RabbitMQ Clustering

Every node in the cluster share their own state, so whatever a node does, the other nodes in the cluster are aware of.

This introduces a synchronization management penalty with really big clusters

It is common to see specialized nodes in clusters

— Nodes for processing the web requests
— Nodes for handling specific heavy queues

Cluster state

The cluster state includes:

— Definition of exchanges
— Queues
— Bindings
— Virtual hosts
— Users
— Policies

Node types

Nodes can be of two types:

— RAM nodes, keep the cluster state in RAM
— Disk nodes, keep the cluster state in RAM **and** disk

Node types **are not** related to the persistence of the message or queue

If a node or cluster crashes, disk nodes will be used to reconstruct the state of the cluster or node [1]

[1] Upon rejoining, a node will notify about their queues and state to the cluster

We should always have at least one disk node in a cluster

but *too many* disk nodes could cause problems when nodes agree to the shared state during multiple node failures

The stats node

With the management plugin enabled, a unique *stats node* is created to handle and keep the stats of the cluster and nodes

There is only **one** stats node in the cluster, in *large* clusters is recommended to have one node as the stats and management node and have at least *one* disk cluster to provide failover.

Nodes and queues

A queue is created in the node receiving the request

RabbitMQ does not distribute (by default) resources to queues or publishing, this could have heavy implications with large queues and exchanges

Nodes and publishing

If a node receives a message but the exchange needs to deliver to a queue in another node, this implies coordination overhead for heavy operations (transactions, alternate exchanges, HA queues)

If a published message is recieved by a node but the queue is hosted in another, the message needs to be transfered between nodes[2]

[2] This doesn't apply to HA queues

Federation

In RabbitMQ we can federate exchanges and queues

— Federate exchanges allow to replicate messages published in one exchange in another exchange (same name) in another node or even cluster

— Federate queues allow a queue to be consumed by downstream queues

Federate exchanges are mostly used in multiple zone clusters (high latency)

Federate queues send messages from a queue to downstream queues in different servers or nodes

If they are part of HA queues, this will replicate such messages in those queues

The downstream queues receive messages *only if* they have consumers for messages

By default the Federate Plugin federates both exchanges and queues

HA and Quorum queues

Federate queues work between clusters

HA queues are replicated queues, instead of having local queues, the queue is replicated in every node in the cluster

Quorum queues are like replicated queues, but they are designed for avoiding data loss

Diference between Quorum and HA queues

	HA	Quorum
Exclusivity	yes	no
Non-durable	yes	no
Per message persistence	optional[3]	always
Message TTL	yes	no
Message priority	yes	no

[3] It depends on message and queue options

Lazy queues

They were designed to solve one problem:

Very long queues with low activity

In lazy queues messages are stored in disk as early as possible and set in RAM only when a queue consumer is active

Policies

With policies we can control default settings for new and existing queues and exchanges[4]

You can create policies from the command line or from the UI

A queue or exchange *can respond only to one policy at a time*

[4] Or both with the same policy