Docker Registry

Overview

Set up a secure private Docker registry in minutes to manage all your Docker images while exercising fine-grained access control. Artifactory places no limitations and lets you set up any number of Docker registries, through the use of local, remote and virtual Docker repositories, and works transparently with the Docker client to manage all your Docker images, whether created internally or downloaded from remote Docker resources such as Docker Hub.

Multiple Docker Registries
Artifactory lets you define as many Docker registries as you wish. This enables you to manage each project in a distinct registry and exercise better access control to your Docker images.

Use Docker Naturally
Artifactory supports the relevant calls of the Docker Registry API so that you can transparently use the Docker client to access images through Artifactory.

Secure private Docker Registry with Fine-grained Access Control
Local Docker Repositories are where you store internal Docker images for distribution across your organization. With the fine-grained access control provided by built-in security features, Artifactory offers secure Docker push and pull with local Docker repositories as fully functional, secure, private Docker registries.

Consistent and reliable access to remote images
Remote Docker Repositories in Artifactory proxy external resources such as Docker Hub, or a remote Docker repository in another Artifactory instance, and cache downloaded images. As a result, overall networking is reduced, and access to images on these remote resources is faster, consistent and reliable.

OCI Support
Artifactory is OCI compliant and supports OCI clients, enabling you to deploy and resolve OCI images in Docker Registries. The OCI client Singularity is not supported.

Docker Buildx Support
Artifactory supports Docker Buildx, allowing you to easily build and push multi-architecture images using the `Docker buildx` CLI. For more information, see Pushing Images in Bulk Using the Docker Buildx CLI.

Confidently Promoting Images to Production
Artifactory lets you promote Docker images, as immutable, stable binaries, through the quality gates all the way to production.

Unlimited Docker Hub access
Artifactory provides you with unlimited, high-performant access to Docker Hub and to Docker Official Images to simplify cloud-native application development, without Docker Hub image-pull limits. This allows you to streamline, automate and simplify the way DevOps teams work.
* Available to SaaS cloud JFrog Platform subscribers, including free subscription offered on AWS, GCP & Azure.

Registries and Repositories

Both Artifactory and Docker use the term "repository", but each uses it in a different way.

A **Docker repository** is a hosted collection of tagged images that, together, create the file system for a container

A **Docker registry** is a host that stores Docker repositories

An **Artifactory repository** is a hosted collection of Docker repositories, effectively, a Docker registry in every way, and one that you can access transparently with the Docker client.

Since Artifactory places no limitation on the number of repositories you may create, you can manage any number of Docker registries in Artifactory.

Page Contents

- Overview
- Registries and Repositories
- Getting Started With Artifactory as a Docker Registry
- Configuring Docker Repositories
  - Local Docker Repositories
  - Remote Docker Repositories
  - Virtual Docker Repositories
  - Reverse Proxy Settings
- Promoting Docker Images
- Pushing and Pulling Images
  - Set Me Up
  - Pushing Multi-Architecture Docker Images to Artifactory
- Browsing Docker Repositories
- Searching for Docker Images
- Listing Docker Images
- Deletion and Cleanup
  - Limiting Unique Tags
- Docker Build Information
- Migrating from Docker V1 to Docker V2

Read More

- Getting Started with Artifactory as a Docker Registry
- Advanced Topics
- Working with Docker Content Trust
- Using Docker V1

Integration Benefits

Docker Registry

Getting Started With Artifactory as a Docker Registry

There are these main ways to get started using Docker with Artifactory:

1. Artifactory SaaS account
2. Artifactory On-Prem

For more details, please refer to **Getting Started with Artifactory as a Docker Registry**.

Configuring Docker Repositories

Artifactory supports three types of repositories when working with Docker:

- **Local repositories** are a place for your internal Docker images. Through Artifactory's security capabilities, these are secure private Docker registries.
- **Remote repositories** are used to proxy remote Docker resources such as Docker Hub.
- **Virtual repositories** can aggregate multiple Docker registries thus enabling a single endpoint you can use for both pushing and pulling Docker images. This enables the admin to manage the different Docker registries without the users knowing, and continue to work with the same end point.

**Make sure to go to the Advanced tab of each repository and set the Registry Port if you are using the Port method for Docker. Then, the reverse proxy generator should add a new section in for the specified port.

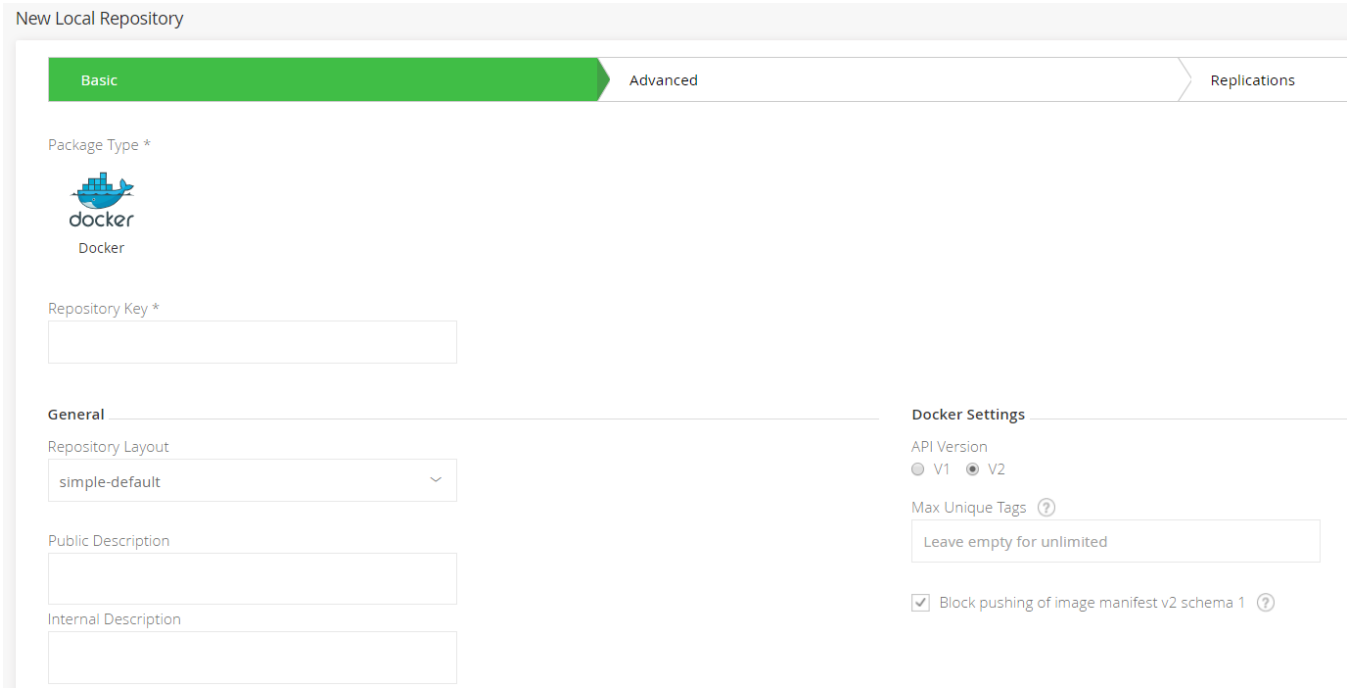
> ⚠ **Do not use underscores when naming Docker repositories**
>
> Due to a limitation in the Docker client, underscores are not permitted in Docker registry names. Therefore, when naming Artifactory Docker repositories, you should not use an underscore. For example, the Docker client will not be able to communicate with a repository named `test_docker_repo`, however it **will** work with a repository named `test.docker.repo`.

