Topics

In JMS a Topic implements *publish and subscribe* semantics. When you publish a message it goes to all the subscribers who are interested - so zero to many subscribers will receive a copy of the message. Only subscribers who had an active subscription at the time the broker receives the message will get a copy of the message.

Queues

A JMS Queue implements *load balancer* semantics. A single message will be received by exactly one consumer. If there are no consumers available at the time the message is sent it will be kept until a consumer is available that can process the message. If a consumer receives a message and does not acknowledge it before closing then the message will be redelivered to another consumer. A queue can have many consumers with messages *load balanced* across the available consumers.

**Queues**

Pros

* Simple messaging pattern with a transparent communication flow
* Messages can be recovered by putting them back on the queue

Cons

* Only one consumer can get the message
* Implies a coupling between producer and consumer as it’s an one-to-one relation

**Topics**

Pros

* Multiple consumers can get a message
* Decoupling between producer and consumers (publish-and-subscribe pattern)

Cons

* More complicated communication flow

@EnableJms is a framework signal to start processing listeners and it has to be explicit because the framework has no way to know that you want to use JMS.

Spring Boot, on the other hand, can take default decisions for you based on the context. If you have the necessary bits to create a ConnectionFactory it will do so. Down the road, if we detect that a ConnectionFactory is available, we'll automatically enable the processing of JMS listeners.

To apply security to activemq broker we have to add below plugin .

