Kafka Documentation by Smit Joshi

Create the Kafka user
- `sudo adduser kafka`

Add Kafka User to sudo group
- `sudo adduser kafka sudo`

login to kafka account
- `su -l kafka`

Install JDK if not exists
- `sudo apt update`
- `sudo apt install openjdk-17-jdk`

Download kafka
- `mkdir ~/downloads`
- `cd ~/downloads`
- `wget https://archive.apache.org/dist/kafka/3.4.0/kafka_2.12-3.4.0.tgz`

Download offset exproler `and install it`
- `https://www.kafkatool.com/download.html`

Extract kafka
- `cd ~`
- `tar -xvzf ~/downloads/kafka_2.12-3.4.0.tgz`

Rename directory `kafka_2.12-3.4.0.tgz to kafka`
- `mv kafka_2.12-3.4.0/ kafka/`

`Install Vim if you don't have`
- `sudo apt install vim`

Configure Kafka Server
- `vim ~/kafka/config/server.properties`
- `change the value of` **`num.partition`** `to` **`3`**

Start the Zookeeper
- `~/bin/zookeeper-server-start.sh ~/kafka/config/zookeeper.properties`

Start Kafka
- `~/kafka/bin/kafka-server-start.sh ~/kafka/config/server.properties`

Create the Service files so that we don't have to write the path every time we wan't sto start the kafka and the zookeeper.
- `sudo vim /etc/systemd/system/zookeeper.service`
- `and drop the following content there.`

```
[Unit]
Description=Apache Zookeeper Service
Requires=network.target
After=network.target

[Service]
Type=simple
User=kafka
ExecStart=/home/kafka/kafka/bin/zookeeper-server-start.sh
/home/kafka/kafka/config/zookeeper.properties
ExecStop=/home/kafka/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

Same way let's create the unit file for the kafka

- `sudo nano /etc/systemd/system/kafka.service`

```
[Unit]
Description=Apache Kafka Service that requires zookeeper service
Requires=zookeeper.service
After=zookeeper.service

[Service] Type=simple
User=kafka
ExecStart= /home/kafka/kafka/bin/kafka-server-start.sh
/home/kafka/kafka/config/server.properties
ExecStop=/home/kafka/kafka/bin/kafka-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

Now you can manage the kafka by executing the following commands

- `sudo systemctl start kafka`
- `sudo systemctl status kafka`
- `sudo systemctl stop kafka`

Testing the kafka server

- `nc -vz localhost 9092`

To Check the logs

- `cat ~/kafka/logs/server.log`

Creating Topic

- `~/kafka/bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic firstTopic`

To check available topics

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

To produce message to the topic

- `~/kafka/bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic firstTopic`

To read the message

- `~/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic firstTopic –from-beginning`