Local Docker Repositories

A local Docker repository is where you can deploy and host your internal Docker images. It is, in effect, a Docker registry able to host collections of tagged Docker images which are your Docker Repositories. Once your images are hosted, you can exercise fine-grained access control, and share them across your organization through replication or by being proxied by repositories in other Artifactory instances.

To define a local Docker repository:

1. From the **Administration** module, select **Repositories** | **Repositories** | **Local**.
2. Click **New Local Repository** and select **Docker** from the **Select Package Type** dialog.
3. Set the **Repository Key**, and in the Docker Settings section, select **V2** as the Docker API version.
4. Set **Max Unique Tags**. This specifies the maximum number of unique tags, per repository, that should be stored for a Docker image. Once the number of tags for an image exceeds this number, older tags will be removed. Leaving the field blank (default) means all tags will be stored.



Remote Docker Repositories

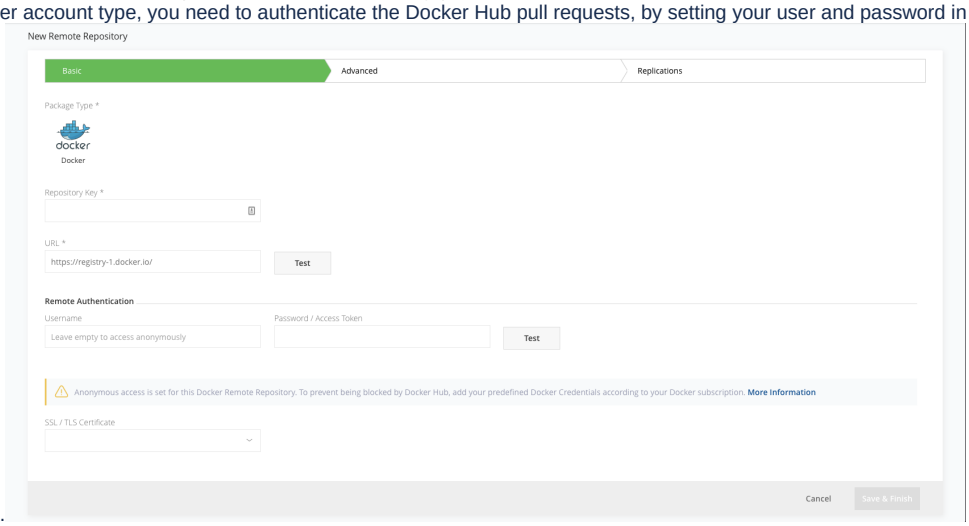
With Docker, you can proxy a remote Docker registry through remote repositories. A Remote Repository defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as *https://registry-1.docker.io/* (which is the Docker Hub), or even a Docker repository managed at a remote site by another instance of Artifactory.

Docker images requested from a remote repository are cached on demand. You can remove downloaded images from the remote repository cache, however, you can not manually push Docker images to a remote Docker repository.

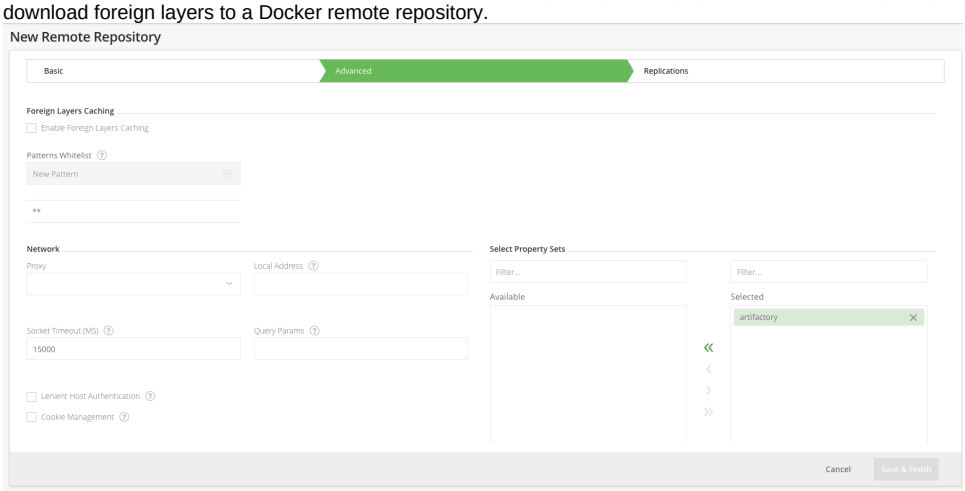
> **Action Required to Prevent Docker Remote Registry Restrictions**
>
> In lieu of the latest Docker remote repository limitations enforced by Docker, anonymous users will be blocked when reaching the download rate limit of 100 pulls per six hours. To prevent this from happening, you are required to authenticate with Docker Hub, by setting your Docker account user and password in your **Remote Docker Repositories**.

To define a remote repository to proxy a remote Docker registry follow the steps below:

1. From the **Administration** module, select **Repositories** | **Repositories** | **Remote**.
2. Click **New Remote Repository** and select **Docker** from the **Select Package Type** dialog.
3. In the Basic tab, set the **Repository Key** value, and specify the URL to the remote registry in the **URL** field.
   If you are proxying the Docker Hub, use `https://registry-1.docker.io/` as the URL, and make sure the **Enable Token Authentication** checkbox is checked (these are the default settings).
4. To use your Docker account type, you need to authenticate the Docker Hub pull requests, by setting your user and password in your basic Docker repository.



