Production Checklist

Introduction

Data services such as RabbitMQ often have many tunable parameters. Some configurations make a lot of sense for development but are not really suitable for production. No single configuration fits every use case. It is, therefore, important to assess your configuration before going into production. This guide aims to help with that.

Virtual Hosts, Users, Permissions

Virtual Hosts

In a single-tenant environment, for example, when your RabbitMQ cluster is dedicated to power a single system in production, using default virtual host (/) is perfectly fine.

In multi-tenant environments, use a separate vhost for each tenant/environment, e.g. project1_development, project1_production, project2_development, project2_production, and so on.

Users

For production environments, delete the default user (guest). Default user only can connect from localhost by default, because it has well-known credentials. Instead of enabling remote connections, consider using a separate user with administrative permissions and a generated password.

It is recommended to use a separate user per application. For example, if you have a mobile app, a Web app, and a data aggregation system, you'd have 3 separate users. This makes a number of things easier:

- Correlating client connections with applications
- Using fine-grained permissions
- Credentials roll-over (e.g. periodically or in case of a breach)
  In case there are many instances of the same application, there's a trade-off between better security (having a set of credentials per instance) and convenience of provisioning (sharing a set of credentials between some or all instances). For IoT applications that involve many clients performing the same or similar function and having fixed IP addresses, it may make sense to authenticate using x509 certificates or source IP addresse ranges.

Resource Limits and Monitoring

RabbitMQ uses Resource-driven alarms to throttle publishers when consumers do not keep up. It is important to evaluate resource limit configurations before going into production.

Memory

By default, RabbitMQ will not accept any new messages when it detects that it's using more than 40% of the available memory (as reported by the OS): {vm_memory_high_watermark, 0.4}. This is a safe default and care should be taken when modifying this value, even when the host is a dedicated RabbitMQ node.

The OS and file system use system memory to speed up operations for all system processes. Failing to leave enough free system memory for this purpose will have an adverse effect on system performance due to OS swapping, and can even result in RabbitMQ process termination.

A few recommendations when adjusting the default vm_memory_high_watermark:

- Nodes hosting RabbitMQ should have at least 128MB of memory available at all times.
- The recommended vm_memory_high_watermark range is 0.40 to 0.66
- Values above 0.7 are not recommended. The OS and file system must be left with at least 30% of the memory, otherwise performance may degrade severely due to paging.

As with every tuning, benchmarking and measuring are required to find the best setting for your environment and workload.

Learn more about RabbitMQ & system memory in a separate guide.

Disk Space

The current 50MB disk_free_limit default works very well for development and tutorials. Production deployments require a much greater safety margin. Insufficient disk space will lead to node failures and may result in data loss as all disk writes will fail.

Why is the default 50MB then? Development environments sometimes use really small partitions to host /var/lib, for example, which means nodes go into resource alarm state right after booting. The very low default ensures that RabbitMQ works out of the box for everyone. As for production deployments, we recommend the following:

- {disk_free_limit, {mem_relative, 1.0}} is the minimum recommended value and it translates to the total amount of memory available. For example, on a host dedicated to RabbitMQ with 4GB of system memory, if available disk space drops below 4GB, all publishers will be blocked and no new messages will be accepted. Queues will need to be drained, normally by consumers, before publishing will be allowed to resume.

- {disk_free_limit, {mem_relative, 1.5}} is a safer production value. On a RabbitMQ node with 4GB of memory, if available disk space drops below 6GB, all new messages will be blocked until the disk alarm clears. If RabbitMQ needs to flush to disk 4GB worth of data, as can sometimes be the case during shutdown, there will be sufficient disk space available for RabbitMQ to start again. In this specific example, RabbitMQ will start and immediately block all publishers since 2GB is well under the required 6GB.
- {disk_free_limit, {mem_relative, 2.0}} is the most conservative production value, we cannot think of any reason to use anything higher.If you want full confidence in RabbitMQ having all the disk space that it needs, at all times, this is the value to use.

Open File Handles Limit

Operating systems limit maximum number of concurrently open file handles, which includes network sockets. Make sure that you have limits set high enough to allow for expected number of concurrent connections and queues.

Make sure your environment allows for at least 50K open file descriptors for effective RabbitMQ user, including in development environments.

As a rule of thumb, multiple the 95th percentile number of concurrent connections by 2 and add total number of queues to calculate recommended open file handle limit. Values as high as 500K are not inadequate and won't consume a lot of hardware resources, and therefore are recommended for production setups. See Networking guide for more information.

Monitoring

Monitoring several aspects of the system, from infrastructure and kernel metrics to RabbitMQ to application-level metrics is highly recommended. While monitoring requires an upfront investment in terms of time, it is very effective at catching issues early (or at all).

Log Collection

It is highly recommended that logs of all RabbitMQ nodes and applications (when possible) are collected and aggregated. Logs can be crucially important in investigating unusual system behaviour.

Security Considerations

Users and Permissions

See the section on vhosts, users, and credentials above.

Erlang Cookie

On Linux and BSD systems, it is necessary to restrict Erlang cookie access only to the users that will run RabbitMQ and tools such as rabbitmqctl.

TLS

We recommend using TLS connections when possible, at least to encrypt traffic. Peer verification (authentication) is also recommended. Development and QA environments can use self-signed TLS certificates. Self-signed certificates can be appropriate in production environments when RabbitMQ and all applications run on a trusted network or isolated using technologies such as VMware NSX.

While RabbitMQ tries to offer a secure TLS configuration by default (e.g. SSLv3 is disabled), we recommend evaluating what TLS versions and cipher suites are enabled. Please see our TLS guide for more information.

Note that TLS can have significant impact on overall system throughput, including CPU usage of both RabbitMQ and applications that use it.

Networking Configuration

Production environments may require network configuration tuning. Please refer to the Networking Guide for details.

Automatic Connection Recovery

Some client libraries, for example, Java, .NET, and Ruby ones, support automatic connection recovery after network failures. If the client used provides this feature, it is recommended to use it instead of developing your own recovery mechanism.

Clustering Considerations

Cluster Size

When determining cluster size, it is important to take several factors into consideration:

- Expected throughput
- Expected replication (number of mirrors)
- Data locality

Since clients can connect to any node, RabbitMQ may need to perform inter-cluster routing of messages and internal operations. Try making consumers and producers connect to the same node, if possible: this will reduce inter-node traffic. Equally helpful would be making consumers connect to the node that currently hosts queue master (can be inferred using HTTP

API). When data locality is taken into consideration, total cluster throughput can reach non-trivial volumes.

For most environments, mirroring to more than half of cluster nodes is sufficient. It is recommended to use clusters with an odd number of nodes (3, 5, and so on).

Partition Handling Strategy

It is important to pick a partition handling strategy before going into production. When in doubt, use the autoheal strategy.

Node Time Synchronization

A RabbitMQ cluster will typically function well without clocks of participating servers being synchronized. However some plugins, such as the Management UI, make use of local timestamps for metrics processing and may display incorrect statistics when the current time of nodes drift apart. It is therefore recommended that servers use NTP or similar to ensure clocks remain in sync.