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.

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

Smart Search
Using Artifactory's Package Search, find your images in the most natural way for Docker using the image name, tag or digest.

JFrog End-to-End Platform
Through Artifactory's tight integration with JFrog Bintray, you can manage your Docker images from development, through your pipeline, all the way to distribution.

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

Read More

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

Learn About

JFrog Container Registry

Integration Benefits

JFrog Artifactory and Docker Registries

Getting Started With Artifactory as a Docker Registry

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

1. Artifactory SaaS account
2. Using Docker Compose (1-minute setup)
3. 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 their 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

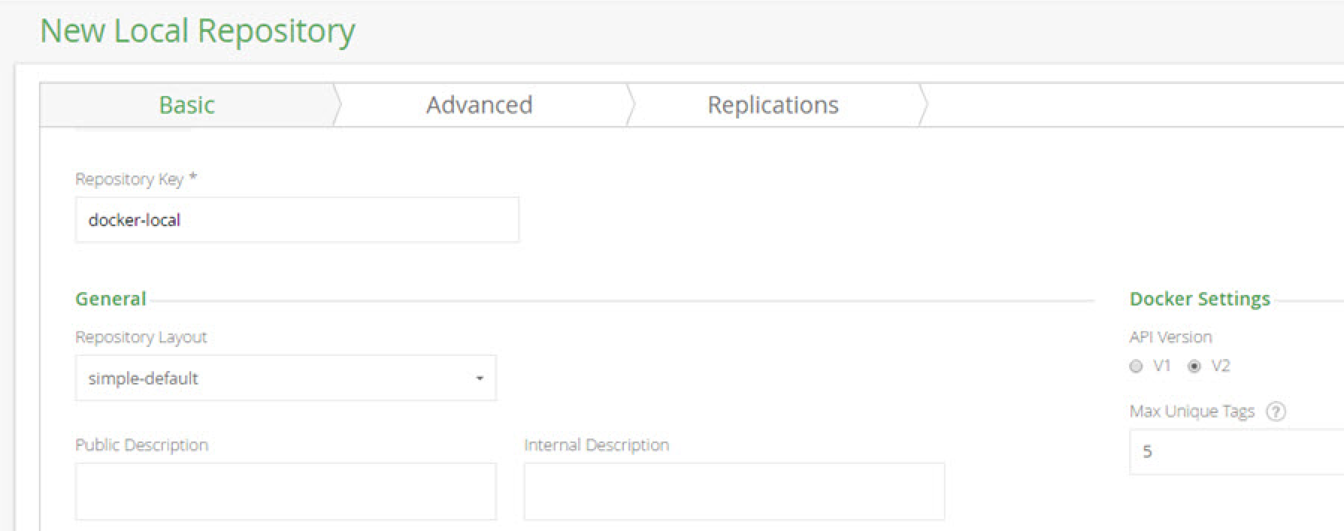
Due to a limitation in the Docker client, underscores are not permitted in Docker registry names. Therefore, when naming Artifactory Docker repositories, you should not use an underscore. For example, the Docker client will not be able to communicate with a repository named *`test_docker_repo`*, however 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, follow the steps below:

1. Create a new Local Repository and set **Docker** as the **Package Type.**
2. Set the **Repository Key**, and in the Docker Settings section, select **V2** as the Docker API version.
3. 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.



New Local Repository

Repository Key * docker-local

General

Repository Layout simple-default

Public Description

Internal Description

Docker Settings

API Version ○ V1 ● V2

Max Unique Tags 5

Remote Docker Repositories

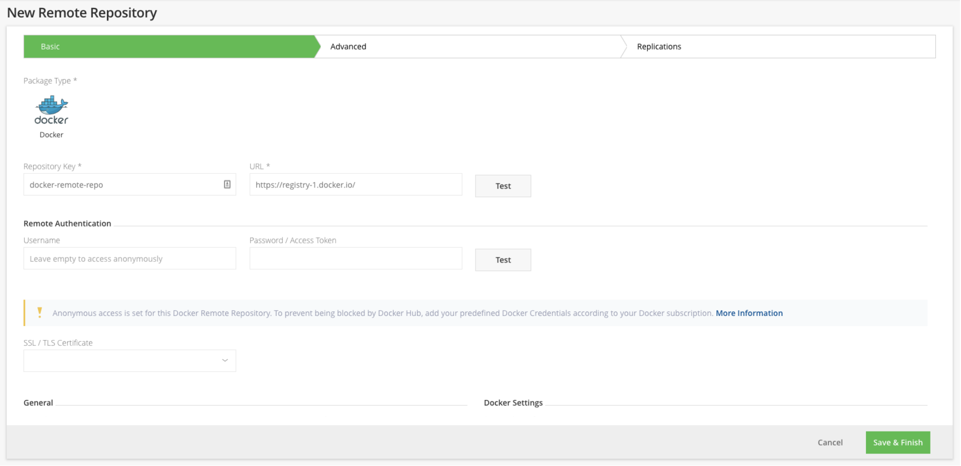
With Docker, you can proxy a remote Docker registry through remote repositories. A Remote Repository defined in Artifactory serves as a caching proxy for a registry managed at a remote URL such as *https://registry-1.docker.io/* (which is the Docker Hub), or even a Docker repository managed at a remote 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**
>
> From Artifactory 6.23.1., 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 the Docker Hub pull requests, by setting your user and password in your Basic Remote Docker Repositories.

