# Installing with Docker

## Overview

> **Make sure you have reviewed the overall installation process**
>
> Before you proceed with the instructions on this page, make sure you have reviewed the whole installation procedure as described in Installing Artifactory.

Artifactory Docker images can be pulled from Bintray and run as a Docker container.

To do this, you need to have Docker client properly installed and configured on your machine. For details about installing and using Docker, please refer to the Docker documentation.

> **Running with Docker for Artifactory 4.x**
>
> Artifactory as a Docker container has been completely redesigned in version 5.0. If you are running previous versions of Artifactory, please refer to Running with Docker in the Artifactory 4.x User Guide

> **Docker Compose**
>
> The way we recommend running Artifactory on Docker is to orchestrate your setup using Docker Compose. This will ensure you have all the required services specified in a single YAML file with pre-configured parameters.

> **Adding Self Signed Certificates to Java cacerts**
>
> You can add extra self signed certificates to Artifactory's Java cacerts so they'll be trusted by Artifactory. To do this, put the needed certificate(s) in a directory on your host and pass this directory as a volume to the "docker run" command mounted to /artifactory_extra_certs: "docker run --name artifactory -d -p 8081:8081 -v /path/to/certs:/artifactory_extra_certs docker.bintray.io/jfrog/artifactory-pro:latest" Container's entrypoint will add them to Java's cacert keystore before starting Artifactory.

---

## Using Docker Compose

To setup an Artifactory environment made of multiple containers (for example, a database, an NGINX load balancer and Artifactory each running in a different container), you can use `docker-compose`.

For more details on Docker Compose, please refer to the Docker documentation.

**Artifactory OSS**, **Artifactory Pro** and **Artifactory HA** can all be run using Docker Compose. For detailed documentation and sample Compose files showing a variety of ways to setup Artifactory with Docker Compose, please refer to the artifactory-docker-examples repository on GitHub.

**Page Contents**

**Read more**

- Building Artifactory OSS
- Changing the Database
- JFrog Products Container Base Image

**Integration Benefits**

JFrog Artifactory and Docker Registries

---

# Artifactory on Docker

Running Artifactory as a container is simple and straightforward, and involves the following basic steps:

- Pulling the Artifactory Docker Image
- Running the Artifactory Container

Since the Artifactory instance running in a Docker container is mutable, all data and configuration files will be lost once the container is removed. If you want your data to persist (for example when upgrading to a new version), you should also follow the next step.

- Managing Data Persistence

## Pulling the Artifactory Docker Image

The Artifactory Docker image may be pulled from Bintray by executing the corresponding Docker command below depending on whether you are pulling Artifactory OSS or Artifactory Pro:

**Pulling the Artifactory Pro Docker Image**

```
docker pull docker.bintray.io/jfrog/artifactory-pro:latest
```

or

**Pulling the Artifactory OSS Docker Image**

```
docker pull docker.bintray.io/jfrog/artifactory-oss:latest
```

or

**Pulling the Artifactory CE Docker Image**

```
docker pull docker.bintray.io/jfrog/artifactory-cpp-ce
```

## Running an Artifactory Container

You can list the Docker images you have downloaded using the **docker images** command, which should display something like the following output:

```
$ docker images

REPOSITORY                                TAG                 IMAGE ID            CREATED             SIZE
docker.bintray.io/jfrog/artifactory-pro   latest                          da70b82904e7        2 days
ago          861.5 MB
...
```

To start an Artifactory container, use the corresponding command below according to whether you are running Artifactory Pro or Artifactory OSS:

**Running Artifactory Pro in a container**

```
$ docker run --name artifactory -d -p 8081:8081 docker.bintray.io/jfrog/artifactory-pro:latest
```

or

**Running Artifactory OSS in a container**

```
$ docker run --name artifactory -d -p 8081:8081 docker.bintray.io/jfrog/artifactory-oss:latest
```

or

**Running Artifactory CE in a container**

```
$ docker run --name artifactory -d -p 8081:8081 docker.bintray.io/jfrog/artifactory-cpp-ce:latest
```

## Setting Java Memory Parameters

To control the memory used by Artifactory, you can pass the environment variable EXTRA_JAVA_OPTIONS.

For example:

**Passing JVM memory to Artifactory Docker container**

```
$ docker run --name artifactory -d -p 8081:8081 -e EXTRA_JAVA_OPTIONS='-Xms512m -Xmx2g -Xss256k -XX:
+UseG1GC' docker.bintray.io/jfrog/artifactory-pro:latest
```

## Supported Environment Variables

Artifactory Docker images can be customized using environment variables.

Pass the values as environment variables with your Docker execution command.

For example:

```
docker run -d --name art -e SERVER_XML_ARTIFACTORY_MAX_THREADS=500 -p 8081:8081
docker.bintray.io/jfrog/artifactory-pro:6.6.3
```

Artifactory will start with maxThreads set to "500" in the Tomcat server.xml.