A Simple Kafka Producer Example

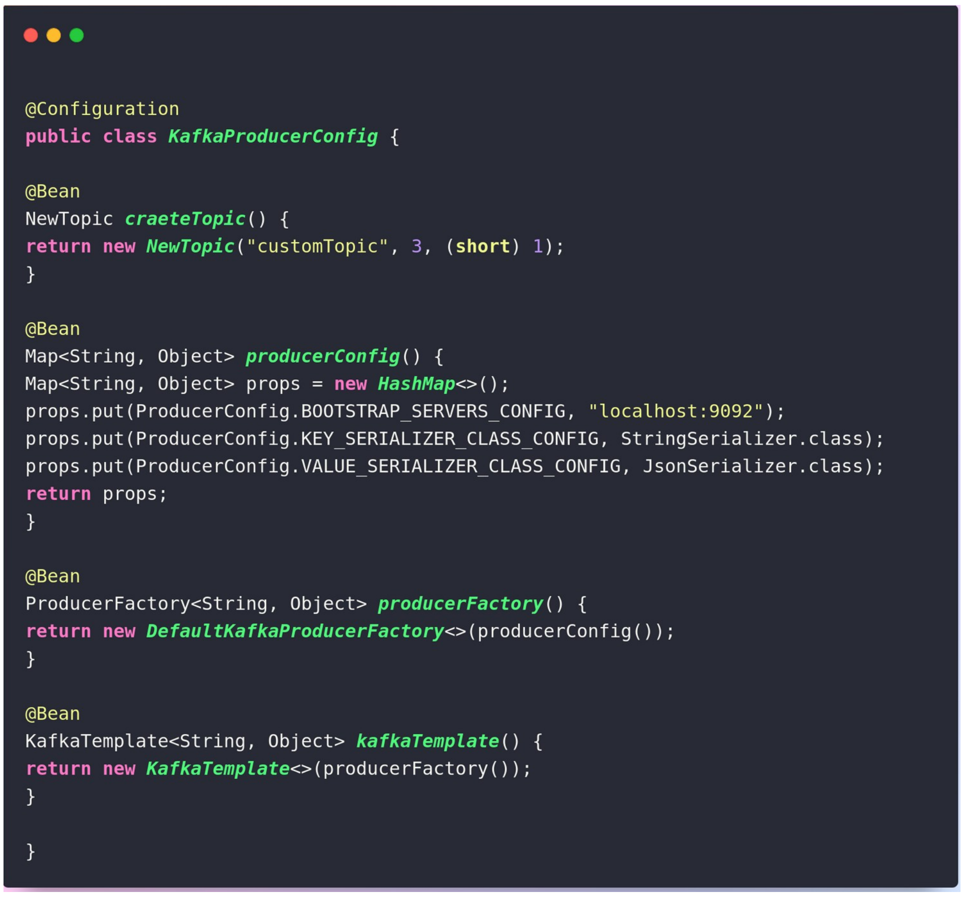
https://github.com/smit-joshi-addon/kafka-producer-example.git

a simple producer which will produce the data and put it int the topic named **customTopic** if there is no topic present it will create one with default configuration. Which will be most likely a topic with single partition, but if you have configured the num.partitions to 3 in the server.properties then it wll create the 3 partitions using the default configurations from the server.properties.

You can also create the configuration class conatining the required information to create the topic on the fly.

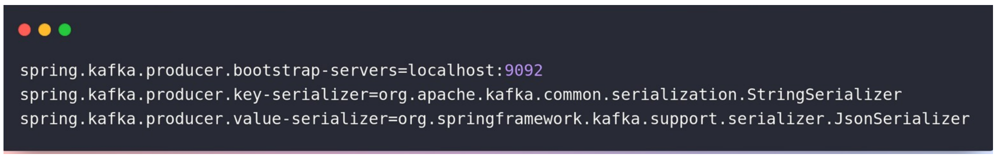
```
@Bean
NewTopic createTopic() {
      return new NewTopic("customTopic", 3, (short) 1);
}
```

Here’s the sample class containg the java based configuration



```
@Configuration
public class KafkaProducerConfig {

@Bean
NewTopic craeteTopic() {
return new NewTopic("customTopic", 3, (short) 1);
}

@Bean
Map<String, Object> producerConfig() {
Map<String, Object> props = new HashMap<>();
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
return props;
}

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

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

}
```

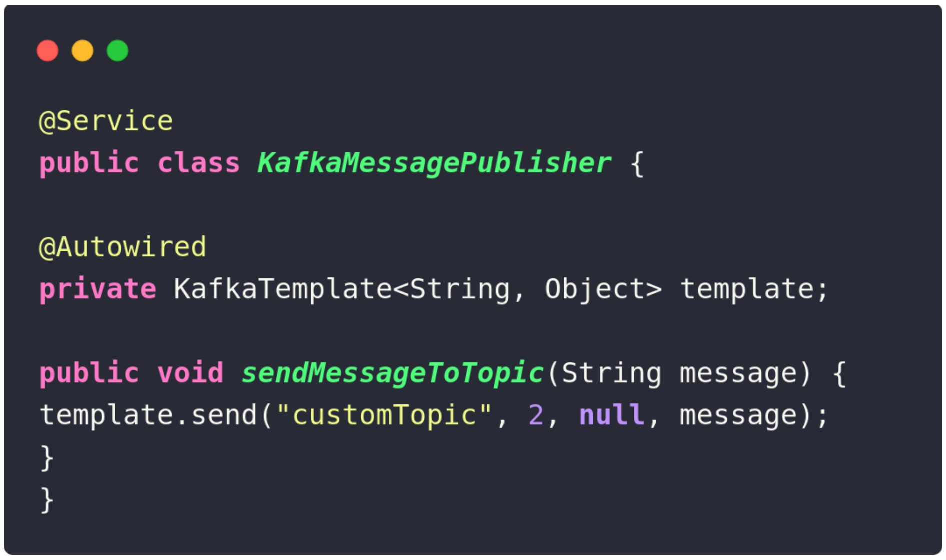
here’s the java based configuration



```
spring.kafka.producer.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
```

The StringSerializer and JsonSerializer will be used to convert the object to the json when the object is published to the kafka topic.

