jenkinsci / docker Public

Code Issues 18 Pull requests 5 Actions Projects Wiki Security Insights

master docker / README.md Go to file

jcjveraa Add DNS-server issue explanation (#1917) 842e084 · 3 months ago History

Preview Code Blame 380 lines (260 loc) · 17.4 KB Raw

Official Jenkins Docker image

docker stars 4.1k docker pulls 4.8G chat on gitter

The Jenkins Continuous Integration and Delivery server available on Docker Hub.

This is a fully functional Jenkins server. https://jenkins.io/.

Usage

```
docker run -p 8080:8080 -p 50000:50000 --restart=on-failure jenkins/jenkins:lts-jdk17
```

NOTE: read the section *Connecting agents* below for the role of the `50000` port mapping. NOTE: read the section *DNS Configuration* in case you see the message "This Jenkins instance appears to be offline."

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 an explicit volume so you can manage it and attach to another container for upgrades :

```
docker run -p 8080:8080 -p 50000:50000 --restart=on-failure -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts-
jdk17
```

This will automatically create a 'jenkins_home' docker volume on the host machine. Docker volumes retain their content even when the container is stopped, started, or deleted.

NOTE: Avoid using a bind mount from a folder on the host machine into `/var/jenkins_home`, as this might result in file permission issues (the user used inside the container might not have rights to the folder on the host machine). If you *really* need to bind mount jenkins_home, ensure that the directory on the host is accessible by the jenkins user inside the container (jenkins user - uid 1000) or use `-u some_other_user` parameter with `docker run`.

```
docker run -d -v jenkins_home:/var/jenkins_home -p 8080:8080 -p 50000:50000 --restart=on-failure
jenkins/jenkins:lts-jdk17
```

This will run Jenkins in detached mode with port forwarding and volume added. You can access logs with command 'docker logs CONTAINER_ID' in order to check first login token. ID of container will be returned from output of command above.

Backing up data

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

Using a bind mount is not recommended since it can lead to permission issues. 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 Use volumes

Setting the number of executors

You can define the number of executors on the Jenkins built-in node using a groovy script. By default it is set to 2 executors, but you can extend the image and change it to your desired number of executors (recommended 0 executors on the built-in node) :

`executors.groovy`

```
import jenkins.model.*
Jenkins.instance.setNumExecutors(0) // Recommended to not run builds on the built-in node
```

and `Dockerfile`

```
FROM jenkins/jenkins:lts
COPY --chown=jenkins:jenkins executors.groovy /usr/share/jenkins/ref/init.groovy.d/executors.groovy
```

Connecting agents

You can run builds on the controller out of the box. The Jenkins project recommends that no executors be enabled on the controller.

In order to connect agents **through an inbound TCP connection**, map the port: `-p 50000:50000`. That port will be used when you connect agents to the controller.

If you are only using SSH (outbound) build agents, this port is not required, as connections are established from the controller. If you connect agents using web sockets (since Jenkins 2.217), the TCP agent port is not used either.

Passing JVM parameters

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

```
docker run --name myjenkins -p 8080:8080 -p 50000:50000 --restart=on-failure --env JAVA_OPTS=-
Dhudson.footerURL=http://mycompany.com jenkins/jenkins:lts-jdk17
```

JVM options specifically for the Jenkins controller should be set through `JENKINS_JAVA_OPTS`, as other tools might also respond to the `JAVA_OPTS` environment variable.

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 --restart=on-failure --env JAVA_OPTS="-
Djava.util.logging.config.file=/var/jenkins_home/log.properties" -v `pwd`/data:/var/jenkins_home
jenkins/jenkins:lts-jdk17
```

Configuring reverse proxy

If you want to install Jenkins behind a reverse proxy with a prefix, example: mysite.com/jenkins, you need to add environment variable `JENKINS_OPTS="--prefix=/jenkins"` and then follow the below procedures to configure your reverse proxy, which will depend if you have Apache or Nginx:

- Apache
- Nginx

DNS configuration

If the message "This Jenkins instance appears to be offline." appears on first startup, and the container logs show lines such as `java.net.UnknownHostException: updates.jenkins.io`, your container may be having issues with resolving DNS names.

To potentially solve the issue, start the container specifying a dns server (for example Cloudflare's 1.1.1.1 or Google's 8.8.8.8, or any other DNS server):

```
docker run -p 8080:8080 -p 50000:50000 --restart=on-failure --dns 1.1.1.1 --dns 8.8.8.8 jenkins/jenkins:lts-jdk17
```

Passing Jenkins launcher parameters

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

```
docker run jenkins/jenkins:lts-jdk17 --version
```

This will show the Jenkins version, the same as when you run Jenkins from an executable war.

You can also define Jenkins arguments via `JENKINS_OPTS`. This is useful for customizing arguments to the jenkins launcher in a derived Jenkins image. The following sample Dockerfile uses this option to force use of HTTPS with a certificate included in the image.