### Java system properties

You can pass Java system properties to the JVM running Artifactory

| Variable | Functionality | Default |
|---|---|---|
| **EXTRA_JAVA_OPTIONS** | Pass Java options to Artifactory JVM | |

### Database

The database environment variables are documented in Changing the Database page

### Tomcat Server.xml

| Variable | Functionality | Default |
|---|---|---|

| | | |
|---|---|---|
| **SERVER_XML_ARTIFACTORY_PORT** | Sets the custom Artifactory port. | 8081 |
| **SERVER_XML_ARTIFACTORY_MAX_THREADS** | Sets the custom Artifactory maxThreads. | 200 |
| **SERVER_XML_ACCESS_MAX_THREADS** | sets the custom Access maxThreads. | 50 |
| **SERVER_XML_ARTIFACTORY_EXTRA_CONFIG** | Adds an extra Artifactory connector config. | |
| **SERVER_XML_ACCESS_EXTRA_CONFIG** | Adds an extra Access connector config | |
| **SERVER_XML_EXTRA_CONNECTOR** | Add another connector to Tomcat. For example to support SSL. | |

## HA-Specific Environment Variables

**Passing Environment Variables to the entrypoint script**

The entrypoint script of the Artifactory Pro Docker image accepts various environment variables. These are documented in the table below, and can be used to manipulate various HA-specific settings. Setting the following variables is particularly useful when using an orchestration tool such as Kubernetes or Docker Compose to spin up new Artifactory nodes. For more details on configuring the *ha-node.properties* please refer to Setting Up Your Storage Configuration.

| Variable | **Functionality** | **Default value** |
|---|---|---|
| HA_IS_PRIMARY | Determines whether the node is set as a Primary node or as a Member node in the cluster. | - |
| HA_NODE_ID | The value of the 'node.id' parameter in the ha-node.properties file. | node-$(hostname) |
| HA_HOST_IP | The IP of the container. This variable is used to compose a full context.url, only when the $HA_CONTEXT_URL variable is not set. Determined by running 'hostname -i'. | $(hostname -i) |
| HA_CONTEXT_URL | The value of the 'context.url' parameter in the generated ha-node.properties file. This is the node URL exposed to cluster members. If not set, the $HA_HOST_IP variable will be used to derive the full context.url. | http://$HA_HOST_IP:8081/artifactory |
| HA_MEMBERSHIP_PORT | The Hazelcast membership port of the node. | 10002 |
| **ART_PRIMARY_BASE_URL** | Set this on a member node only if the nodes are not going to be a part of the same docker network, so that they're not reachable to each other by the container name, or if you the name of the primary node **container** is not "artifactory-node1". The entrypoint script would send an HTTP request to the primary node using this URL to wait for the Primary node to start up. | http://artifactory-node1:8081/artifactory |
| HA_DATA_DIR | The value for the 'artifactory.ha.data.dir' parameter in the ha-node.properties file. | /var/opt/jfrog/artifactory/data |
| HA_BACKUP_DIR | The value for the 'artifactory.ha.backup.dir' parameter in the ha-node.properties file. | /var/opt/jfrog/artifactory/backup |

# Managing Data Persistence

**The "artifactory" user**

Previously, the Artifactory Docker container started as user `root`, but was run by user `artifactory`. From version 6.2, user `artifactory` is used to both start and run the Docker container. Note that:

- the `artifactory` user default ID is 1030
- the `artifactory` user must have write privileges to any persistent storage mounted on the Artifactory container

For your data and configuration to remain once the Artifactory Docker container is removed, you need to store them on an external volume mounted to the Docker container. There are two ways to do this:

- Using Host Directories
- Using a Docker Named Volume

### Using Host Directories

The external volume is a directory in your host's file system (such as **/var/opt/jfrog/artifactory**). When you pass this to the `docker run` command, the Artifactory process will use it to read configuration and store its data.

To mount the above example, you would use the following command:

```
$ docker run --name artifactory-pro -d -v /var/opt/jfrog/artifactory:/var/opt/jfrog/artifactory -p 8081:8081
docker.bintray.io/jfrog/artifactory-pro:latest
```

This mounts the **/var/opt/jfrog/artifactory** directory on your host machine to the container's **/var/opt/jfrog/artifactory** and will then be used by Artifactory for configuration and data.

### Using a Docker Named Volume

In this case, you create a docker named volume and pass it to the container. By default, the named volume is a local directory under */var/lib/docker/volumes/<name>*, but can be set to work with other locations. For more details, please refer to the Docker documentation for Docker Volumes.

The example below creates a Docker named volume called **artifactory_data** and mounts it to the Artifactory container under **/var/opt/jfrog/artifactory:**

```
$ docker volume create --name artifactory5_data
$ docker run --name artifactory-pro -d -v artifactory5_data:/var/opt/jfrog/artifactory -p 8081:8081 docker.
bintray.io/jfrog/artifactory-pro:latest
```

In this case, even if the container is stopped and removed, the volume persists and can be attached to a new running container using the above `docker run` command.

---

## Extra Configuration Directory

You can mount extra configuration files, such as *binarystore.xml*, *artifactory.lic* or *db.properties*, that are needed for your Artifactory installation.
To do this, you need to mount the file or directory on the host into the Artifactory Docker container's */artifactory_extra_conf* folder. When the Artifactory Docker container starts, it will copy the files from */artifactory_extra_conf* to *ARTIFACTORY_HOME/etc* (usually */var/opt/jfrog/artifactory/etc*).