Here’s the sample class publishing the given String message to the topic.



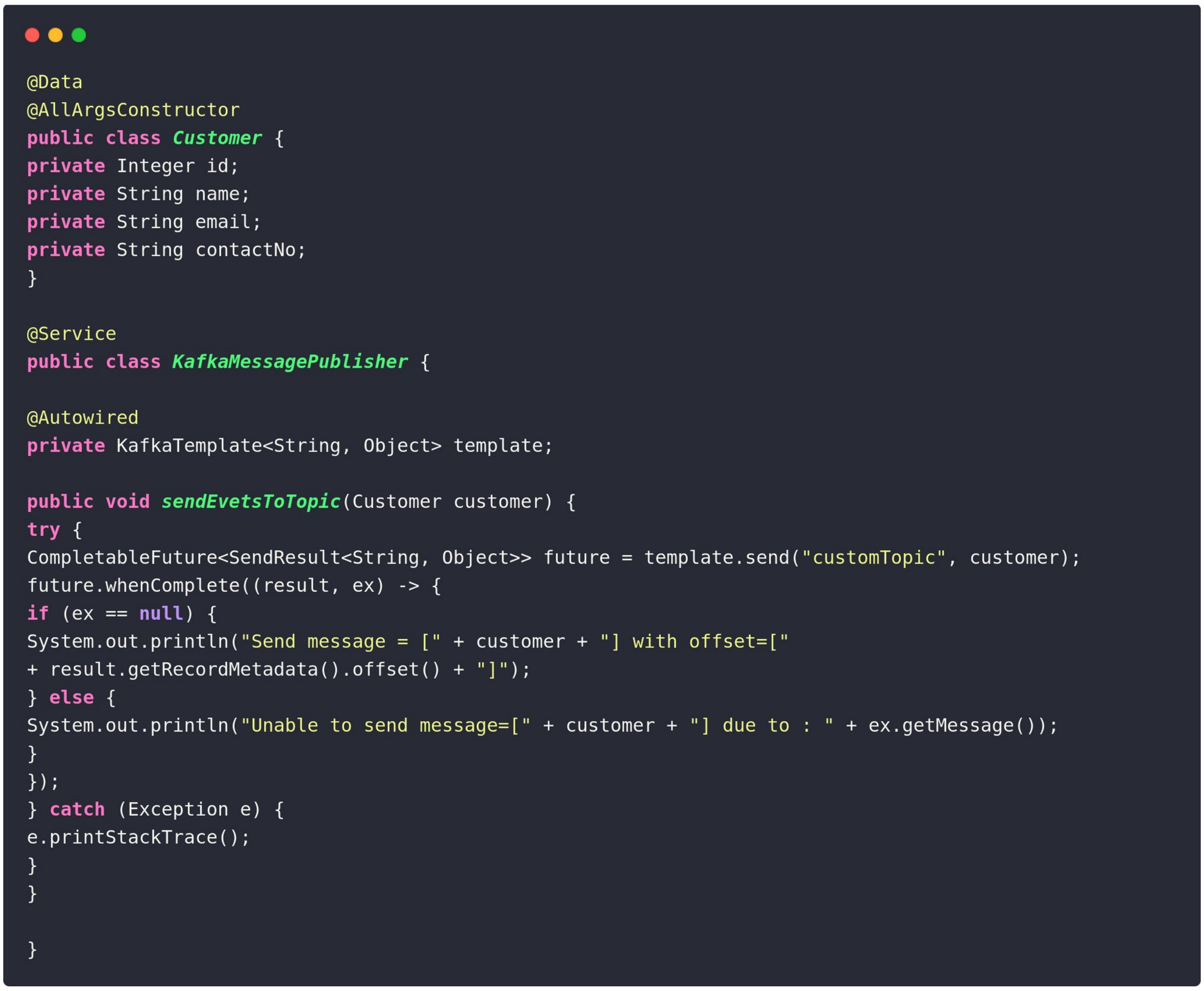
```java
@Service
public class KafkaMessagePublisher {

@Autowired
private KafkaTemplate<String, Object> template;

public void sendMessageToTopic(String message) {
template.send("customTopic", 2, null, message);
}
}
```

