ActiveMQ

Prerequisites:

Download and install classic ActiveMQ.

Practical Task:

I. Implement publish/subscribe interaction between two applications. Check durable vs non-durable subscription.

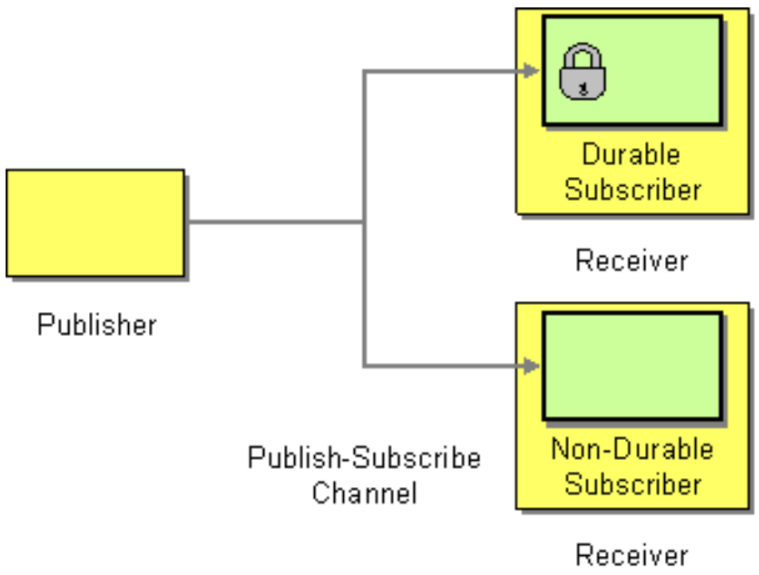


Figure 1: image info

II. Implement request-reply interaction between two applications using a temporary queue in ActiveMQ.

III. Implement subscriber scaling, i.e. create n subscribers to a topic with the same ClientID (see Virtual Topics in ActiveMQ)

References

1. ActiveMQ Documentation
2. Spring: Messaging with JMS

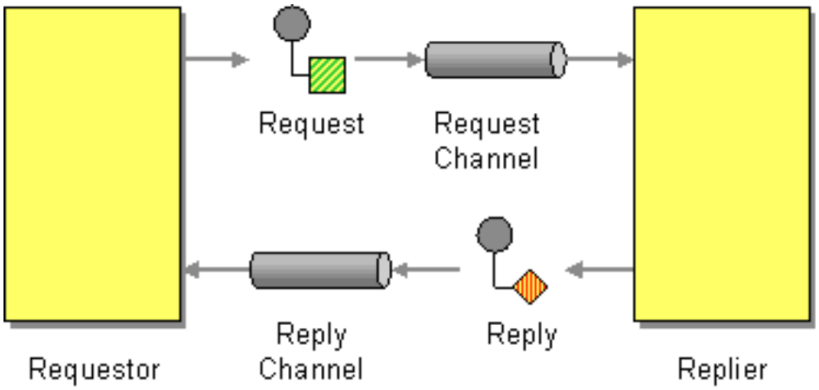


Figure 2: image info

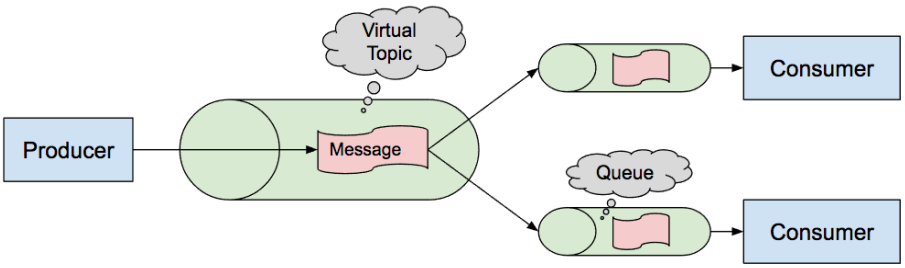


Figure 3: image info

RabbitMQ

Prerequisites:

Download and install RabbitMQ.

