

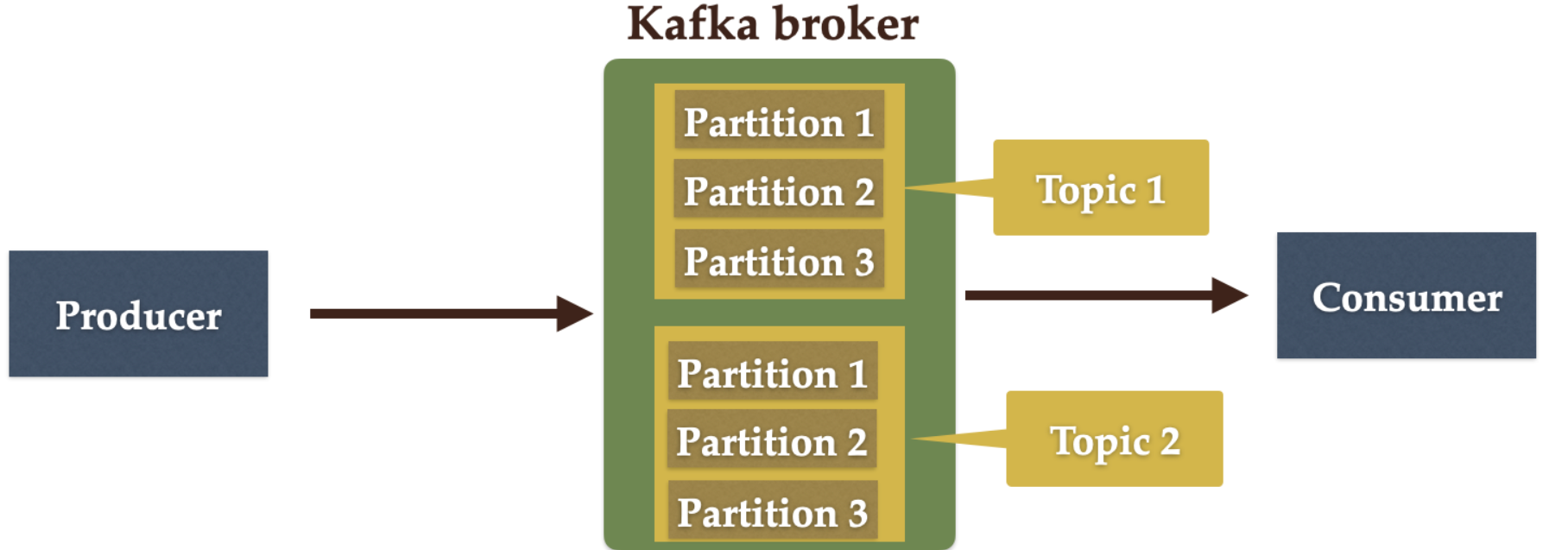
It is identified by a name.There can be multiple topics.