5. Click the Advanced tab to configure the Advanced Docker Repository settings you can enable Foreign Layers Caching to allow Artifactory to download foreign layers to a Docker remote repository.



6. Select the **Enable Foreign Layers Caching** checkbox to allow Artifactory to download foreign layers to a Docker remote repository.
7. You have an option to apply Patterns Allow List by setting Include patterns to match external URLs when trying to download foreign layers. Specify an Allow List of Ant-style path expressions that specify where foreign layers may be downloaded from. Supported expressions include (*, **, ?).
   By default, this field is set to ** which means that foreign layers may be downloaded from any external source.
   For example, specifying **/github.com/** will only allow downloading foreign layers from a github.com host.
8. To configure the Network settings, see Network Settings.
9. Click **Save and Finish**.

Docker Repository Path and Domain

When accessing a remote Docker repository through Artifactory, the repository URL must be prefixed with **`api/docker`** in the path.

For Example:

*`http://my-remote-site:8081/artifactory/`**`api/docker`**`/<repository key>`*

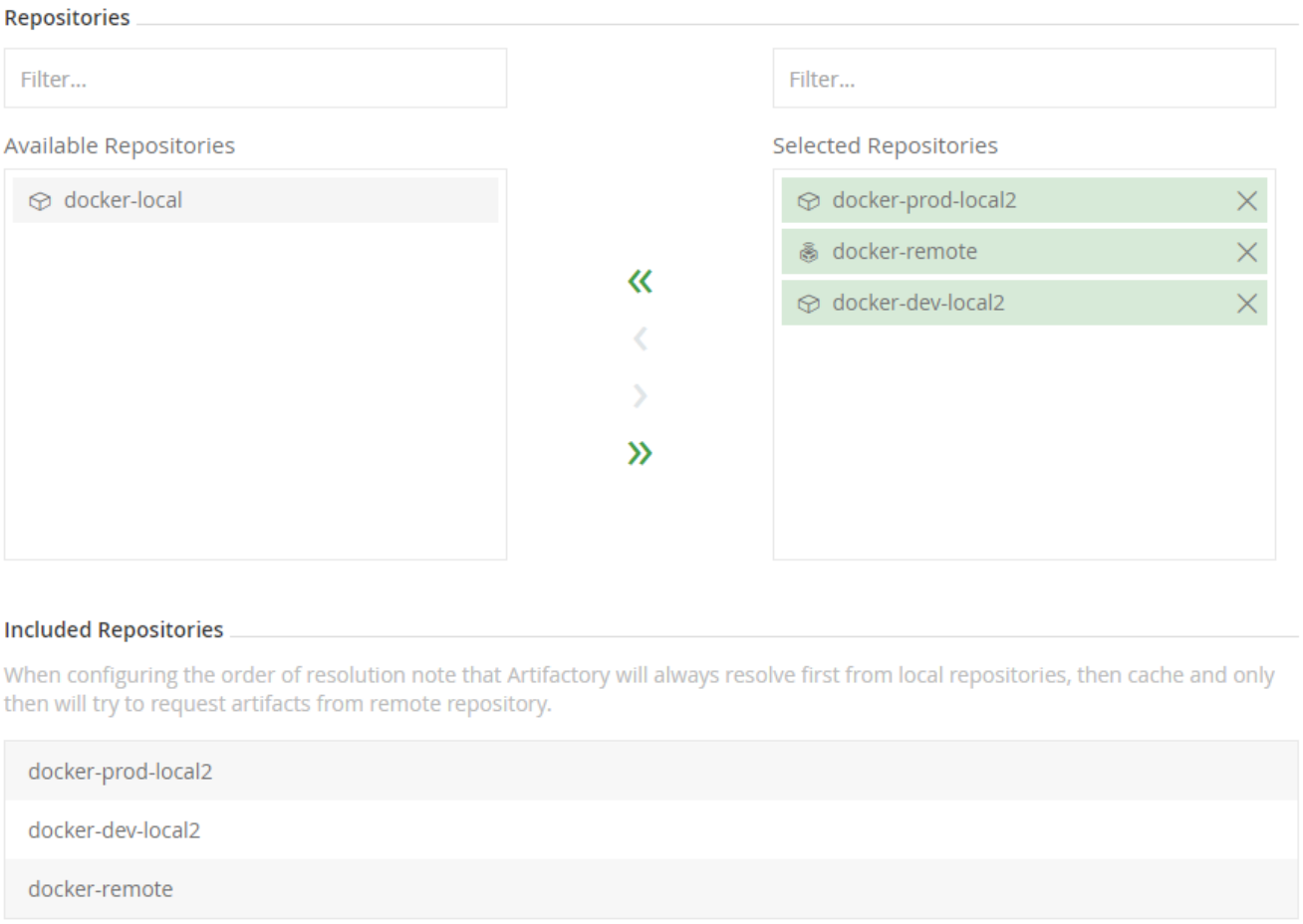
Virtual Docker Repositories

Artifactory supports virtual Docker Repositories. A Virtual Repositories defined in Artifactory aggregates images from both local and remote repositories that are included in the virtual repositories.

This allows you to access images that are hosted locally on local Docker repositories, as well as remote images that are proxied by remote Docker repositories, and access all of them from a single URL defined for the virtual repository. Using virtual repositories can be very useful since users will continue to work with the virtual repository while the admin can manage the included repositories, replace the default deployment target and those changes will be transparent to the users.

To define a virtual Docker repository follow the steps below:

1. From the **Administration** module, select **Repositories** | **Repositories** | **Virtual**.
2. Click **New Virtual Repository** and select **Docker** from the **Select Package Type** dialog.
3. Set the **Repository Key** value.
4. Select the underlying local and remote Docker repositories to include under the **Repositories** section.
5. You can optionally also configure your **Default Deployment Repository.** This is the repository to which Docker images uploaded to this virtual repository will be routed, and once this is configured, your virtual Docker repository is a fully-fledged Docker registry. Using the default deployment repository, you can set up your virtual repository to wrap a series of repositories that represent the stages of your pipeline, and then promote images from the default deployment repository through the pipeline to production. Any repository that represents a stage in your pipeline within this virtual repository can be configured with permissions for authenticated or unauthenticated (anonymous) access according to your needs.