here it will publish the message to the customTopic, again if there is no topic named “customTopic” it will create the topic and then push the message into it.

So how to publish an object ?

To publish a custom object for example Customer object, you’ll have to define the StringSerializer and the JsonSerializer as the key,value serializers in either the properties or tha java based configurations.

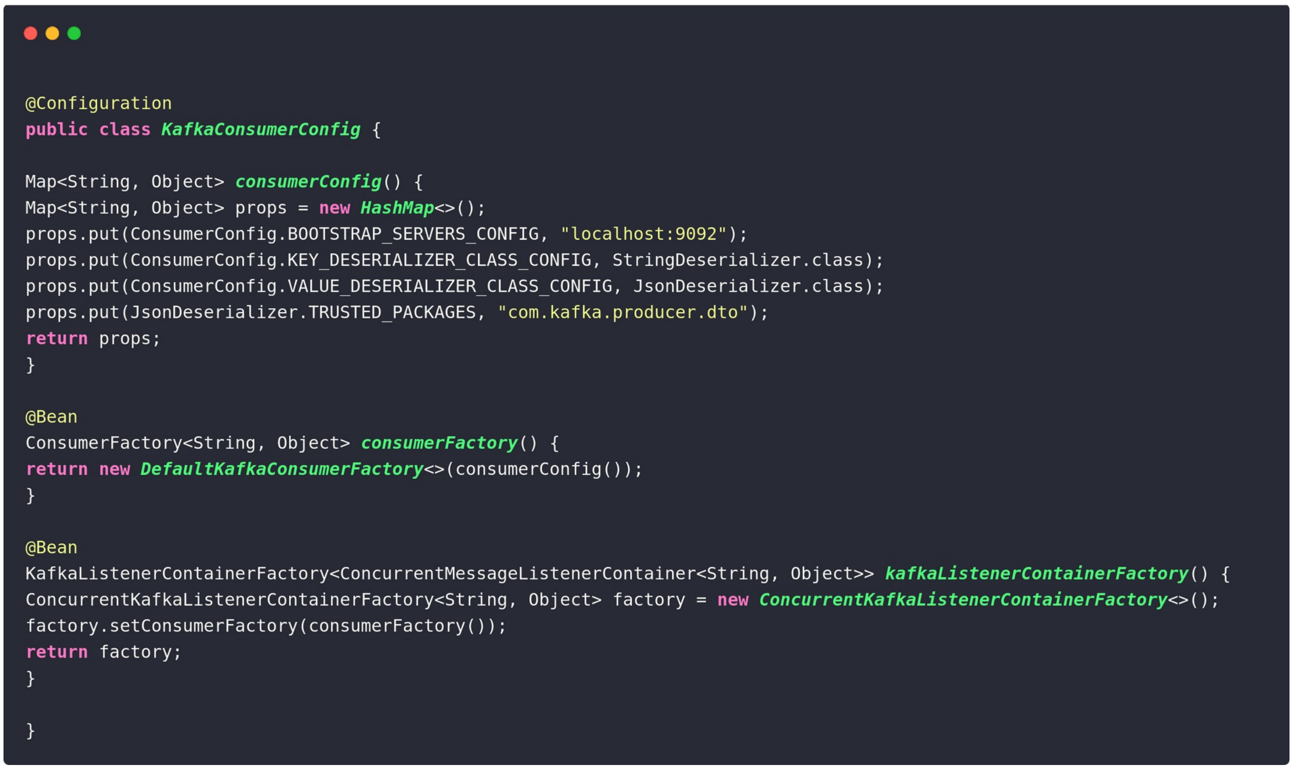
Here’s an example demonstarting the usecase.



```
@Data
@AllArgsConstructor
public class Customer {
private Integer id;
private String name;
private String email;
private String contactNo;
}

@Service
public class KafkaMessagePublisher {

@Autowired
private KafkaTemplate<String, Object> template;

public void sendEvetsToTopic(Customer customer) {
try {
CompletableFuture<SendResult<String, Object>> future = template.send("customTopic", customer);
future.whenComplete((result, ex) -> {
if (ex == null) {
System.out.println("Send message = [" + customer + "] with offset=["
+ result.getRecordMetadata().offset() + "]");
} else {
System.out.println("Unable to send message=[" + customer + "] due to : " + ex.getMessage());
}
});
} catch (Exception e) {
e.printStackTrace();
}
}

}
```