**Kafka Partitions:**

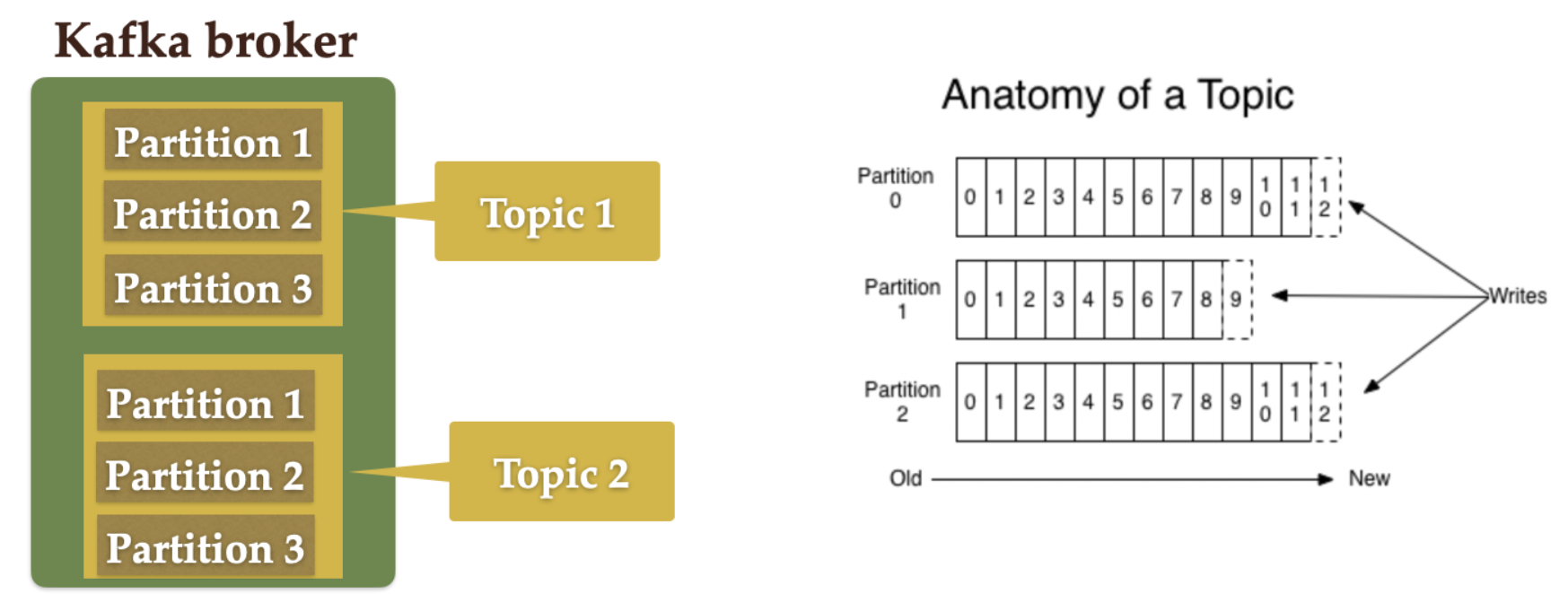
Kafka topics are divided into a number of partitions, which contain records in an unchangeable sequence.

Kafka Brokers will store messages for a topic. But the capacity of data can be enormous and it may not be possible to store in a single computer. Therefore it will be partitioned into multiple parts and distributed among multiple computers since Kafka is a distributed system.



