(/)

OFFICIAL REPOSITORY

jenkins (/r/_/jenkins/) ☆

Last pushed: 4 months ago

Repo Info (/_/jenkins/)

Short Description

Official Jenkins Docker image

Full Description

DEPRECATION NOTICE

This image has been deprecated in favor of the `jenkins/jenkins:lts` (https://hub.docker.com/r/jenkins/jenkins) image provided and maintained by Jenkins Community (https://jenkins.io/) as part of project's release process. The images found here will receive no further updates after LTS 2.60.x. Please adjust your usage accordingly.

Supported tags and respective `Dockerfile` links

- `latest`, `2.60.3` (*Dockerfile*) (https://github.com/jenkinsci/jenkins-ci.org-docker/blob/587b2856cd225bb152c4abeeaaa24934c75aa460/Dockerfile)
- `alpine`, `2.60.3-alpine` (*Dockerfile*) (https://github.com/jenkinsci/jenkins-ci.org-docker/blob/c2d6f2122fa03c437e139a317b7fe5b9547fe49e/Dockerfile)

Quick reference

- **Where to get help**:
  the Docker Community Forums (https://forums.docker.com/), the Docker Community Slack (https://blog.docker.com/2016/11/introducing-docker-community-directory-docker-community-slack/), or Stack Overflow (https://stackoverflow.com/search?tab=newest&q=docker)

- **Where to file issues**:
  https://github.com/cloudbees/jenkins-ci.org-docker/issues (https://github.com/cloudbees/jenkins-ci.org-docker/issues)

- **Maintained by**:
  the Jenkins Project (https://github.com/cloudbees/jenkins-ci.org-docker)

- **Supported architectures**: (more info (https://github.com/docker-library/official-images#architectures-other-than-amd64))
  `amd64` (https://hub.docker.com/r/amd64/jenkins/)

- **Published image artifact details**:
  repo-info repo's `repos/jenkins/` directory (https://github.com/docker-library/repo-info/blob/master/repos/jenkins) (history (https://github.com/docker-library/repo-info/commits/master/repos/jenkins))
  (image metadata, transfer size, etc)

- **Image updates**:
  official-images PRs with label `library/jenkins` (https://github.com/docker-library/official-images/pulls?q=label%3Alibrary%2Fjenkins)
  official-images repo's `library/jenkins` file (https://github.com/docker-library/official-images/blob/master/library/jenkins) (history (https://github.com/docker-library/official-images/commits/master/library/jenkins))

- **Source of this description**:
  docs repo's `jenkins/` directory (https://github.com/docker-library/docs/tree/master/jenkins) (history (https://github.com/docker-library/docs/commits/master/jenkins))

- **Supported Docker versions**:
  the latest release (https://github.com/docker/docker-ce/releases/latest) (down to 1.6 on a best-effort basis)

Jenkins

The Jenkins Continuous Integration and Delivery server.

This is a fully functional Jenkins server, based on the Long Term Support release http://jenkins.io/ (http://jenkins.io/).

For weekly releases check out `jenkinsci/jenkins` (https://hub.docker.com/r/jenkinsci/jenkins/)

How to use this image

```
docker run -p 8080:8080 -p 50000:50000 jenkins
```

This will store the workspace in /var/jenkins_home. All Jenkins data lives in there - including plugins and configuration. You will probably want to make that a persistent volume (recommended):

```
docker run -p 8080:8080 -p 50000:50000 -v /your/home:/var/jenkins_h
```

This will store the jenkins data in `/your/home` on the host. Ensure that `/your/home` is accessible by the jenkins user in container (jenkins user - uid 1000) or use `-u some_other_user` parameter with `docker run`.

You can also use a volume container:

```
docker run --name myjenkins -p 8080:8080 -p 50000:50000 -v /var/jer
```

Then myjenkins container has the volume (please do read about docker volume handling to find out more).

Backing up data

If you bind mount in a volume - you can simply back up that directory (which is jenkins_home) at any time.

This is highly recommended. Treat the jenkins_home directory as you would a database - in Docker you would generally put a database on a volume.

If your volume is inside a container - you can use `docker cp $ID:/var/jenkins_home` command to extract the data, or other options to find where the volume data is. Note that some symlinks on some OSes may be converted to copies (this can confuse jenkins with lastStableBuild links etc)

For more info check Docker docs section on Managing data in containers (https://docs.docker.com/engine/tutorials/dockervolumes/)

Setting the number of executors

You can specify and set the number of executors of your Jenkins master instance using a groovy script. By default its set to 2 executors, but you can extend the image and change it to your desired number of executors :

`executors.groovy`

```
import jenkins.model.*
Jenkins.instance.setNumExecutors(5)
```

and `Dockerfile`

```
FROM jenkins
COPY executors.groovy /usr/share/jenkins/ref/init.groovy.d/executor
```

Attaching build executors

You can run builds on the master (out of the box) but if you want to attach build slave servers: make sure you map the port: `-p 50000:50000` - which will be used when you connect a slave agent.