```
FROM jenkins/jenkins:lts-jdk17

COPY --chown=jenkins:jenkins certificate.pfx /var/lib/jenkins/certificate.pfx
COPY --chown=jenkins:jenkins https.key /var/lib/jenkins/pk
ENV JENKINS_OPTS="--httpPort=-1 --httpsPort=8083 --httpsKeyStore=/var/lib/jenkins/certificate.pfx --
httpsKeyStorePassword=Password12"
EXPOSE 8083
```

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

```
FROM jenkins/jenkins:lts-jdk17
ENV JENKINS_SLAVE_AGENT_PORT=50001
```

or as a parameter to docker,

```
docker run --name myjenkins -p 8080:8080 -p 50001:50001 --restart=on-failure --env JENKINS_SLAVE_AGENT_PORT=50001
jenkins/jenkins:lts-jdk17
```

Note: This environment variable will be used to set the system property `jenkins.model.Jenkins.slaveAgentPort`.

> If this property is already set in **JAVA_OPTS** or **JENKINS_JAVA_OPTS**, then the value of `JENKINS_SLAVE_AGENT_PORT` will be ignored.

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/jenkins:lts-jdk17
# if we want to install via apt
USER root
```

```
RUN apt-get update && apt-get install -y ruby make more-thing-here
# drop back to the regular jenkins user - good practice
USER jenkins
```

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/jenkins:lts-jdk17
COPY --chown=jenkins:jenkins custom.groovy /usr/share/jenkins/ref/init.groovy.d/custom.groovy
```

Preinstalling plugins

Install plugins

You can rely on the plugin manager CLI to pass a set of plugins to download with their dependencies. This tool will perform downloads from update centers, and internet access is required for the default update centers.

Setting update centers

During the download, the CLI will use update centers defined by the following environment variables:

- `JENKINS_UC` - Main update center. This update center may offer plugin versions depending on the Jenkins LTS Core versions. Default value: https://updates.jenkins.io
- `JENKINS_UC_EXPERIMENTAL` - Experimental Update Center. This center offers Alpha and Beta versions of plugins. Default value: https://updates.jenkins.io/experimental
- `JENKINS_INCREMENTALS_REPO_MIRROR` - Defines Maven mirror to be used to download plugins from the Incrementals repo. Default value: https://repo.jenkins-ci.org/incrementals
- `JENKINS_UC_DOWNLOAD` - Download url of the Update Center. Default value: `$JENKINS_UC/download`
- `JENKINS_PLUGIN_INFO` - Location of plugin information. Default value: https://updates.jenkins.io/current/plugin-versions.json

It is possible to override the environment variables in images.

❗ Note that changing update center variables **will not** change the Update Center being used by Jenkins runtime, it concerns only the plugin manager CLI.

Installing Custom Plugins

Installing prebuilt, custom plugins can be accomplished by copying the plugin HPI file into `/usr/share/jenkins/ref/plugins/` within the `Dockerfile`:

```
COPY --chown=jenkins:jenkins path/to/custom.hpi /usr/share/jenkins/ref/plugins/
```

Usage

You can run the CLI manually in Dockerfile:

```
FROM jenkins/jenkins:lts-jdk17
RUN jenkins-plugin-cli --plugins pipeline-model-definition github-branch-source:1.8
```

Furthermore it is possible to pass a file that contains this set of plugins (with or without line breaks).

```
FROM jenkins/jenkins:lts-jdk17
COPY --chown=jenkins:jenkins plugins.txt /usr/share/jenkins/ref/plugins.txt
RUN jenkins-plugin-cli -f /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.

In case you *do* want to override, append '.override' to the name of the reference file. E.g. a file named `/usr/share/jenkins/ref/config.xml.override` will overwrite an existing `config.xml` file in JENKINS_HOME.