Resolve Latest Docker Image

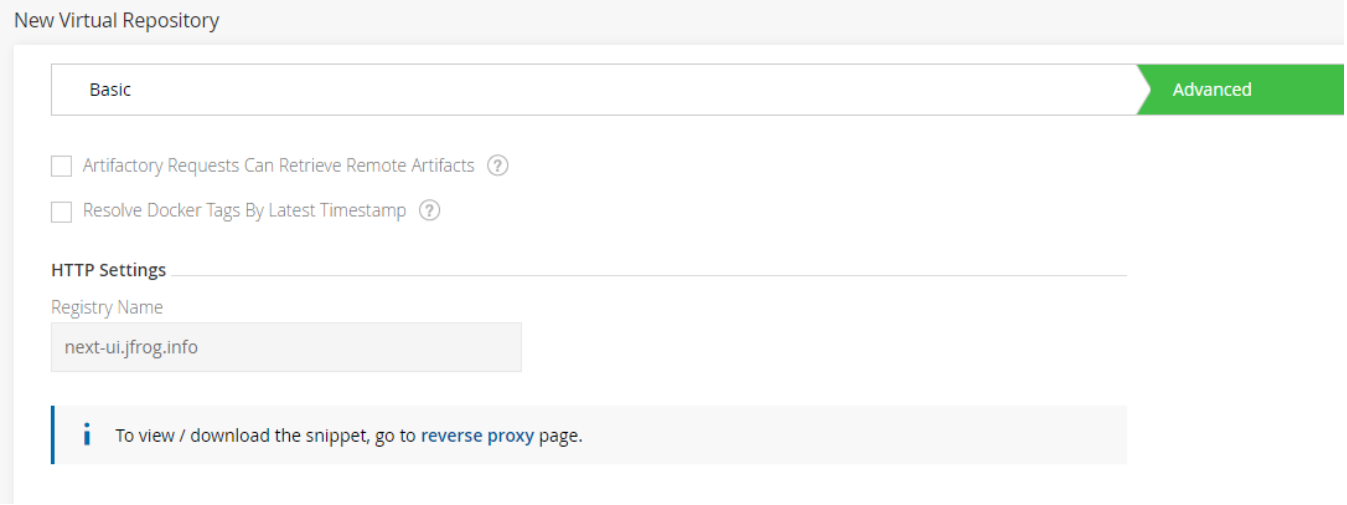
To set your virtual Docker repository to pull Docker images according to their modification time, enable *Resolve Docker Tags By Latest Timestamp*. This is useful in scenarios where two or more aggregated repositories contain the same tag name. For example, `busybox:1.1`.

When enabled, instead of fetching the image that is positioned higher in the resolution order in the virtual repository, Artifactory will return the Docker image last deployed to one of the aggregated repositories in the Virtual repository. Artifactory will first try to fetch the tag from the Local repositories according to the modification time, if not found, it will continue to try to fetch the image from the Remote repositories according to the resolution order.

This functionality is useful for multi-site environments where you create the same image on two different instances.

REST API

This can also be configured by setting the *resolveDockerTagsByTimestamp* parameter to true (false by default) when *creating a new repository* using REST API.



Reverse Proxy Settings

Artifactory supports access to Docker registries either through a reverse proxy (using the subdomain method or through port bindings), or using direct access.

When accessing through a reverse proxy, if you are using the Artifactory Reverse Proxy configuration generator you can configure a Docker repository's reverse proxy settings under the **Advanced** settings tab. For details, please refer to Docker Reverse Proxy Settings.

Promoting Docker Images

Artifactory supports promoting Docker images from one Docker repository in Artifactory to another.

Promoting is useful when you need to move Docker images through different acceptance and testing stages, for example, from a development repository, through the different gateways all the way to production. Instead of rebuilding the image multiple times using promotion will ensure the image you will have in your production environment is the one built by your CI server and passed all the relevant tests.

Promotion can be triggered using the following endpoint with cURL:

```
POST api/docker/<repoKey>/v2/promote
{
    "targetRepo" : "<targetRepo>",
    "dockerRepository" : "<dockerRepository>",
    "tag" : "<tag>",
        "targetTag" : "<tag>",
    "copy": <true | false>
}
```

where:

repoKey	Source repository key
targetRepo	The target repository to move or copy

dockerRepository	The docker repository name to promote
tag	An optional tag name to promote, if null - the entire docker repository will be promoted. Default: `latest`
targetTag	The new tag that the image should have after being promoted if you want to
copy	When true, a copy of the image is promoted. When false, the image is moved to the target repository

An example for promoting the docker image *`jfrog/ubuntu"]`* with all of it's tags from *`docker-local`* to *`docker-prod`* using cURL would be:

```
curl -i -uadmin:password -X POST "https://artprod.company.com/api/docker/<repoKey>/v2/promote" -H "Content-Type: application/json" -d '{"targetRepo":"docker-prod","dockerRepository":"jfrog/ubuntu"}'
```

https://artprod.company.com/api/docker/<repoKey>/v2/promote

Notice that the above example is executed through your reverse proxy. To go directly through Artifactory, you would execute this command as follows:

```
curl -i -uadmin:password -X POST "http://localhost:8080/artifactory/api/docker/docker-local/v2/promote" -H "Content-Type: application/json" -d '{"targetRepo":"docker-prod","dockerRepository":"jfrog/ubuntu"}'
```

The following example adds retagging with a specific version of the *`jfrog/ubuntu`* image (4.9.0) being retagged to `latest` as it gets promoted:

