Chapter 4: Distributed RabbitMQ

In this chapter we will look the ways that RabbitMQ achieves Scalability and High Availability. This chapter is leaning more to system and DevOps engineers work scope, but also it is good idea to be read by the developers.

We will start by taking an overview of all mechanisms and give brief definition for each of them and then we will dive deeper into every topic.

Overview

AMQP 0-9-1, AMQP 1.0 and the other messaging protocols supported by RabbitMQ via plug-ins (e.g. STOMP), are (of course) inherently distributed: applications almost always connect to RabbitMQ on a remote host.

Often it is necessary or desirable to make the RabbitMQ broker itself distributed. There are three ways in which to accomplish that: with clustering, with Federation, and using the Shovel plugin. This page provides an overview of each approach.

Note that all three approaches are not mutually exclusive and can be combined: clusters can be connected together with Federation or Shovel, or both.

Clustering

Clustering connects multiple machines together to **form a cluster**. Inter-node communication is performed transparently to clients. The design of clustering assumes that network connections are reasonably reliable and provides a LAN-like latency.

All nodes in the cluster must run compatible versions of RabbitMQ and **Erlang**.

Nodes authenticate to each other using **a pre-shared secret** typically installed by deployment automation tools.

Virtual hosts, exchanges, users, and permissions are **automatically replicated** across all nodes in a cluster. Queues may be located on a single node, or **mirrored across multiple nodes**.

A client connecting to any node in a cluster can **use all non-exclusive queues in the cluster**, even if they are not located on that node.

Clustering nodes can help improve availability, data safety of queue contents and sustain more concurrent client connections. The **Clustering** and **Queue Mirroring** guides provide more details on these topics.

Federation

Federation allows an exchange or queue on one broker to receive messages published to an exchange or queue on another (the brokers may be individual machines, or clusters). Communication is via AMQP (with optional SSL), so for two exchanges or queues to federate they must be granted appropriate users and permissions.

Federated exchanges are connected with one way point-to-point links. By default, messages will only be forwarded over a federation link once, but this can be increased to allow for more complex routing topologies. Some messages may not be forwarded over the link; if a message would not be routed to a queue after reaching the federated exchange, it will not be forwarded in the first place.

Federated queues are similarly connected with one way point-to-point links. Messages will be moved between federated queues an arbitrary number of times to follow the consumers.

Typically you would use federation to link brokers across the internet for pub/sub messaging and work queueing.

Shovels

Connecting brokers with **the Shovel plugin** is conceptually similar to connecting them with Federation. However, the plugin works at a lower level.

Whereas federation aims to provide opinionated distribution of exchanges and queues, the shovel simply consumes messages from a queue on one broker, and forwards them to an exchange on another.

Typically you would use the shovel to link brokers across the internet when you need more control than federation provides.

Dynamic shovels can also be useful for moving messages around in an ad-hoc manner on a single broker.

Summary

Federation and/or Shovel	Clustering

Federation and/or Shovel	**Clustering**
Brokers are logically separate and may have different owners.	A cluster forms a single logical broker.
Brokers can run different (and incompatible in certain ways) versions of RabbitMQ and Erlang.	Nodes must run compatible versions RabbitMQ and Erlang.
Brokers can be connected via unreliable WAN links. Communication is via AMQP 0-9-1 (optionally secured by **TLS**, requiring appropriate users and permissions to be set up.	Brokers must be connected via reasonably reliable LAN links. Nodes will authenticate to each other using a shared secret and optionally **use TLS-enabled links**.
Brokers can be connected in whatever topology you arrange. Links can be one- or two-way.	All nodes connect to all other nodes in both directions.
Emphasizes Availability and Partition Tolerance (AP) from the **CAP theorem**.	Emphasizes Consistency and Partition Tolerance (CP) from the **CAP theorem**.
Some exchanges in a broker may be federated while some may be local.	Clustering is all-or-nothing.
A client connecting to any broker can only use non-exclusive queues in that broker.	A client connecting to any node can use non-exclusive queues on all nodes.

4.1 Clustering

In this section we will dive deeper in the RabbitMQ clustering mechanism.