rabbitmq

DOCKER OFFICIAL IMAGE

RabbitMQ is an open source multi-protocol messaging broker.

1B+

Linux ARM 64 PowerPC 64 LE riscv64 IBM Z 386 x86-64 ARM Docker Official Image

Copy and paste to pull this image

```
docker pull rabbitmq
```

View Available Tags

Description Tags

Quick reference

- **Maintained by:**
  the Docker Community
- **Where to get help:**
  the Docker Community Forums, the Docker Community Slack, or Stack Overflow

Supported tags and respective `Dockerfile` links

- `3.10.7` , `3.10` , `3` , `latest`
- `3.10.7-management` , `3.10-management` , `3-management` , `management`
- `3.10.7-alpine` , `3.10-alpine` , `3-alpine` , `alpine`
- `3.10.7-management-alpine` , `3.10-management-alpine` , `3-management-alpine` , `management-alpine`
- `3.9.22` , `3.9`
- `3.9.22-management` , `3.9-management`
- `3.9.22-alpine` , `3.9-alpine`
- `3.9.22-management-alpine` , `3.9-management-alpine`

Quick reference (cont.)

- **Where to file issues:**
  https://github.com/docker-library/rabbitmq/issues
- **Supported architectures:** (more info)
  `amd64` , `arm32v6` , `arm32v7` , `arm64v8` , `i386` , `ppc64le` , `riscv64` , `s390x`
- **Published image artifact details:**
  repo-info repo's `repos/rabbitmq/` directory (history)
  (image metadata, transfer size, etc)
- **Image updates:**
  official-images repo's `library/rabbitmq` label
  official-images repo's `library/rabbitmq` file (history)
- **Source of this description:**
  docs repo's `rabbitmq/` directory (history)

What is RabbitMQ?

RabbitMQ is open source message broker software (sometimes called message-oriented middleware) that implements the Advanced Message Queuing Protocol (AMQP). The RabbitMQ server is written in the Erlang programming language and is built on the Open Telecom Platform framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages.

> wikipedia.org/wiki/RabbitMQ

How to use this image

Running the daemon

One of the important things to note about RabbitMQ is that it stores data based on what it calls the "Node Name", which defaults to the hostname. What this means for usage in Docker is that we should specify `-h` / `--hostname` explicitly for each daemon so that we don't get a random hostname and can keep track of our data:

```
$ docker run -d --hostname my-rabbit --name some-rabbit rabbitmq:3
```

This will start a RabbitMQ container listening on the default port of 5672. If you give that a minute, then do `docker logs some-rabbit`, you'll see in the output a block similar to:

```
=INFO REPORT==== 6-Jul-2015::20:47:02 ===
node           : rabbit@my-rabbit
home dir       : /var/lib/rabbitmq
config file(s) : /etc/rabbitmq/rabbitmq.config
cookie hash    : UoNOcDhfxW9uoZ92wh6BjA==
log            : tty
sasl log       : tty
database dir   : /var/lib/rabbitmq/mnesia/rabbit@my-rabbit
```

Note the `database dir` there, especially that it has my "Node Name" appended to the end for the file storage. This image makes all of `/var/lib/rabbitmq` a volume by default.

Environment Variables

For a list of environment variables supported by RabbitMQ itself, see the Environment Variables section of rabbitmq.com/configure

WARNING: As of RabbitMQ 3.9, all of the docker-specific variables listed below are deprecated and no longer used. Please use a configuration file instead; visit rabbitmq.com/configure to learn more about the configuration file. For a starting point, the 3.8 images will print out the config file it generated from supplied environment variables.

```
# Unavailable in 3.9 and up
RABBITMQ_DEFAULT_PASS_FILE
RABBITMQ_DEFAULT_USER_FILE
RABBITMQ_MANAGEMENT_SSL_CACERTFILE
RABBITMQ_MANAGEMENT_SSL_CERTFILE
RABBITMQ_MANAGEMENT_SSL_DEPTH
RABBITMQ_MANAGEMENT_SSL_FAIL_IF_NO_PEER_CERT
RABBITMQ_MANAGEMENT_SSL_KEYFILE
RABBITMQ_MANAGEMENT_SSL_VERIFY
RABBITMQ_SSL_CACERTFILE
RABBITMQ_SSL_CERTFILE
RABBITMQ_SSL_DEPTH
RABBITMQ_SSL_FAIL_IF_NO_PEER_CERT
RABBITMQ_SSL_KEYFILE
RABBITMQ_SSL_VERIFY
RABBITMQ_VM_MEMORY_HIGH_WATERMARK
```

Setting default user and password

If you wish to change the default username and password of `guest` / `guest`, you can do so with the `RABBITMQ_DEFAULT_USER` and `RABBITMQ_DEFAULT_PASS` environmental variables. These variables were available previously in the docker-specific entrypoint shell script but are now available in RabbitMQ directly.

```
$ docker run -d --hostname my-rabbit --name some-rabbit -e RABBITMQ_DEFAULT_USER=user -e RABBITMQ_DEFAULT_PASS
```

You can then go to `http://localhost:8080` or `http://host-ip:8080` in a browser and use `user` / `password` to gain access to the management console

Setting default vhost

If you wish to change the default vhost, you can do so with the `RABBITMQ_DEFAULT_VHOST` environmental variables:

```
$ docker run -d --hostname my-rabbit --name some-rabbit -e RABBITMQ_DEFAULT_VHOST=my_vhost rabbitmq:3-manageme
```

Memory Limits

RabbitMQ contains functionality which explicitly tracks and manages memory usage, and thus needs to be made aware of cgroup-imposed limits (e.g. `docker run --memory=..`).