The files mounted into */artifactory_extra_conf* will be copied over to *ARTIFACTORY_HOME/etc* every time the container starts, so you should avoid modifying the files in *ARTIFACTORY_HOME/etc*.

**Example 1: Passing in a custom db.properties file**

```
$ docker run --name artifactory-pro -d -v /var/opt/jfrog/artifactory:/var/opt/jfrog/artifactory -v /conf/db.
properties:/artifactory_extra_conf/db.properties -p 8081:8081 docker.bintray.io/jfrog/artifactory-pro:latest
```

**Example 2: Passing in a custom binarystore.xml**

```
$ docker run --name artifactory-pro -d -v /var/opt/jfrog/artifactory:/var/opt/jfrog/artifactory -v /conf
/binarystore.xml:/artifactory_extra_conf/binarystore.xml -p 8081:8081 docker.bintray.io/jfrog/artifactory-
pro:latest
```

---

# Artifactory NGINX Docker image

The Artifactory Docker image can be run with an Nginx Docker image that can be used to manage SSL, reverse proxy and other web server features. For configuration details, please refer to Configuring NGINX.

A custom Docker image that is already setup for Artifactory with NGINX and is available at: `docker.bintray.io/jfrog/nginx-artifactory-pro`

**Docker Compose**

We recommend running this container along with the Artifactory container using an orchestration tool such as docker-compose, which allows easy networking and linking of the two containers. See artifactory docker-compose examples in JFrog's artifactory-docker-examples repository on GitHub.

**Artifactory NGINX Docker user**

From version 6.2, the Artifactory Nginx Docker container starts and runs as user `nginx`. Note that:

- the `nginx` user ID is 104; the group ID is 107
- any mounted volume must be writable by the `nginx` user

To run the image locally, use:

```
$ docker run --name artifactory-pro-nginx -d -p 8000:80 -p 8443:443 docker.bintray.io/jfrog/nginx-
artifactory-pro:latest
```

This will start an NGINX instance with default settings

- NGINX listening on ports 80 and 443
- Self signed SSL key and certificate
- Forwarding requests to host `artifactory`

## Custom Configurations

We recommend customizing the Artifactory NGINX container to your needs as described below.

### Using Your Own SSL Key and Certificate

- Place your SSL key and certificate in a directory on your host
- Mount the directory into the Artifactory NGINX container to */var/opt/jfrog/nginx/ssl*

```
$ docker run --name artifactory-pro-nginx -d -p 8000:80 -p 8443:443 -v $HOST_SSL_PATH:/var/opt/jfrog
/nginx/ssl docker.bintray.io/jfrog/nginx-artifactory-pro:latest
```

### Using a Custom Artifactory Configuration File

To customize the *artifactory.conf* file used by the NGINX container, create your own *artifactory.conf* file and mount it into the container to: */var/opt/jfrog/nginx/conf.d/artifactory.conf.*

For this, you should disable the auto update of the configuration feature using the SKIP_AUTO_UPDATE_CONFIG environment variable

```
$ docker run --name artifactory-pro-nginx -d -p 8000:80 -p 8443:443 \
        -e SKIP_AUTO_UPDATE_CONFIG=true \
        -v $CUSTOM_ART_CONF_FILE:/var/opt/jfrog/nginx/conf.d/artifactory.conf docker.bintray.io/jfrog/nginx-
artifactory-pro:latest
```

---

## Running Artifactory Docker with a Custom User/Group ID

Artifactory Docker container can be configured to run with a custom user/group ID by passing the following parameter: "--user $uid:$gid".



The mounted host directory must be writable by the given user id.

The following example will get Artifactory running as user ID 1234 and Group ID 4321.

```
$ docker run --name artifactory-pro --user 1234:4321 -d -v /var/opt/jfrog/artifactory:/var/opt/jfrog
/artifactory -p 8081:8081 docker.bintray.io/jfrog/artifactory-pro:latest
```

---

## Upgrading Artifactory

For details on how to upgrade Artifactory running in a Docker container, please refer to Running in a Docker Container in the Upgrading Artifactory page.

---

## Running Artifactory With a Different Database

By default, Artifactory runs with an embedded Derby Database that comes built-in, however, Artifactory supports additional databases. To switch to one of the other supported databases, please refer to Changing the Database.

---

## Building Artifactory OSS From Sources

The Artifactory OSS Docker image sources are available for download allowing you to build the image yourself. For details, please refer to Building Artifactory OSS.

---

## Accessing Artifactory

Once the Artifactory container is up and running, you access Artifactory in the usual way by browsing to:

```
http://SERVER_DOMAIN:8081/artifactory
```

---

## Docker for Windows limitation

There is a known limitation with running with Docker on Windows.

The limitation is described in the following JIRA issue. There is an optional workaround there, but it's not recommended for production deployments.

---

## Troubleshooting Docker

Please refer to the main Troubleshooting page.

---

Watch the Screencast