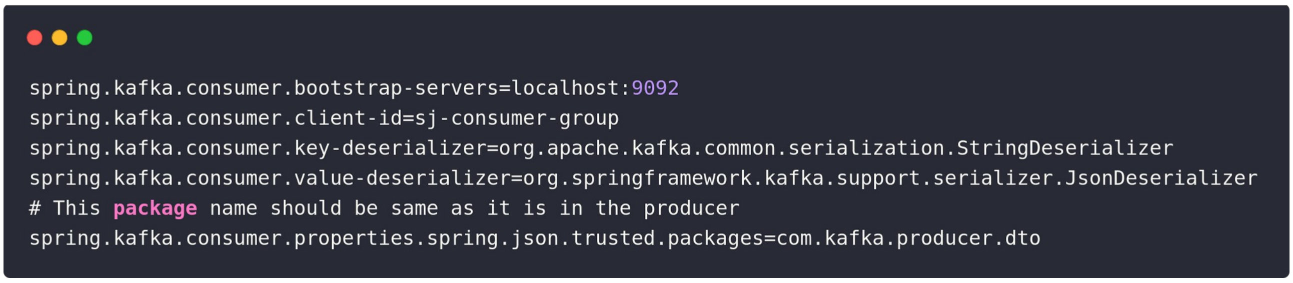
Consumeing the messages
checkout the repo https://github.com/smit-joshi-addon/kafka-consumer-example.git

configs
Java based config



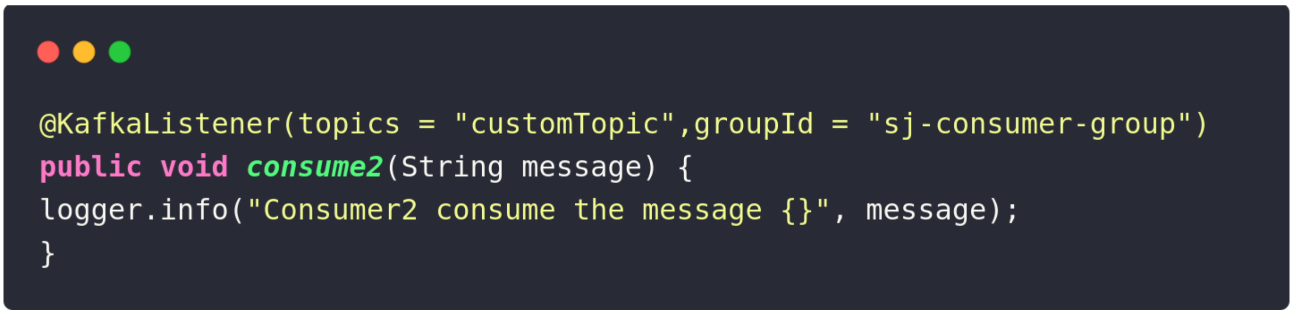
```java
@Configuration
public class KafkaConsumerConfig {

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.kafka.producer.dto");
return props;
}

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

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

}
```

properties based config



```properties
spring.kafka.consumer.bootstrap-servers=localhost:9092
spring.kafka.consumer.client-id=sj-consumer-group
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer
# This package name should be same as it is in the producer
spring.kafka.consumer.properties.spring.json.trusted.packages=com.kafka.producer.dto
```

