<https://betterprogramming.pub/spring-boot-kafka-non-blocking-retries-a-hands-on-tutorial-a0c425acc3dd>

<https://docs.spring.io/spring-kafka/api/org/springframework/kafka/annotation/RetryableTopic.html>

When we process messages Kafka topics, errors can happen

Example : consumer service or other infrastructure can be down.

Default kafks failure handling behaviour retries to process the message infinitely.

Some fatal errors cannot be fixed ,we should not re-process them

@RetryableTopic

* Robust strategy to handle failed messages
* Can send failed message to DeadLetterQueue – limits the number of retries , define timeout,
* Exclude fatal exception reprocessing etc.

Blocking and Non-Blocing messaging retries.

|  |
| --- |
| @Configuration |
|  | public class KafkaConsumerConfiguration { |
|  |  |
|  | @Bean |
|  | ConcurrentKafkaListenerContainerFactory<String, String> kafkaBlockingRetryContainerFactory() { |
|  | ConcurrentKafkaListenerContainerFactory<String, String> factory = new ConcurrentKafkaListenerContainerFactory<>(); |
|  | factory.setConsumerFactory(consumerFactory); |
|  | factory.setCommonErrorHandler(new DefaultErrorHandler( |
|  | new DeadLetterPublishingRecoverer(template), new FixedBackOff(5000, 3)) |
|  | ); |
|  | return factory; |
|  | } |
|  | } |

FixedBackOff strategy with 3 attempts with recovery interval of 5 seconds

If all retry attempts are failes then message sent to Dead Letter Queue(DLT)

Incoming messages are blocked until previous once are processed.

Docker-compose.yml

version: '2.1'networks: app-tier: driver: bridgeservices: zookeeper: image: confluentinc/cp-zookeeper:latest environment: ZOOKEEPER\_CLIENT\_PORT: 2181 ZOOKEEPER\_TICK\_TIME: 2000 ports: - 22181:2181 kafka: image: confluentinc/cp-kafka:latest depends\_on: - zookeeper ports: - 29092:29092 environment: KAFKA\_BROKER\_ID: 0 KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181 KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT\_HOST://localhost:29092 KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

topic: my-topic server: port: 8090spring: kafka: bootstrap-servers: http://localhost:29092 properties: spring.deserializer.value.delegate.class: org.springframework.kafka.support.serializer.JsonDeserializer security.protocol: PLAINTEXT producer: key-serializer: org.apache.kafka.common.serialization.StringSerializer value-serializer: org.apache.kafka.common.serialization.StringSerializer consumer: group-id: my-group key-deserializer: org.apache.kafka.common.serialization.StringDeserializer value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

* The name of the topic we’ll listen to is called my-topic.
* The name of the Kafka consumer group is my-group.
* We’ve configured the KafkaProducer and KafkaConsumer properties for serialization and deserialization.
* We use the 29092 port for the Kafka bootstrap server we defined in the docker-compose.yml file.

Kafka listener

|  |
| --- |
| @Component |
|  | @Slf4j |
|  | public class MyKafkaListener { |
|  |  |
|  | @RetryableTopic( |
|  | attempts = "5", |
|  | topicSuffixingStrategy = TopicSuffixingStrategy.SUFFIX\_WITH\_INDEX\_VALUE, |
|  | backoff = @Backoff(delay = 1000, multiplier = 2.0), |
|  | exclude = {SerializationException.class, DeserializationException.class} |
|  |  |
|  | ) |
|  | @KafkaListener(id = "${spring.kafka.consumer.group-id}", topics = "${topic}") |
|  | public void handleMessage(String message, @Header(KafkaHeaders.RECEIVED\_TOPIC) String topic) { |
|  | log.info("Received message: {} from topic: {}", message, topic); |
|  | throw new RuntimeException("Test exception"); |
|  | } |
|  |  |
|  | @DltHandler |
|  | public void handleDlt(String message, @Header(KafkaHeaders.RECEIVED\_TOPIC) String topic) { |
|  | log.info("Message: {} handled by dlq topic: {}", message, topic); |
|  | } |
|  | } |

* We use the @ Component annotation to register the Bean in the Spring Boot app.
* We have a method handleMessage() where we define our Kafka listener and use the@RetryableTopic.
* When all retry attempts are exhausted, the message is forwarded to the handleDlt() method, indicated by the @DltHandler annotation. The default name of the DLT topic will be my-topic-dlt.
* Note that the throw new RuntimeException("Test exception") part is required for testing purposes.