Also see JENKINS-24986

Here is an example to get the list of plugins from an existing server:

```
JENKINS_HOST=username:password@myhost.com:port
curl -sSL "http://$JENKINS_HOST/pluginManager/api/xml?depth=1&xpath=/*/*/shortName|/*/*/version&wrapper=plugins" |
perl -pe 's/.*?<shortName>([\w-]+).*?<version>([^<]+)()(<\/\w+>)+/\1 \2\n/g'|sed 's/ /:/'
```

Example Output:

```
cucumber-testresult-plugin:0.8.2
pam-auth:1.1
matrix-project:1.4.1
script-security:1.13
...
```

For 2.x-derived images, you may also want to

```
RUN echo 2.0 > /usr/share/jenkins/ref/jenkins.install.UpgradeWizard.state
```

to indicate that this Jenkins installation is fully configured. Otherwise a banner will appear prompting the user to install additional plugins, which may be inappropriate.

Access logs

To enable Jenkins user access logs from Jenkins home directory inside a docker container, set the `JENKINS_OPTS` environment variable value to `--accessLoggerClassName=winstone.accesslog.SimpleAccessLogger --simpleAccessLogger.format=combined --simpleAccessLogger.file=/var/jenkins_home/logs/access_log`

Naming convention in tags

The naming convention for the tags on Docker Hub follows the format `<repository_name>:<tag>`, where the repository name is jenkins/jenkins and where the tag specifies the image version. In the case of the LTS and latest versions, the tags are `lts` and `latest`, respectively.

You can use these tags to pull the corresponding Jenkins images from Docker Hub and run them on your system. For example, to pull the LTS version of the Jenkins image use this command: `docker pull jenkins/jenkins:lts`

Docker Compose with Jenkins

To use Docker Compose with Jenkins, you can define a docker-compose.yml file including a Jenkins instance and any other services it depends on. For example, the following docker-compose.yml file defines a Jenkins controller and a Jenkins SSH agent:

```yaml
services:
  jenkins:
    image: jenkins/jenkins:lts
    ports:
      - "8080:8080"
    volumes:
      - jenkins_home:/var/jenkins_home
  ssh-agent:
    image: jenkins/ssh-agent
volumes:
  jenkins_home:
```

This `docker-compose.yml` file creates two containers: one for Jenkins and one for the Jenkins SSH agent.

The Jenkins container is based on the `jenkins/jenkins:lts` image and exposes the Jenkins web interface on port 8080. The `jenkins_home` volume is a named volume that is created and managed by Docker.

It is mounted at `/var/jenkins_home` in the Jenkins container, and it will persist the Jenkins configuration and data.

The ssh-agent container is based on the `jenkins/ssh-agent` image and runs an SSH server to execute Jenkins SSH Build Agent.

To start the Jenkins instance and the other services defined in the `docker-compose.yml` file, run the `docker compose up -d`.

This will pull the necessary images from Docker Hub if they are not already present on your system, and start the services in the background.

You can then access the Jenkins web interface on `http://localhost:8080` on your host system to configure and manage your Jenkins instance (where `localhost` points to the published port by your Docker Engine).

NOTE: read the section *DNS Configuration* in case you see the message "This Jenkins instance appears to be offline." In that case add the dns configuration to the yaml:

```
services:
  jenkins:
# ... other config
    dns:
      - 1.1.1.1
      - 8.8.8.8
# ... other config
```

Updating plugins file

The plugin-installation-manager-tool supports updating the plugin file for you.

Example command:

```
JENKINS_IMAGE=jenkins/jenkins:lts-jdk17
docker run -it ${JENKINS_IMAGE} bash -c "stty -onlcr && jenkins-plugin-cli -f /usr/share/jenkins/ref/plugins.txt --av
mv plugins2.txt 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!

If you mount the Jenkins home directory to a Docker named volume, then the upgrade consists of `docker pull` and nothing more.

We recommend using `docker compose`, especially in cases where the user is also running a parallel nginx/apache container as a reverse proxy for the Jenkins container.

Upgrading plugins

By default, plugins will be upgraded if they haven't been upgraded manually and if the version from the docker image is newer than the version in the container. Versions installed by the docker image are tracked through a marker file.

To force upgrades of plugins that have been manually upgraded, run the docker image with `-e PLUGINS_FORCE_UPGRADE=true`.

The default behaviour when upgrading from a docker image that didn't write marker files is to leave existing plugins in place. If you want to upgrade existing plugins without marker you may run the docker image with `-e TRY_UPGRADE_IF_NO_MARKER=true`. Then plugins will be upgraded if the version provided by the docker image is newer.

Hacking

If you wish to contribute fixes to this repository, please refer to the dedicated documentation.

Security

For information related to the security of this Docker image, please refer to the dedicated documentation.

Questions?

We're on Gitter, https://gitter.im/jenkinsci/docker