Practical Task:

I. Implement a Spring Boot application for sending notifications to customers about the receipt of goods based on the following architecture:

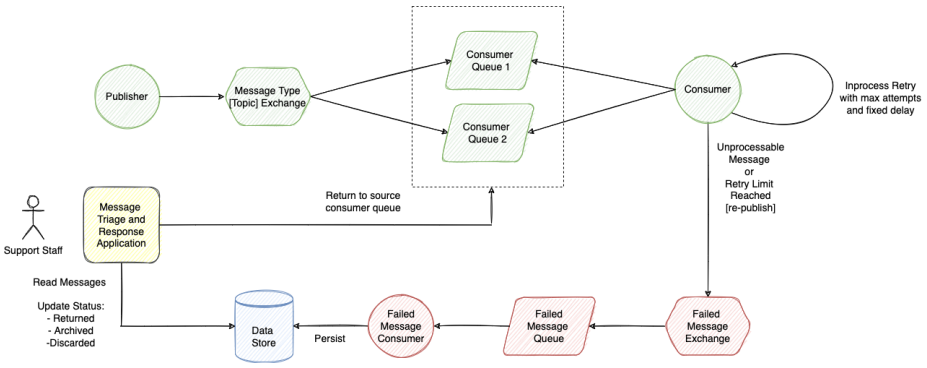


Figure 4: image info

Notes:

1. Failed Message Exchange is not configured as DLX for the source queues. Consumer is responsible to re-publish failed messages.

II. Update previous implementation and change retry mechanism from inprocess to retry exchange/queue. Retry queue should have ttl, after message expires it should be routed to the source queue.

Notes:

1. Retry exchange is not configured as DLX for the source queues. Consumer is responsible to re-publish messages for retry. If retry limit reached message should be re-published to Failed Message Exchange instead.

III. Update previous implementation, configure message ttl, max size and dlx attributes on consumer queues. Expired messages or during queue overflow messages should be sent to DLQ by broker.

Notes:

1. Dead Letter exchange should be specified as DLX attribute on source queues in addition to message TTL and max length attributes.