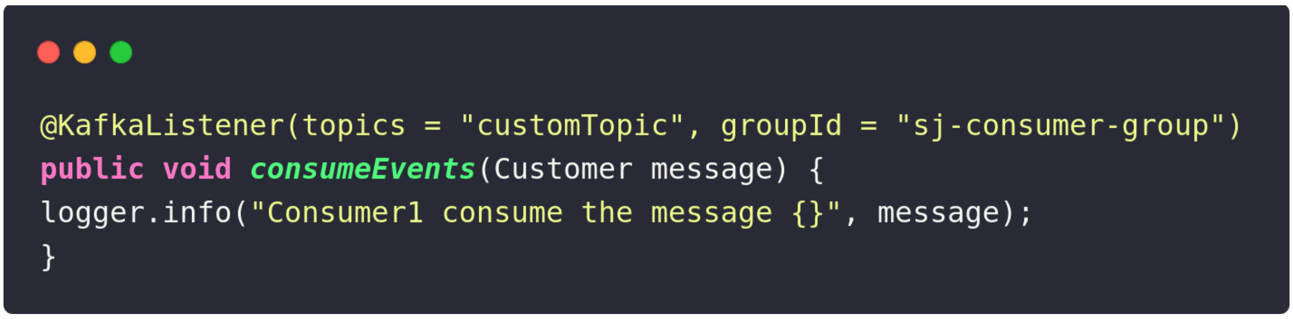
this will listn the simple String message



```java
@KafkaListener(topics = "customTopic",groupId = "sj-consumer-group")
public void consume2(String message) {
logger.info("Consumer2 consume the message {}", message);
}
```

to consume the Objects like Customer you'll have to add the StringDesirializer and the JsonDeserializer in the configuration.

An example listning the Customer

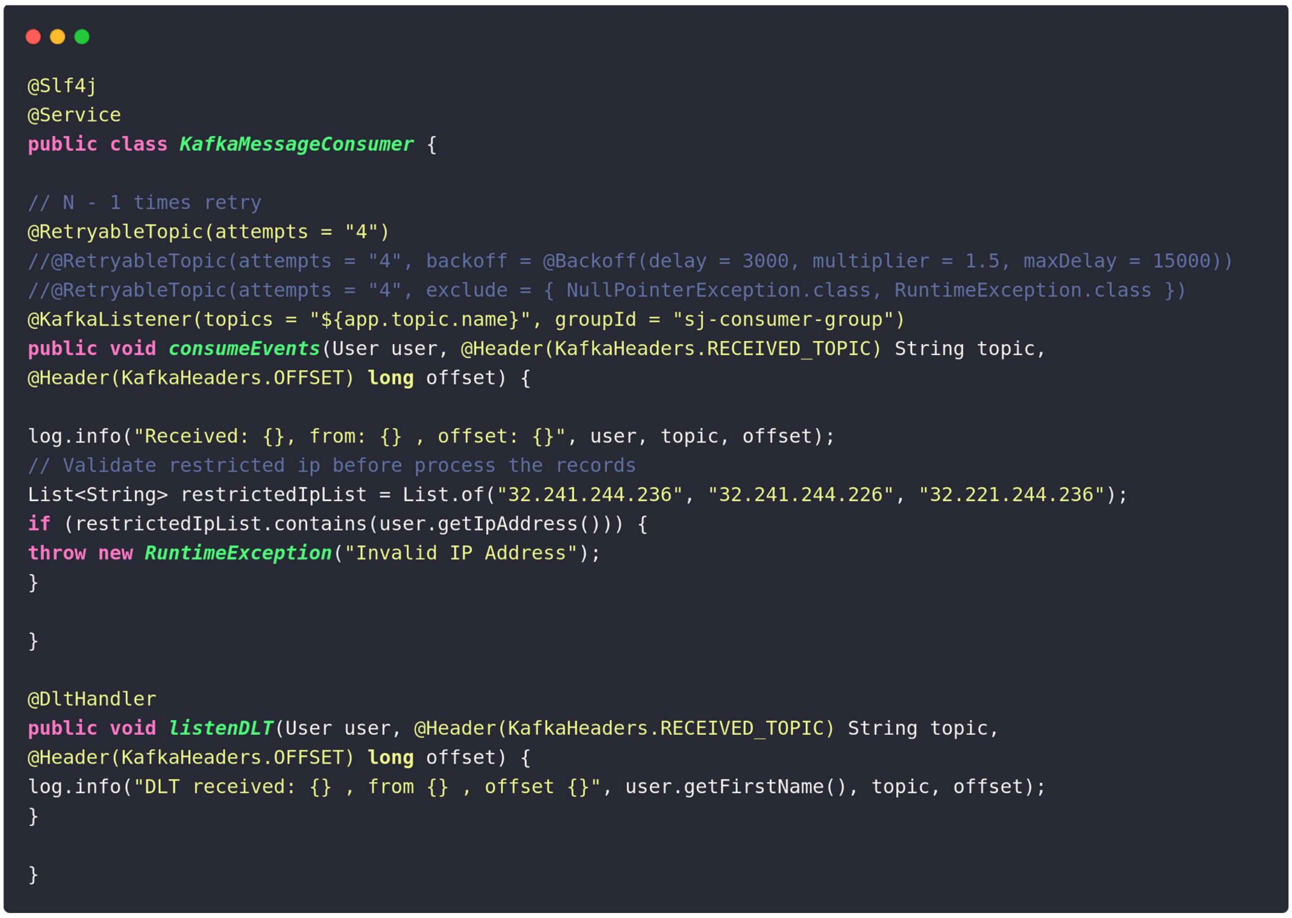


```java
@KafkaListener(topics = "customTopic", groupId = "sj-consumer-group")
public void consumeEvents(Customer message) {
logger.info("Consumer1 consume the message {}", message);
}
```