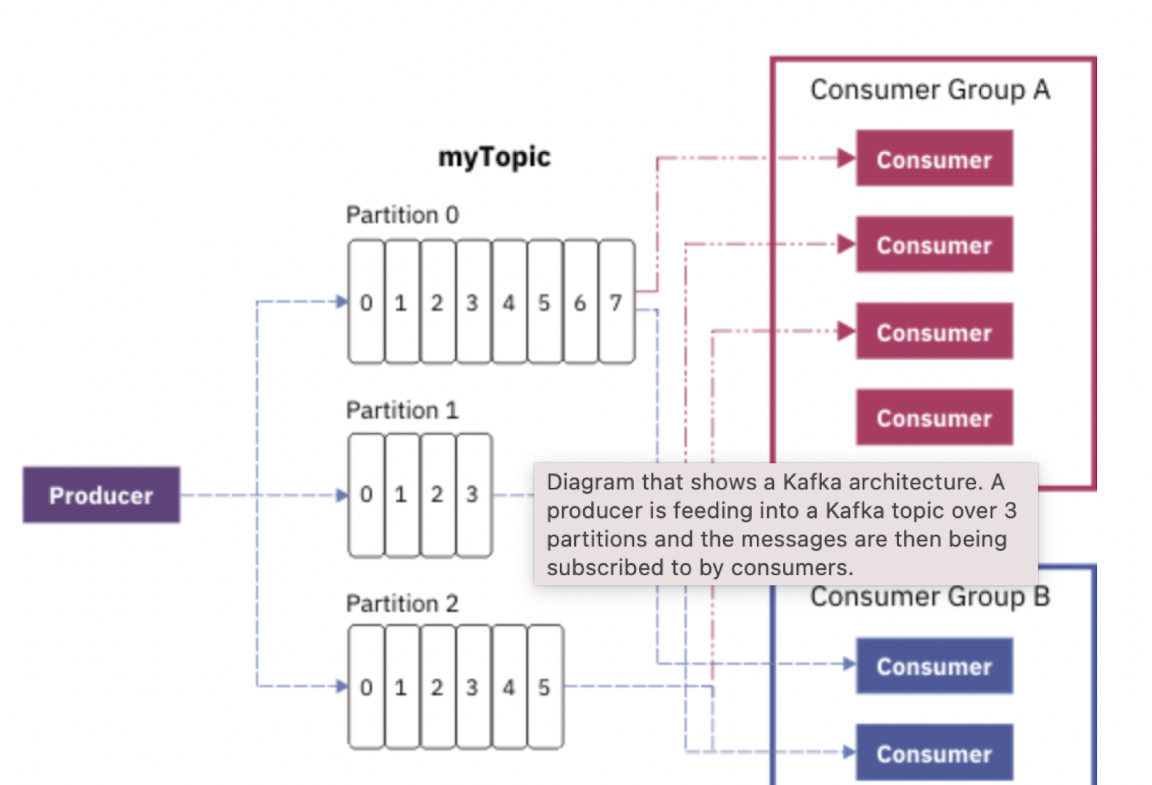
**Kafka Offsets:**

Offset is a sequence of ids given to messages as they arrive at a partition. Once the offset is assigned it will never be changed. The first message gets an offset zero. The next message receives an offset one and so on.



**Kafka Consumer Group:**

A consumer group contains one or more consumers working together to process the messages.

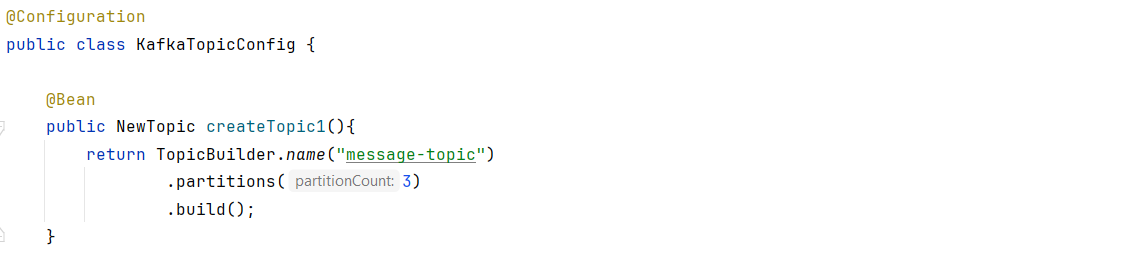


**Kafka with Spring Boot**

To configure and connect Kafka with Spring Boot application we add *spring-kafka* dependency.

**Creating Kafka Topics:**

To create a new topic, we can define the *NewTopic* beans in a *@Configuration* class that will be loaded during application startup. It will create a new topic only if there is no existing topic with the same name.

