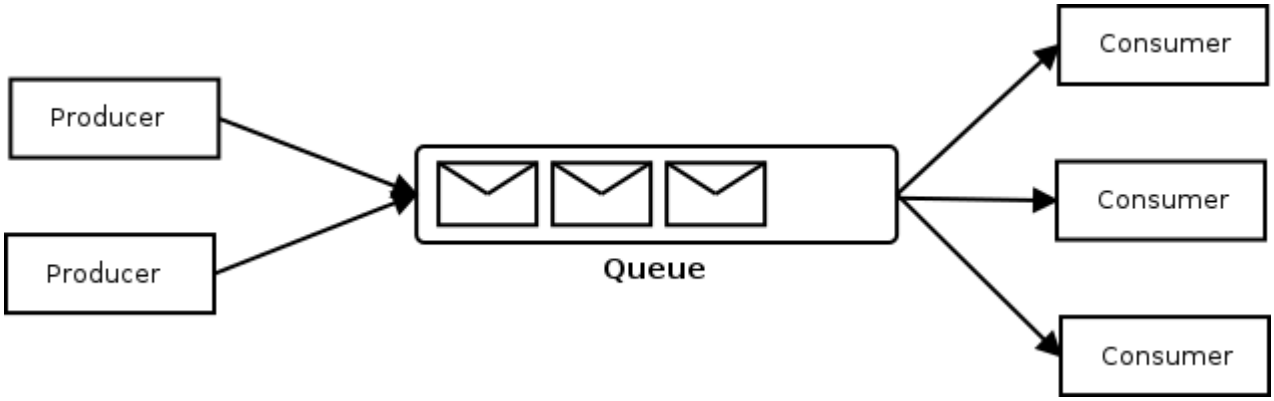
Clustering

Clustering is a large topic and often means different things to different people. We'll try to list the various aspects of clustering and how they relate to ActiveMQ

Queue consumer clusters

ActiveMQ supports reliable high performance load balancing of messages on a queue across consumers. In enterprise integration, this scenario is known as the competing consumers pattern. The following figure illustrates the concept:



This solution receives the messages sent by the producers, enqueues them and distributes them between all the registered consumers. This has a number of benefits:

- The load is distributed in a very dynamic fashion. Additional consumers could be provisioned and attached to the queue in high load periods, without modifying any configuration in the queue, as the new consumer would behave as just another competing consumer.
- Better availability than systems using a load balancer. Load balancers usually rely on a monitorization system to find out which real-servers are unavailable. With competing consumers, a failed consumer won't be competing for messages and therefore messages won't be delivered to it even without monitorization.
- High reliability, if a consumer fails, any unacknowledged messages are redelivered to other consumers on the queue.

On the downside, this pattern might not be ideal in systems where the order processing is required. To mitigate this problem while maintaining the benefits, the competing consumers pattern should be used in conjunction with other ActiveMQ features like the exclusive consumers and the message groups as stated in the ActiveMQ's FAQ.

Broker clusters

The most common mental model of clustering in a JMS context is that there is a collection of JMS brokers and a JMS client will connect to one of them; then if the JMS broker goes down, it will auto-reconnect to another broker.

We implement this using the **failover://** protocol in the JMS client. See the Failover Transport Reference page for details of how to configure the failover protocol. *Note:* The reliable:// protocol in ActiveMQ 3.x has now been changed to the failover:// protocol

If we just run multiple brokers on a network and tell the clients about them using either static discovery or dynamic discovery, then clients can easily failover from one broker to another. However, stand alone brokers don't know about consumers on other brokers; so if there are no consumers on a certain broker, messages could just pile up without being consumed. We have an outstanding feature request to tackle this on the client side - but currently the solution to this problem is to create a Network of brokers to store and forward messages between brokers.

Discovery of brokers

We support auto-discovery of brokers using static discovery or dynamic discovery, so clients can automatically detect and connect to a broker out of a logical group of brokers as well for brokers to discover and connect to other brokers to form large networks.

Networks of brokers

If you are using client/server or hub/spoke style topology and you have many clients and many brokers, there is a chance that one broker has producers but no consumers, so that messages pile up without being processed. To avoid this, ActiveMQ supports a Networks of Brokers which provides *store and forward* to move messages from brokers with producers to brokers with consumers which allows us to support distributed queues and topics across a network of brokers.

This allows a client to connect to any broker - and fail over to another broker if there is a failure - providing a cluster of brokers from the clients perspective.

Networks of brokers also allows us to scale up to a massive number of clients in a network as we can run as many brokers as we need.

You can think of this as a cluster of clients connecting with a cluster of brokers with auto-failover and discovery, making a simple and easy to use messaging fabric.

Master Slave

The problem with running lots of stand alone brokers or brokers in a network is that messages are owned by a single physical broker at any point in time. If that broker goes down, you have to wait for it to be restarted before the message can be delivered. (If you are using non-persistent messaging and a broker goes down you generally lose your message).

The idea behind MasterSlave is that messages are replicated to a slave broker so that even if you have a catastrophic hardware failure of the master's machine, file system or data centre, you get immediate failover to the slave with no message loss.

Replicated Message Stores

An alternative to MasterSlave is to have some way to replicate the message store; so for the disk files to be shared in some way. For example using a SAN or shared network drive you can share the files of a broker so that if it fails another broker can take over straight away.

So by supporting a Replicated Message Store you can reduce the risk of message loss to provide either a HA backup or a full DR solution capable of surviving a data centre failure.