The upstream configuration setting for this is `vm_memory_high_watermark` in `rabbitmq.conf`, and it is described under "Memory Alarms" in the documentation. If you set a relative limit via `vm_memory_high_watermark.relative`, then RabbitMQ will calculate its limits based on the host's total memory and not the limit set by the contianer runtime.

Erlang Cookie

See the RabbitMQ "Clustering Guide" for more information about cookies and why they're necessary. For setting a consistent cookie (especially useful for clustering but also for remote/cross-container administration via `rabbitmqctl`), provide a cookie file (default location of `/var/lib/rabbitmq/.erlang.cookie`).

For example, you can provide the cookie via a file (such as with Docker Secrets):

```
docker service create ... --secret source=my-erlang-cookie,target=/var/lib/rabbitmq/.erlang.cookie ... rabbitm
```

(Note that it will likely also be necessary to specify `uid=XXX,gid=XXX,mode=0600` in order for Erlang in the container to be able to read the cookie file properly. See Docker's `--secret` documentation for more details.)

Management Plugin

There is a second set of tags provided with the management plugin installed and enabled by default, which is available on the standard management port of 15672, with the default username and password of `guest` / `guest`:

```
$ docker run -d --hostname my-rabbit --name some-rabbit rabbitmq:3-management
```

You can access it by visiting `http://container-ip:15672` in a browser or, if you need access outside the host, on port 8080:

```
$ docker run -d --hostname my-rabbit --name some-rabbit -p 8080:15672 rabbitmq:3-management
```

You can then go to `http://localhost:8080` or `http://host-ip:8080` in a browser.

Enabling Plugins

Creating a Dockerfile will have them enabled at runtime. To see the full list of plugins present on the image `rabbitmq-plugins list`

```
FROM rabbitmq:3.8-management
RUN rabbitmq-plugins enable --offline rabbitmq_mqtt rabbitmq_federation_management rabbitmq_stomp
```

You can also mount a file at `/etc/rabbitmq/enabled_plugins` with contents as an erlang list of atoms ending with a period.

Example `enabled_plugins`

```
[rabbitmq_federation_management,rabbitmq_management,rabbitmq_mqtt,rabbitmq_stomp].
```

Additional Configuration

If configuration is required, it is recommended to supply an appropriate `/etc/rabbitmq/rabbitmq.conf` file (see the "Configuration File(s)" section of the RabbitMQ documentation for more details), for example via bind-mount, Docker Configs, or a short `Dockerfile` with a `COPY` instruction.

Alternatively, it is possible to use the `RABBITMQ_SERVER_ADDITIONAL_ERL_ARGS` environment variable, whose syntax is described in section 7.8 ("Configuring an Application") of the Erlang OTP Design Principles User's Guide (the appropriate value for `-ApplName` is `-rabbit`), this method requires a slightly different reproduction of its equivalent entry in `rabbitmq.conf`. For example, configuring `channel_max` would look something like `-e RABBITMQ_SERVER_ADDITIONAL_ERL_ARGS="-rabbit channel_max 4007"`. Where the space between the variable `channel_max` and its value `4007` correctly becomes a comma when translated in the environment.

Health/Liveness/Readiness Checking

See the "Official Images" FAQ and the discussion on docker-library/rabbitmq#174 (especially the large comment by Michael Klishin from RabbitMQ upstream) for a detailed explanation of why this image does not come with a default `HEALTHCHECK` defined, and for suggestions for implementing your own health/liveness/readiness checks.

Image Variants

The `rabbitmq` images come in many flavors, each designed for a specific use case.

`rabbitmq:<version>`

This is the defacto image. If you are unsure about what your needs are, you probably want to use this one. It is designed to be used both as a throw away container (mount your source code and start the container to start your app), as well as the base to build other images off of.

`rabbitmq:<version>-alpine`

This image is based on the popular Alpine Linux project, available in the `alpine` official image. Alpine Linux is much smaller than most distribution base images (~5MB), and thus leads to much slimmer images in general.

This variant is useful when final image size being as small as possible is your primary concern. The main caveat to note is that it does use musl libc instead of glibc and friends, so software will often run into issues depending on the depth of their libc requirements/assumptions. See this Hacker News comment thread for more discussion of the issues that might arise and some pro/con comparisons of using Alpine-based images.

To minimize image size, it's uncommon for additional related tools (such as `git` or `bash`) to be included in Alpine-based images. Using this image as a base, add the things you need in your own Dockerfile (see the `alpine` image description for examples of how to install packages if you are unfamiliar).

License

View license information for the software contained in this image.

As with all Docker images, these likely also contain other software which may be under other licenses (such as Bash, etc from the base distribution, along with any direct or indirect dependencies of the primary software being contained).

Some additional license information which was able to be auto-detected might be found in the `repo-info` repository's `rabbitmq/` directory.

As for any pre-built image usage, it is the image user's responsibility to ensure that any use of this image complies with any relevant licenses for all software contained within.

Kubernetes