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

1. Create a new Remote Repository and set **Docker** as the **Package Type.**
2. Set the **Repository Key** value, and specify the URL to the remote registry in the **URL** field.
   In order to use your Docker account type, you need to authenticate the Docker Hub pull requests, by setting your user and password in your Basic Remote Docker Repository tab.



New Remote Repository

Basic | Advanced | Replications

Package Type *

Docker

Repository Key *: docker-remote-repo

URL *: https://registry-1.docker.io/ Test

Remote Authentication

Username: Leave empty to access anonymously

Password / Access Token: Test

! Anonymous access is set for this Docker Remote Repository. To prevent being blocked by Docker Hub, add your predefined Docker Credentials according to your Docker subscription. More Information

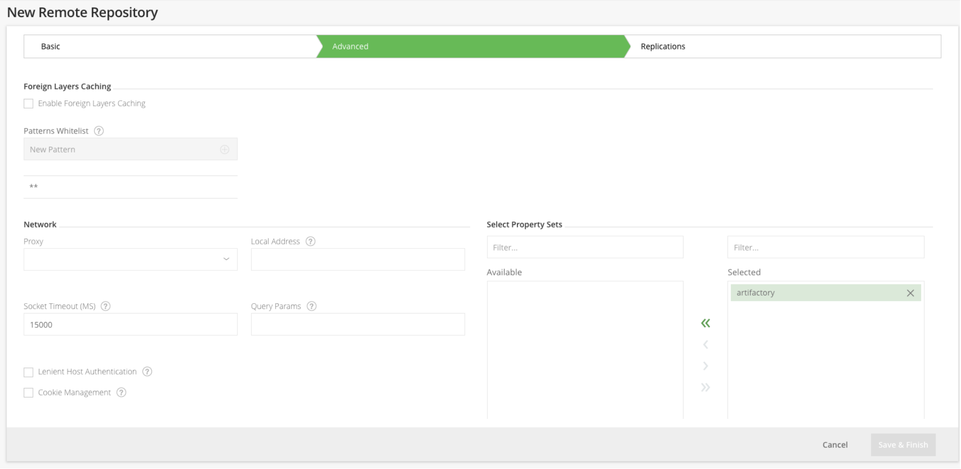
SSL / TLS Certificate

General

Docker Settings

Cancel Save & Finish

   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).
3. 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.



New Remote Repository

Basic | Advanced | Replications

Foreign Layers Caching

□ Enable Foreign Layers Caching

Patterns Whitelist ⓘ

New Pattern

**

Network

Proxy

Local Address ⓘ

Socket Timeout (MS) ⓘ: 15000

Query Params ⓘ

□ Lenient Host Authentication ⓘ

□ Cookie Management ⓘ

Select Property Sets

Filter...

Available

Filter...

Selected: artifactory

Cancel Save & Finish

4. Select the **Enable Foreign Layers Caching** checkbox to allow Artifactory to download foreign layers to a Docker remote repository.
5. You have an option to apply Patterns Whitelist by setting Include patterns to match external URLs when trying to download foreign layers.
   Specify a whitelist 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.
6. To configure the Network settings, see Network Settings.
7. 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>*