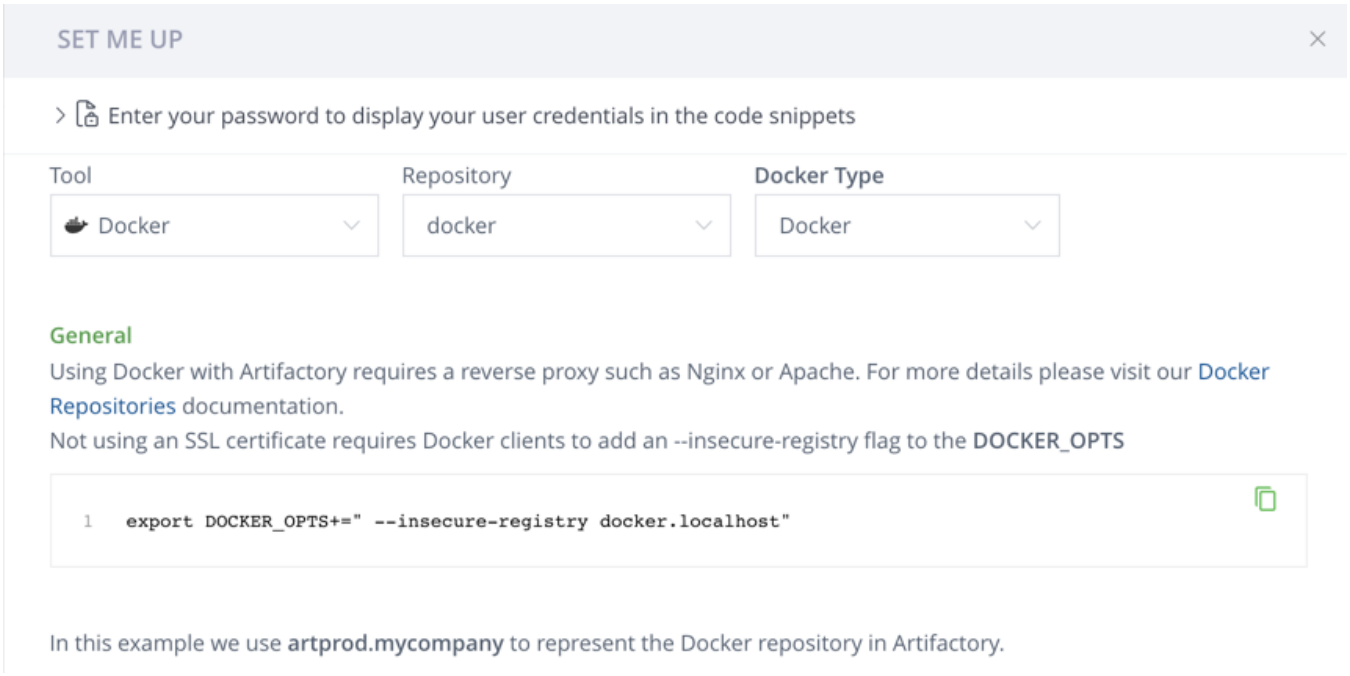
```
curl -i -uadmin:password -X POST "https://artprod.company.com/api/docker/docker-local/v2/promote" -H "Content-Type: application/json" -d '{"targetRepo":"docker-prod","dockerRepository":"jfrog/ubuntu", "tag" : "4.9.0", "targetTag" : "latest"}'
```

Pushing and Pulling Images

Set Me Up

To get the corresponding `docker push` and `docker pull` commands for any repository, go to **Artifactory** | **Artifacts** | **Artifact Repository Browser**, in the **Application** module, and click **Set Me Up**.

Docker



To login use the *docker login* command.

```
docker login artprod.mycompany
```

And provide your Artifactory username and password or API key.
If anonymous access is enabled you do not need to login.

To manually set your credentials, or if you are using Docker v1, copy the following snippet to your ~/.docker/config.json file.

```
{
    "auths": {
        "https://artprod.mycompany": {
            "auth": "<USERNAME>:<PASSWORD> (converted to base 64)",
            "email": "youremail@email.com"
        }
    }
```

To enter multiple registries see the following example.

Deploy

To push an image tag an image using the *docker tag* and then *docker push* command.

```
docker tag <IMAGE_ID> artprod.mycompany/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```

```
docker push artprod.mycompany/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```

Resolve

To pull an image use the *docker pull* command specifying the docker image and tag.

```
docker pull artprod.mycompany/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```

Reverse Proxy

Click here to view the reverse proxy configuration snippet.

To use reverse proxy configuration, place the snippet in the sites-enabled directory and reload. This will affect Artifactory reverse proxy configuration and Docker repositories if configured.

Podman

SET ME UP

> Enter your password to display your user credentials in the code snippets

Tool	Repository	Docker Type
Docker	docker	Podman

General

To login use the *podman login* command.

```
podman login artprod.mycompany
```

And provide your Artifactory username and password or API key.
If anonymous access is enabled you do not need to login.

To manually set your credentials, copy the following snippet to either your `${XDG_RUNTIME_DIR}/containers/auth.json` or `~/.docker/config.json` file. You can also set the podman login **authfile** flag and provide the path of your authentication file.

```
{
    "auths": {
        "https://artprod.mycompany": {
            "auth": "<USERNAME>:<PASSWORD> (converted to base 64)",
            "email": "youremail@email.com"
        }
    }
}
```

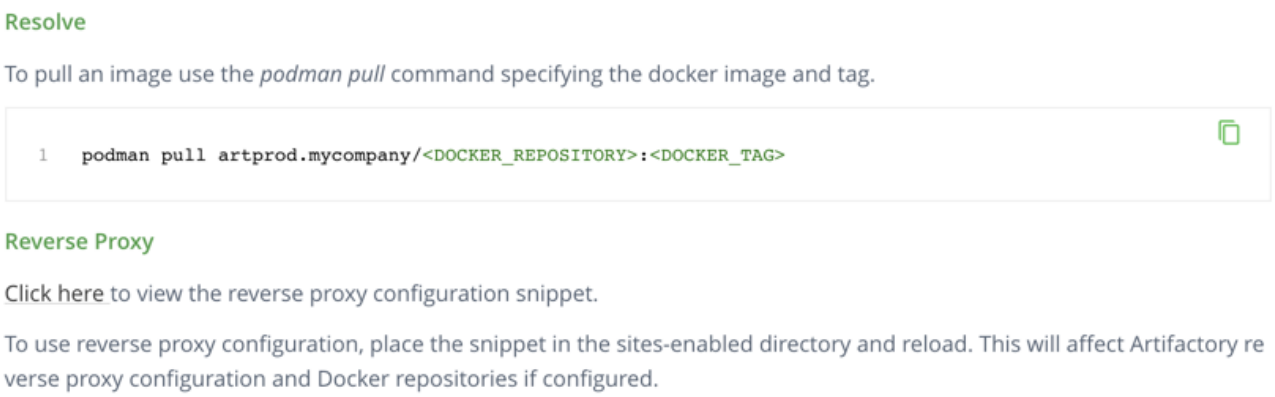
To enter multiple registries see the following example.

Deploy

To push an image tag an image using the *podman tag* and then *podman push* command.