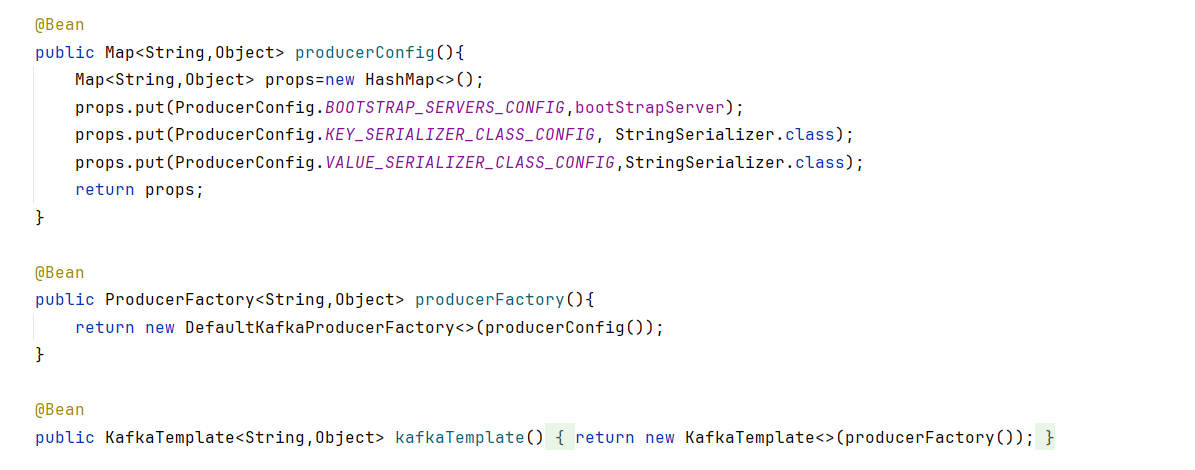


**Producing Messages:**

To create messages, we first need to configure a [*ProducerFactory*](http://docs.spring.io/spring-kafka/api/org/springframework/kafka/core/ProducerFactory.html). This sets the strategy for creating Kafka [*Producer*](https://kafka.apache.org/0100/javadoc/org/apache/kafka/clients/producer/Producer.html) instances.

Then we need a [***KafkaTemplate***](http://docs.spring.io/spring-kafka/api/org/springframework/kafka/core/KafkaTemplate.html) which wraps a Producer instance and provides convenient methods for sending messages to kafka topics. Producer instances are thread safe. So, using a single instance throughout an application context will give higher performance. KafkaTemplate instances are also thread safe and use of one instance is recommended.

Producer Configuration:



**Publishing Messages:**

Messages can be published using KafkaTemplate Class.

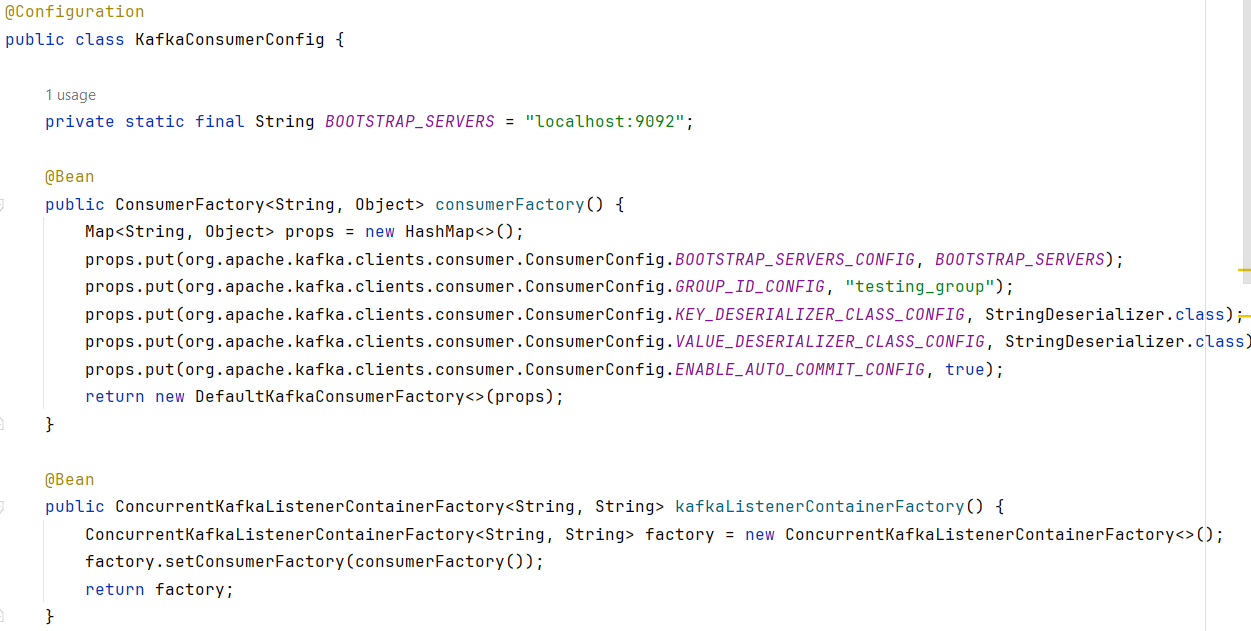


The *send* API returns a *CompletableFuture* object. If we want to block the sending thread and get the result about the sent message, we can call the *get* API of the *CompletableFuture* object. The thread will wait for the result, but it will slow down the producer.