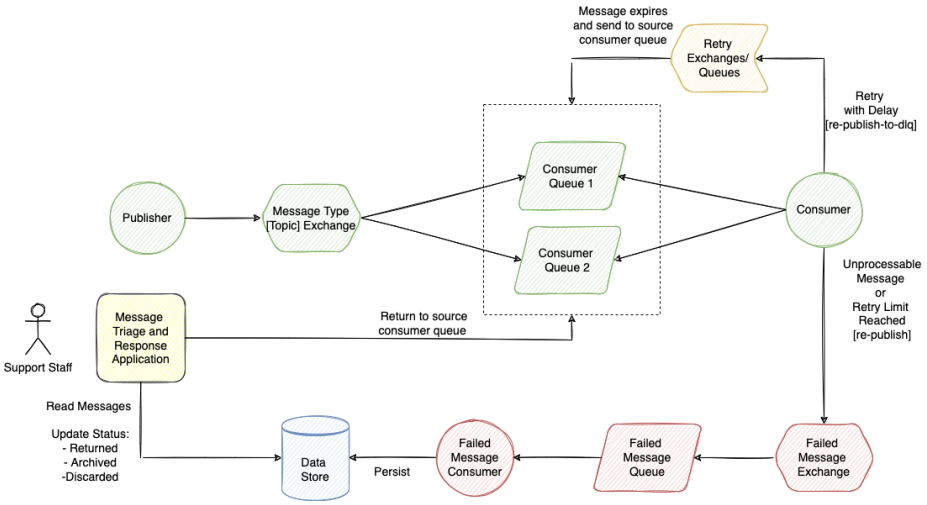


Figure 5: image info

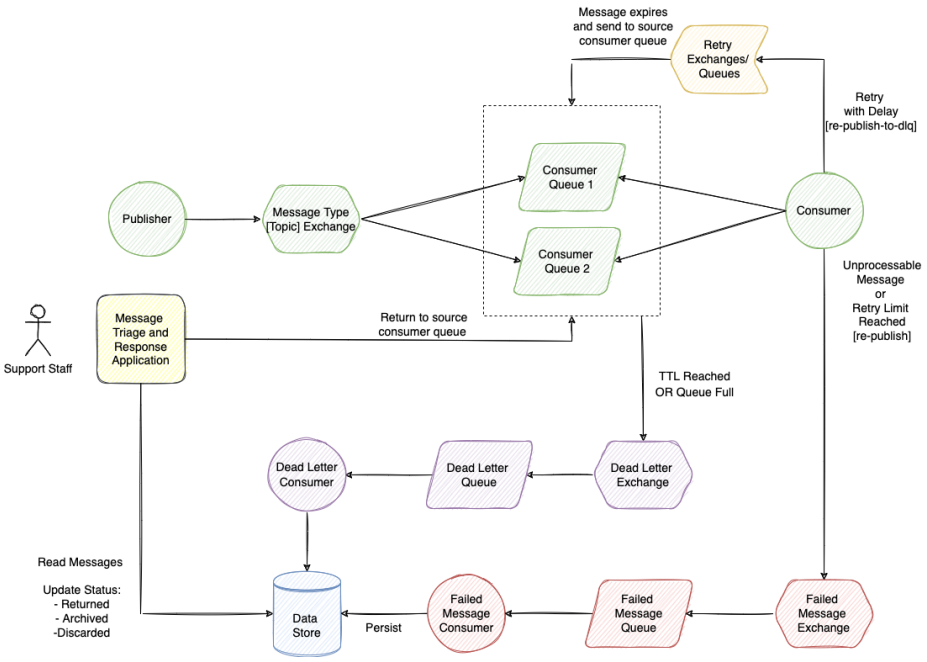
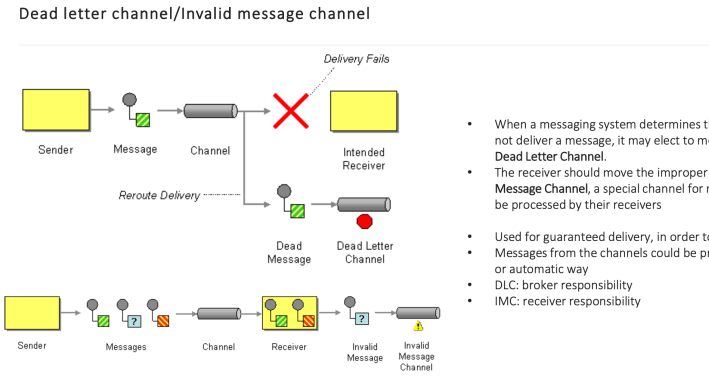


Figure 6: image info

Tips



1. Dead letter channel/Invalid message channel

References

1. RabbitMQ Documentation
2. Spring. Messaging with RabbitMQ
3. Spring Cloud Stream, RabbitMQ Binder
4. Learning RabbitMQ

Apache Kafka

Prerequisites:

Configure a Kafka cluster using Docker with the following parameters: * Number of brokers - 3 * Number of partitions - 3 * Replication factor - 2 * observe the Kafka broker logs to see how leaders/replicas for every partition are assigned

Tips

- if you're working on a machine with 16 Gb of RAM or less, you might need to fall back to just 2 brokers
- an example of a Docker Compose for a 2-node cluster based on the official Confluent Kafka image, can be found here

Practical Task:

I. Implement a pair of `"at least once"` producer and `"at most once"` consumer.

II. Implement another pair of producer and consumer with exactly-once delivery (use the Transactional API)

III. Implement a taxi application using Spring Boot. The application should consist of three components:

1. REST service fo sending taxi coordinates and car ID.
2. Kafka broker.
3. Three consumers to calculate the distance traveled by a car.

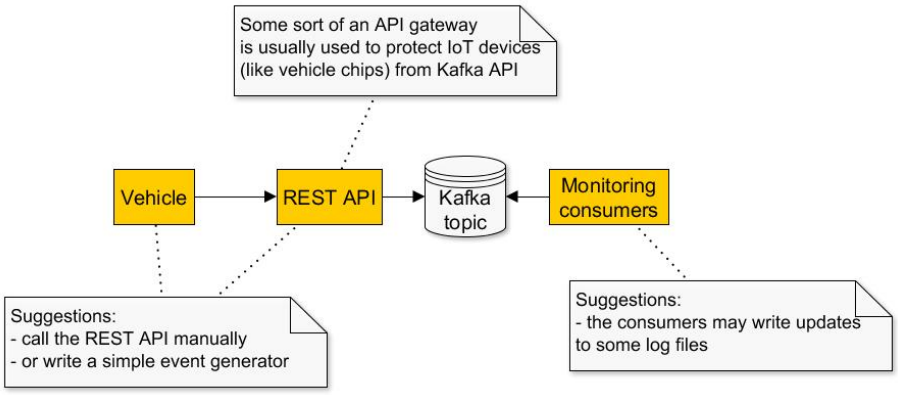


Figure 7: image

Important

- Messages from every vehicle must be processed `sequentially`!

Tips

- the first two subtasks may be done as integration tests (for example, using the Embedded Kafka from Spring Boot)

References

1. Kafka Introduction
2. Kafka Quickstart Guide
3. Spring Kafka Introduction
4. Learn Apache Kafka for Beginners