So far so good, let say you received the message but your database is down for some reason so you are unable to save the message / data to your db. Or may be you are unable to send the message from the publisher side, in these senarios we can use the @RetrayableTopic and retry again either continuasly or within some time intervals.

Here's the dedicated repo to get started : https://github.com/smit-joshi-addon/kafka-error-handling-example.git

let's look at a simple example from that.



```java
@Slf4j
@Service
public class KafkaMessageConsumer {

// N - 1 times retry
@RetryableTopic(attempts = "4")
//@RetryableTopic(attempts = "4", backoff = @Backoff(delay = 3000, multiplier = 1.5, maxDelay = 15000))
//@RetryableTopic(attempts = "4", exclude = { NullPointerException.class, RuntimeException.class })
@KafkaListener(topics = "${app.topic.name}", groupId = "sj-consumer-group")
public void consumeEvents(User user, @Header(KafkaHeaders.RECEIVED_TOPIC) String topic,
@Header(KafkaHeaders.OFFSET) long offset) {

log.info("Received: {}, from: {} , offset: {}", user, topic, offset);
// Validate restricted ip before process the records
List<String> restrictedIpList = List.of("32.241.244.236", "32.241.244.226", "32.221.244.236");
if (restrictedIpList.contains(user.getIpAddress())) {
throw new RuntimeException("Invalid IP Address");
}

}

@DltHandler
public void listenDLT(User user, @Header(KafkaHeaders.RECEIVED_TOPIC) String topic,
@Header(KafkaHeaders.OFFSET) long offset) {
log.info("DLT received: {} , from {} , offset {}", user.getFirstName(), topic, offset);
}

}
```

If the attempts in the Retrayable topic is specified to 4 then it will retry 3 times, there are diferent senarious like, if you want to retry continualy then you can just use the attempts but in cases where you would like to wait for some time before retrying then you can set the backoff, and add the delay.

Also if you are wondering what will happen to message if all the attempts are finished and still the conection is not up ?

In that kind of senarious we can create a Dead Latter Topic. (works as name suggests) stores the Messages which are unprocessed, in our case there is a tiny little method with @DltHandler on that, wich will handle these senario if all the attempts are failed then the message will go into the Dead Latter Topics.

Other repo: https://github.com/smit-joshi-addon/kafka-avro-schema-registy.git
This repo contains the avro schema registry demo.

In simple words avro registry is something that will maintain the different version of the schema changes, and will force the consumer & producers to follow the schema. It will also contain the different versions of the schema changes. For example if you have a User entity with (name,email) this is version 1, and let say some change occurs and you add another fields like (age,password,role) this is version 2, or may be you modify the existing fields from name to fullName or username, this will be the version 3.