Kafka is a fast-stream processing platform. Therefore, it’s better to handle the results asynchronously so that the subsequent messages do not wait for the result of the previous message.

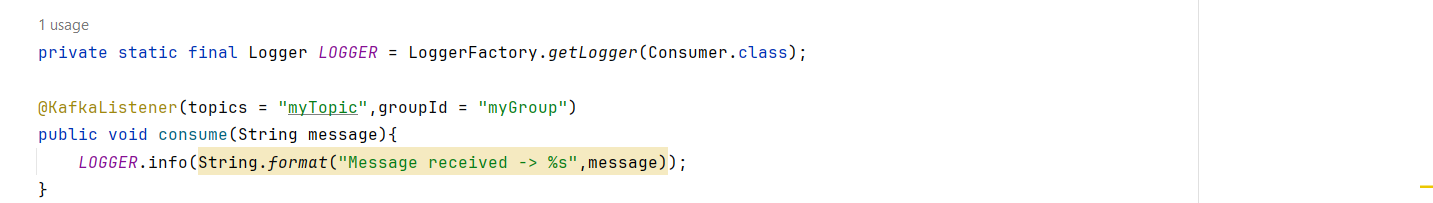
**Consuming Messages:**

For consuming messages, we need to configure a [*ConsumerFactory*](http://docs.spring.io/autorepo/docs/spring-kafka-dist/1.1.3.RELEASE/api/org/springframework/kafka/core/ConsumerFactory.html) and a [*KafkaListenerContainerFactory*](http://docs.spring.io/autorepo/docs/spring-kafka-dist/1.1.3.RELEASE/api/org/springframework/kafka/config/KafkaListenerContainerFactory.html).



The *@KafkaListener* annotation is used to designate a bean method as a listener for a specific topic(s). It simplifies the process of building a Kafka consumer and internally utilizes *ConcurrentMessageListenerContainer* for message consumption.

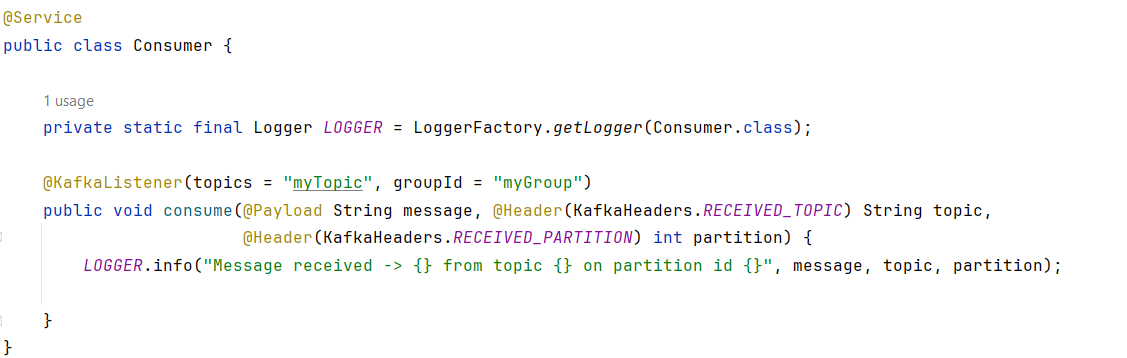
[*@EnableKafka*](http://docs.spring.io/autorepo/docs/spring-kafka-dist/1.1.3.RELEASE/api/org/springframework/kafka/annotation/EnableKafka.html) annotation is required on the configuration class to enable the detection of *@KafkaListener* annotation on spring-managed beans:

****

We can implement multiple listeners for a topic, each with a different group Id. Furthermore, one consumer can listen for messages from various topics.