Here are some of the @RetryableTopic's essential properties:

* The attempts property defines how many retries we want to have. In this case, we’ll have 4 retries plus 1 for the original topic.
* The app will auto-create the topics with suffixes named after the index value. For example, my-topic-retry-1. The naming strategy is defined by the topicSuffixingStrategy property.
* The backoff property instructs the app to retry the failed messages in 1 second. We have a multiplier of 2.0. This means that the second attempt will happen after 2 seconds, the third after 4 seconds, and so on.
* The exclude property lets us configure which exceptions we don’t want to retry. For example, it’s a good practice to ignore fatal exceptions, such as DeserializationException. For a complete list of unrecoverable failures, check the [documentation](https://docs.spring.io/spring-kafka/reference/html/#backoff-handlers).

The RetryableTopic provides other powerful options. If you want to check them out, visit the [documentation](https://docs.spring.io/spring-kafka/api/org/springframework/kafka/annotation/RetryableTopic.html).

Rest Controller

|  |
| --- |
| @RestController |
|  | @RequiredArgsConstructor |
|  | @Slf4j |
|  | public class ProducerController { |
|  |  |
|  | private final KafkaTemplate<String, String> kafkaTemplate; |
|  |  |
|  | @Value("${topic}") |
|  | private String topic; |
|  |  |
|  | @GetMapping("/produce/{message}") |
|  | public void produceMessage(@PathVariable("message") final String message) { |
|  | kafkaTemplate.send(topic, message); |
|  | } |
|  | } |

* The produceMessage() method will send messages to our Kafka topic.
* The kafkaTemplate is autowired by default. Of course, we could configure our own Bean if we need a custom implementation.

|  |
| --- |
| @SpringBootApplication |
|  | public class KafkaErrorHandlingApplication { |
|  |  |
|  | public static void main(String[] args) { |
|  | SpringApplication.run(KafkaErrorHandlingApplication.class, args); |
|  | } |
|  |  |
|  | } |

## Test the application

Now, it’s time for testing!

1. Start the local infrastructure by running:

docker-compose up

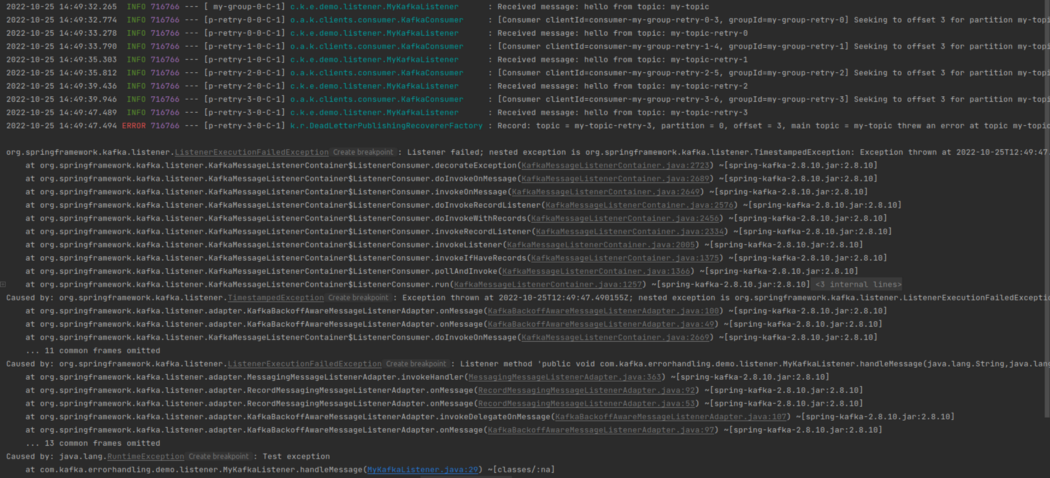
2. Run the main class - KafkaErrorHandlingApplication.java.

3. Send a test message via the controller. For example:

GET <http://localhost:8090/produce/hello>

You should see in the app’s console that the listener receives the messages. Since we throw a RuntimeException, the message will be forwarded to the retry topics and finally to the DLT.

The log looks like this:



Console log of the spring boot app

Perfect! Check the timestamp on the screenshot to see that the retry attempts happen as configured by the BackOff policy.

Also, the DLT works as expected:



Message handled by the DLT