```
podman tag <IMAGE_ID> artprod.mycompany/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```

```
podman push artprod.mycompany/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```



Pushing Multi-Architecture Docker Images to Artifactory

JFrog Artifactory supports the following methods for pushing multi-architecture Docker images to a Docker Registry:

- Pushing Docker images for each architecture one by one
- Pushing Docker images in bulk using the Docker Buildx CLI (Supported from Artifactory 7.21.2 and higher)
- Pushing Multi-Architecture Docker Images Using Docker Build from Artifactory 7.21.2 and higher.

Pushing Docker Images One by One

> **Backward Compatibility**
>
> To learn how the standard Docker Pull REST API functions in Artifactory 7.21.2, see Pushing Multi-Architecture Docker Images Using Docker Build.

You can push multi-architecture Docker images, using a '`Manifest Lists`' file, (officially referred by Docker as the '`fat manifest file`'), which references image manifests for platform-specific versions of an image. For more information, click here.

The process of pushing multi-architecture Docker images is similar to the standard Docker Push process, with a few exceptions:

1. Each architecture gets a different tag.
2. After all the architectures have been built and pushed, a single '`fat manifest file`' is created and contains all of the images with the relevant tagging.
3. After pushing the '`fat manifest file`', the images are published with the given tags.

```
$ docker build -t domain/docker/multiarch-image:amd64 --build-arg ARCH=amd64/<docker_file>
$ docker push domain/docker/multiarch-image:amd64

$ docker build -t domain_name: port/docker/multiarch-image:arm64 --build-arg ARCH=arm64/<docker_file>
$ docker push domain/docker/multiarch-image:arm64

$ docker manifest create \
domain_name:port1/docker/multiarch-image:tag \
--amend domain/docker/multiarch-image:amd64 \
--amend domain/docker/multiarch-image:arm64 \
$ docker manifest push domain/docker/multiarch-image:my-tag
```

Pushing Images in Bulk Using the Docker Buildx CLI

From Artifactory 7.21.2, the `Docker buildx` command is supported, allowing you to create and upload Docker '`manifest lists`' to the Docker registry in Artifactory. `Docker buildx` allows you to build and push multi-architecture images using a single command instead of having to build and push each of the architecture images separately. For more information, see Working with buildx.

To support the `Docker buildx`, Artifactory saves each architecture of the image under the following path structure with the tag that includes the originally published tag, the image operating system, and the image architecture.

```
imageName:tag-os-arch
```

The following example shows the Docker BuildX API usage.

```
docker buildx build  --platform linux/amd64,linux/arm64 --tag domain/docker/multiarch-image:tag --
output=type=image,push=true  --push  .
```

Pushing Multi-Architecture Docker Images Using Docker Build

From Artifactory version 7.21.2 and higher, if you continue to push multi-architecture Docker images using `Docker build`, all your pushed images will be duplicated, and the architecture tag will be automatically added to each image.

In the following example, pushing the following images using Docker Build will result in Artifactory automatically duplicating the images and adding the `linux` tag to each image.

List Manifest