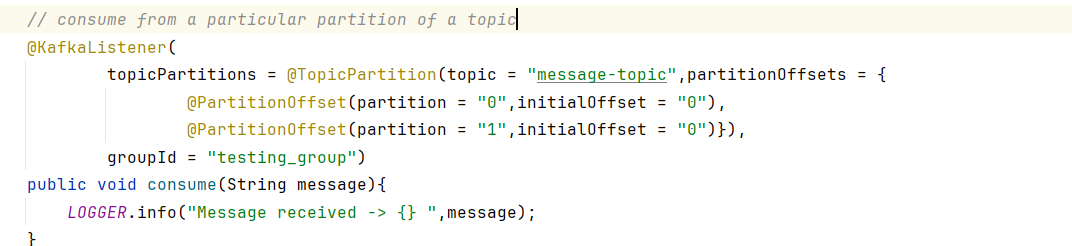
**@KafkaListener(topics ={ "topic1, topic2"}, groupId = "myGroup")**

Spring also supports retrieval of messages using [*@Payload*](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/messaging/handler/annotation/Payload.html) and headers info using [*@Header*](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/messaging/handler/annotation/Header.html) annotation



**Consuming Messages from a specific partition:**

For a topic with multiple partitions, however, a *@KafkaListener* can explicitly subscribe to a particular partition of a topic with an initial offset.



Since the *initialOffset* has been set to 0 in this listener, all the previously consumed messages from partitions 0 and 3 will be re-consumed every time this listener is initialized.

If we don’t need to set the offset, we can use the *partitions* property of *@TopicPartition* annotation to set only the partitions without the offset:

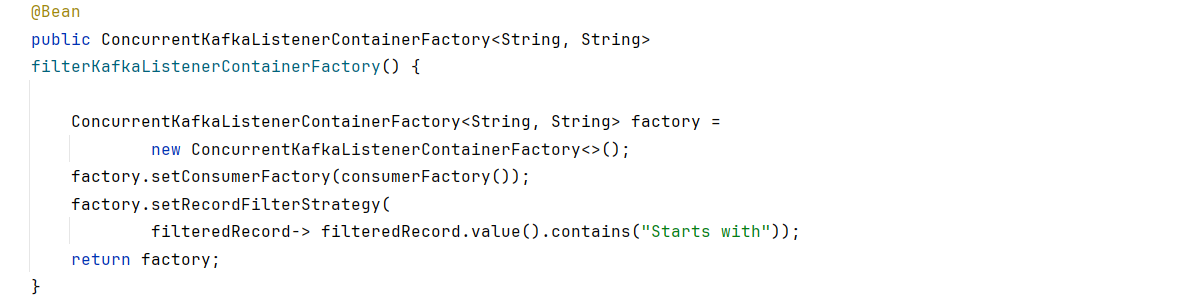
**@KafkaListener(topicPartitions**

**= @TopicPartition(topic = "topicName", partitions = { "0", "1" }))**

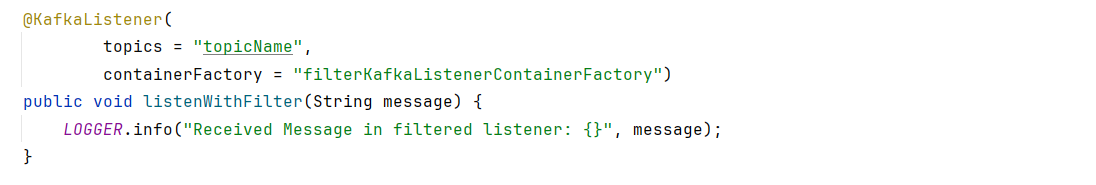
### 

### **Adding Message Filter for Listeners:**

We can configure listeners to consume specific message content by adding a custom filter. This can be done by setting a [*RecordFilterStrategy*](http://docs.spring.io/spring-kafka/api/org/springframework/kafka/listener/adapter/RecordFilterStrategy.html) to the *KafkaListenerContainerFactory*