Passing JVM parameters

You might need to customize the JVM running Jenkins, typically to pass system properties or tweak heap memory settings. Use JAVA_OPTS environment variable for this purpose :

```
docker run --name myjenkins -p 8080:8080 -p 50000:50000 --env JAVA_
```

Configuring logging

Jenkins logging can be configured through a properties file and `java.util.logging.config.file` Java property. For example:

```
mkdir data
cat > data/log.properties <<EOF
handlers=java.util.logging.ConsoleHandler
jenkins.level=FINEST
java.util.logging.ConsoleHandler.level=FINEST
EOF
docker run --name myjenkins -p 8080:8080 -p 50000:50000 --env JAVA_
```

Passing Jenkins launcher parameters

Arguments you pass to docker running the jenkins image are passed to jenkins launcher, so you can run for example :

```
$ docker run jenkins --version
```

This will dump Jenkins version, just like when you run jenkins as an executable war.

You also can define jenkins arguments as `JENKINS_OPTS` . This is useful to define a set of arguments to pass to jenkins launcher as you define a derived jenkins image based on the official one with some customized settings. The following sample Dockerfile uses this option to force use of HTTPS with a certificate included in the image

```
FROM jenkins:1.565.3

COPY https.pem /var/lib/jenkins/cert
COPY https.key /var/lib/jenkins/pk
ENV JENKINS_OPTS --httpPort=-1 --httpsPort=8083 --httpsCertificate=
EXPOSE 8083
```

You can also change the default slave agent port for jenkins by defining `JENKINS_SLAVE_AGENT_PORT` in a sample Dockerfile.

```
FROM jenkins:1.565.3
ENV JENKINS_SLAVE_AGENT_PORT 50001
```

or as a parameter to docker,

```
$ docker run --name myjenkins -p 8080:8080 -p 50001:50001 --env JEN
```

Installing more tools

You can run your container as root - and install via apt-get, install as part of build steps via jenkins tool installers, or you can create your own Dockerfile to customise, for example:

```
FROM jenkins
# if we want to install via apt
USER root
RUN apt-get update && apt-get install -y ruby make more-thing-here
USER jenkins # drop back to the regular jenkins user - good practic
```

In such a derived image, you can customize your jenkins instance with hook scripts or additional plugins. For this purpose, use `/usr/share/jenkins/ref` as a place to define the default JENKINS_HOME content you wish the target installation to look like :

```
FROM jenkins
COPY plugins.txt /usr/share/jenkins/ref/
COPY custom.groovy /usr/share/jenkins/ref/init.groovy.d/custom.groo
RUN /usr/local/bin/plugins.sh /usr/share/jenkins/ref/plugins.txt
```

When jenkins container starts, it will check JENKINS_HOME has this reference content, and copy them there if required. It will not override such files, so if you upgraded some plugins from UI they won't be reverted on next start.

Also see JENKINS-24986 (https://issues.jenkins-ci.org/browse/JENKINS-24986)

For your convenience, you also can use a plain text file to define plugins to be installed (using core-support plugin format). All plugins need to be listed as there is no transitive dependency resolution.

```
pluginID:version
credentials:1.18
maven-plugin:2.7.1
...
```

And in derived Dockerfile just invoke the utility plugin.sh script

```
FROM jenkins
COPY plugins.txt /usr/share/jenkins/plugins.txt
RUN /usr/local/bin/plugins.sh /usr/share/jenkins/plugins.txt
```

Upgrading

All the data needed is in the /var/jenkins_home directory - so depending on how you manage that - depends on how you upgrade. Generally - you can copy it out - and then "docker pull" the image again - and you will have the latest LTS - you can then start up with -v pointing to that data (/var/jenkins_home) and everything will be as you left it.

As always - please ensure that you know how to drive docker - especially volume handling!

Image Variants

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

`jenkins:<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.

`jenkins:alpine`

This image is based on the popular Alpine Linux project (http://alpinelinux.org), available in the `alpine` official image (https://hub.docker.com/_/alpine). Alpine Linux is much smaller than most distribution base images (~5MB), and thus leads to much slimmer images in general.

This variant is highly recommended when final image size being as small as possible is desired. The main caveat to note is that it does use musl libc (http://www.musl-libc.org) instead of glibc and friends (http://www.etalabs.net/compare_libcs.html), so certain software might run into issues depending on the depth of their libc requirements. However, most software doesn't have an issue with this, so this variant is usually a very safe choice. See this Hacker News comment thread (https://news.ycombinator.com/item?id=10782897) 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 (https://hub.docker.com/_/alpine/) for examples of how to install packages if you are unfamiliar).

License

View license information (https://jenkins.io/license/) 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 `jenkins/` directory (https://github.com/docker-library/repo-info/tree/master/repos/jenkins).

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.

Docker Pull Command

```
docker pull jenkins
```