```
docker.artifactory.<domain_name>/test/busybox:1.33
```

Image A

1. Image A is pushed during the build.

   ```
   docker.artifactory<domain_name>/test/busybox:1.33-amd64
   ```

2. Artifactory duplicates the image and adds the `'linux'` tag.

   ```
   docker.artifactory.<domain_name>/test/busybox:1.33-linux-amd64
   ```

Image B

1. Image B si pushed during the build.

   ```
   docker.artifactory.<domain_name>/test/busybox:1.33-s390x
   ```

2. Artifactory duplicates the image and adds the `'linux'` tag.

   ```
   artifactory.us..<domain_name>/test/busybox:1.33-linux-s390x
   ```

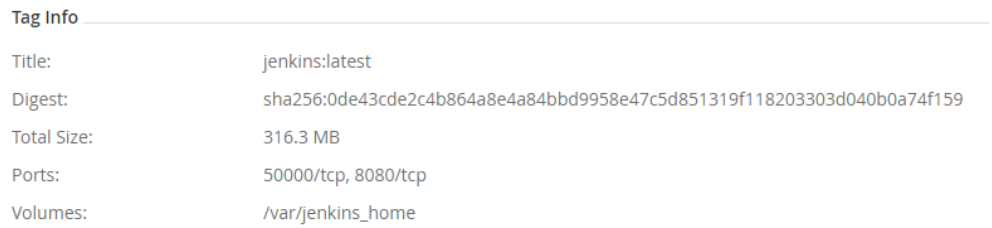
Browsing Docker Repositories

For general information on how to browse repositories, please refer to Browsing Artifactory.

The **Docker Info** tab presents three sections: **Tag Info**, **Docker Tag Visualization**, and **Labels**.

Tag Info

Presents basic details about the selected tag.



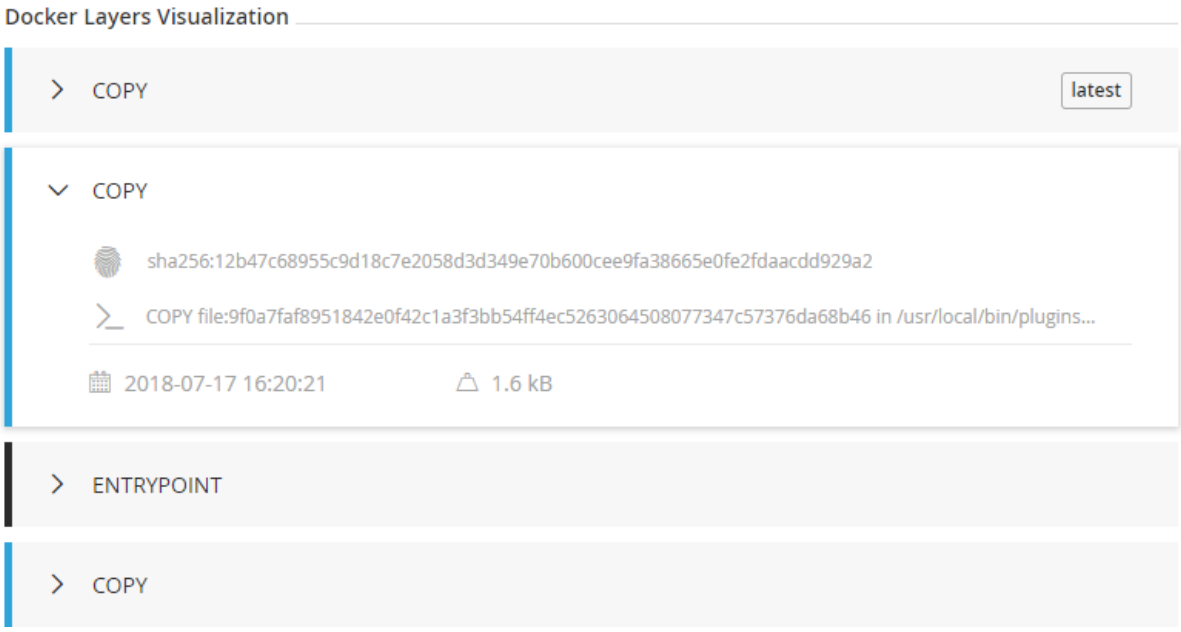
Title	The Docker tag name.
Digest	The tag's SHA 256 digest.
Total Size	The total size of the image
Label Count	The number of labels attached to this tag. Click the label count to view the attached labels at the bottom of the screen.

Docker Tag Visualization

This section maps the entire set of commands used to generate the selected tag along with the digest of the corresponding layer. Essentially, you would see the same series of commands using `docker history`.

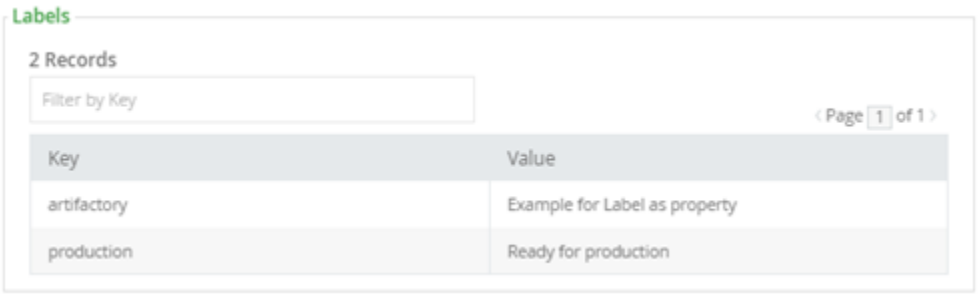
You can select any layer of the image to view the following properties.

Symbol	Property
</>	The layer ID
	The layer size
	The timestamp when the layer was created
>_	The command that created the layer
	The digest

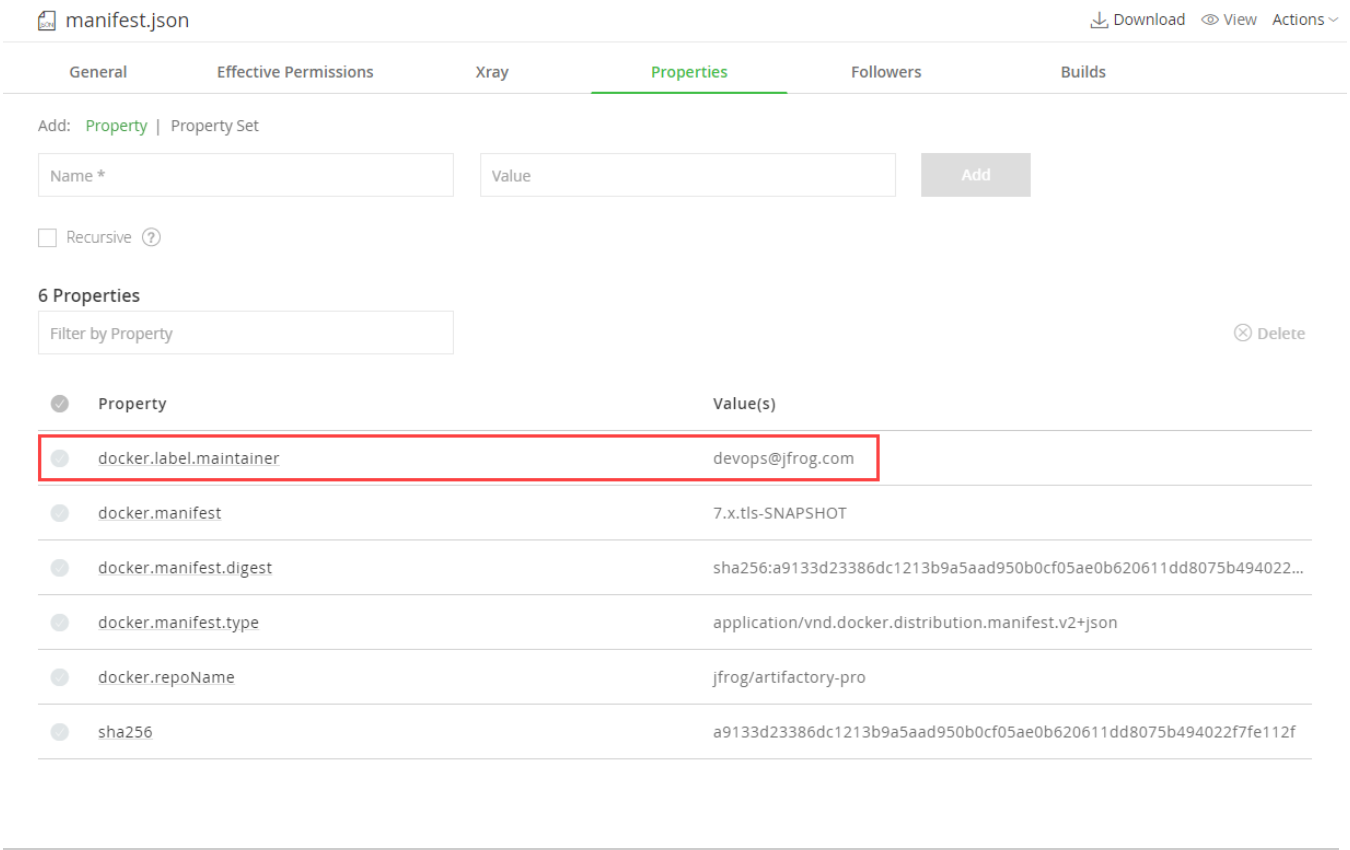


Labels

This section displays the labels attached to the image.

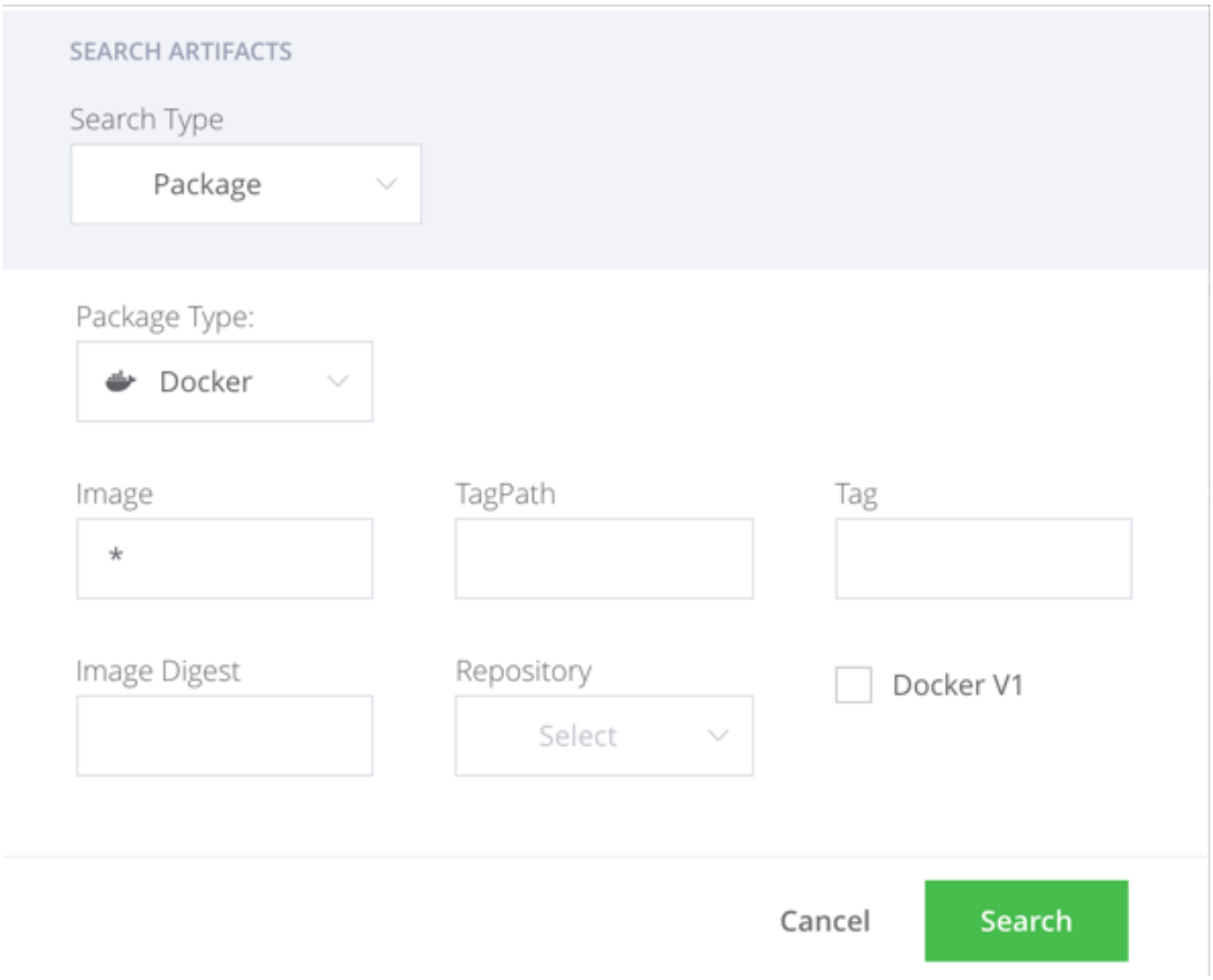


Note also, that Artifactory extracts any labels associated with a Docker image and creates corresponding properties on the *`manifest.json`* file which you can use to specify search parameters, this can be used to easily add additional metadata to any image.



Searching for Docker Images

You can search for Docker images by their name, tag or image digest using the Artifact Package Search or through the REST API.



Listing Docker Images

Artifactory supports the following REST API endpoints related to Docker registries:

- List Docker Images provides a list of Docker images in the specified Artifactory Docker registry. This endpoint mimics the Docker `_catalog` REST API.
- List Docker Tags provides a list of tags for the specified Docker image.

Artifactory also supports pagination for this endpoint.

To enable fetch from cache using the ListDockerRepositories and the ListDockerTags REST APIs, set the *`artifactory.docker.catalogs.tags.fallback.fetch.remote.cache`* system property to true (default false) in the artifactory.system.properties file:

```
## Enable fetch from cache in Docker repositories
#artifactory.docker.catalogs.tags.fallback.fetch.remote.cache=true
```

Artifactory needs to be restarted for this change to take effect.

Deletion and Cleanup

Artifactory natively supports removing tags and repositories and complies with the Docker Hub spec.

Deletion of Docker tags and repositories automatically cleans up any orphan layers that are left (layers not used by any other tag/repository).

Currently, the Docker client does not support DELETE commands, but deletion can be triggered manually. To delete an entire Docker repository using cURL, execute the following command:

```
curl -u<user:password> -X DELETE "<Artifactory URL>/artifactory/<Docker v2 repository name>/<image namespace>"
```

Or for a specific tag version:

```
curl -u<user:password> -X DELETE "<Artifactory URL>/artifactory/<Docker v2 repository name>/<image namespace>/<tag>"
```

For example, to remove the latest tag of an Ubuntu repository:

```
//Removing the latest tag from the "jfrog/ubuntu" repository
curl -uadmin:password -X DELETE "https://artprod.company.com/artifactory/dockerv2-local/jfrog/ubuntu/latest"
```

Empty Directories

Any empty directories that are left following removal of a repository or tag will automatically be removed during the next folder pruning job (which occurs every 5 minutes by default).

Limiting Unique Tags

To avoid clutter and bloat in your Docker registries caused by many snapshots being uploaded for an image, set the `Max Unique Tags` field in the Local Docker Repository configuration to limit the number of unique tags.

Docker Build Information

You may store exhaustive build information in Artifactory by running your Docker builds with JFrog CLI.

JFrog CLI collects build-info from your build agents and then publishes it to Artifactory. Once published, the build info can be viewed in the Build Browser under **Builds**.

For more details on Docker build integration using JFrog CLI, please refer to Managing Docker Images in the JFrog CLI User Guide.

Migrating from Docker V1 to Docker V2

If you are still using Docker V1, we strongly recommend upgrading to Docker V2. This requires that you migrate any Docker repositories that were created for Docker V1, and is done with a simple cURL endpoint.

For details, please refer to Migrating a V1 repository to V2 under the Using Docker V1 documentation.

Using Docker V1?

This document shows how to use Artifactory with the Docker V2 . If you are using the Docker V1, please refer to Using Docker V1.