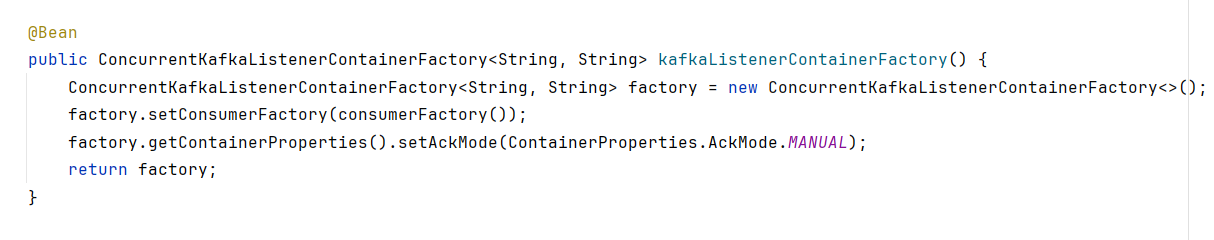
We can then configure a listener to use this container factory.In this listener, all the messages matching the filter will be discarded.



### **Acknowledgement and Offset Commit:**

Acknowledgement in Apache Kafka is a mechanism used by consumers to inform Kafka brokers that they have successfully processed a message from a topic partition. This mechanism is crucial for ensuring message delivery guarantees and managing offsets properly.

When a consumer receives a message from a Kafka topic partition, it needs to acknowledge that message to indicate successful processing.



To enable manual acknowledgement, you need to set the AckMode of the Kafka listener container to MANUAL. This is done by configuring the ConcurrentKafkaListenerContainerFactory bean in your Spring Kafka configuration, as shown in the provided example.



The consumer processes the message (e.g., performs business logic, updates database), and then explicitly calls the acknowledge() method on the Acknowledgement object to acknowledge the successful processing of the message.

Committing offsets: When the consumer acknowledges a message, it also commits the corresponding offset to Kafka. This offset represents the position of the consumer in the topic partition. Kafka uses offsets to track the progress of consumers and ensure that messages are not processed more than once.

### **Configuring Concurrency**

The *ConcurrentMessageListener* provides the ability to set up multiple listeners within the same application using the setConcurrency() method.

@Bean

ConcurrentKafkaListenerContainerFactory<?, ?> kafkaListenerContainerFactory(...) {

ConcurrentKafkaListenerContainerFactory<Object, Object> factory = new ConcurrentKafkaListenerContainerFactory<>();

//...

factory.setConcurrency(3);

return factory;

}

### **Spring Boot Kafka JsonSerializer and JsonDeserializer:**

In Spring Boot with Apache Kafka, you can use JsonSerializer and JsonDeserializer to serialize and deserialize Java objects to and from JSON format when producing and consuming messages, respectively. This allows you to work with JSON data seamlessly within your Kafka applications.

@Bean

public ProducerFactory<String, YourMessageClass> producerFactory() {

Map<String, Object> config = new HashMap<>();

config.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9092");

return new DefaultKafkaProducerFactory<>(config, new StringSerializer(),

new JsonSerializer<>());

}

@Bean

public ConsumerFactory<String, YourMessageClass> consumerFactory() {

Map<String, Object> config = new HashMap<>();

config.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9092");

return new DefaultKafkaConsumerFactory<>(config, new StringDeserializer(),

new JsonDeserializer<>(YourMessageClass.class));

}

By configuring JsonSerializer and JsonDeserializer, you can seamlessly serialize and deserialize JSON messages in your Spring Boot Kafka applications, allowing for easy integration with systems that use JSON data.

**Refrences:**

<https://www.baeldung.com/spring-kafka>

<https://howtodoinjava.com/spring-boot/apache-kafka-using-spring-boot/>

<https://www.javaguides.net/2022/06/spring-boot-apache-kafka-tutorial.html>

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