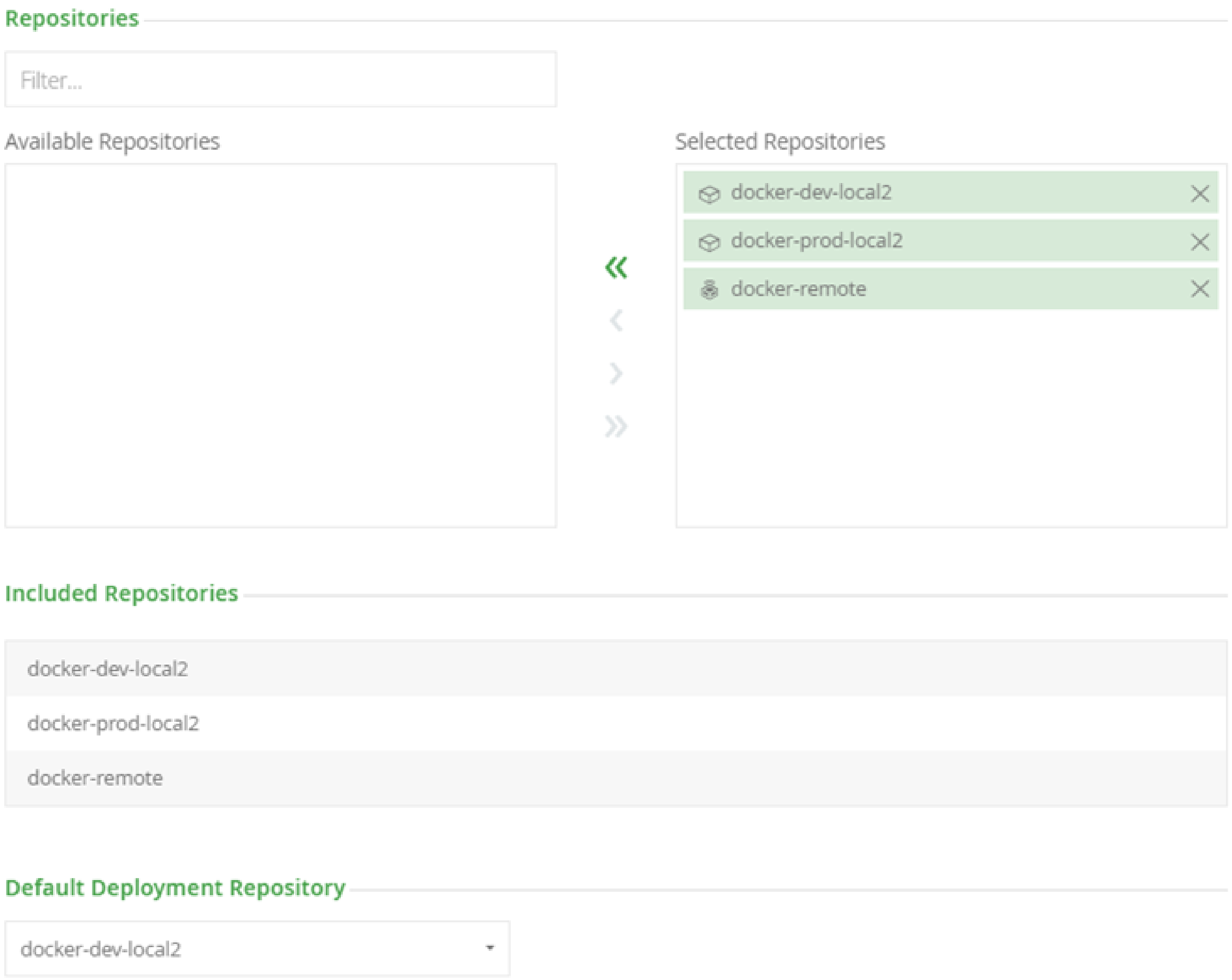
Virtual Docker Repositories

From version 4.1, Artifactory supports virtual Docker Repositories. A Virtual Repository 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. Create a new Virtual Repository and set **Docker** as the **Package Type.**
2. Set the **Repository Key** value.
3. Select the underlying local and remote Docker repositories to include under the **Repositories** section.
4. 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.



Repositories

Filter...

Available Repositories

Selected Repositories

- docker-dev-local2
- docker-prod-local2
- docker-remote

Included Repositories

- docker-dev-local2
- docker-prod-local2
- docker-remote

Default Deployment Repository

docker-dev-local2

Resolve Latest Docker Image

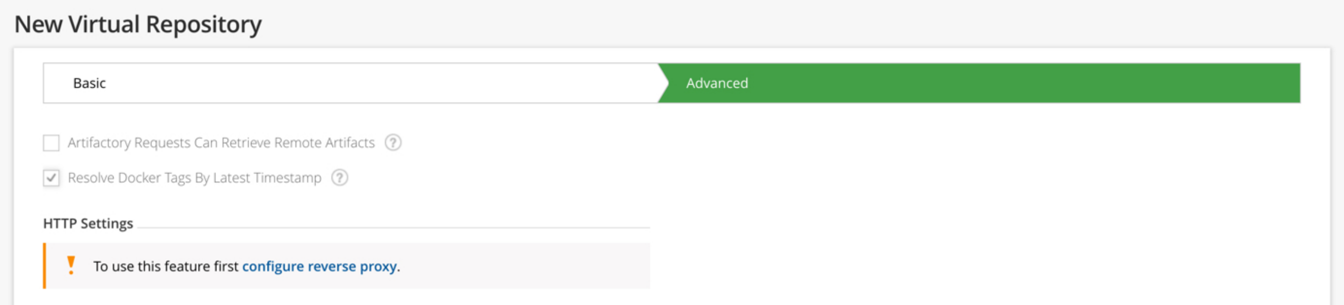
To set your virtual Docker repository to pull Docker images according to their modification time, enable *Resolve Docker Tags By Latest Timestamp*. This is useful in scenarios where two or more aggregated repositories contain the same tag name. For example, *busybox:1.1*.

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.



New Virtual Repository

☐ Artifactory Requests Can Retrieve Remote Artifacts

☑ Resolve Docker Tags By Latest Timestamp

HTTP Settings

To use this feature first configure reverse proxy.

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: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/v2/promote" -H "Content-Type: application
/json" -d '{"tagetRepo":"docker-prod","dockerRepository":"jfrog/ubuntu"}'
```

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"}'
```

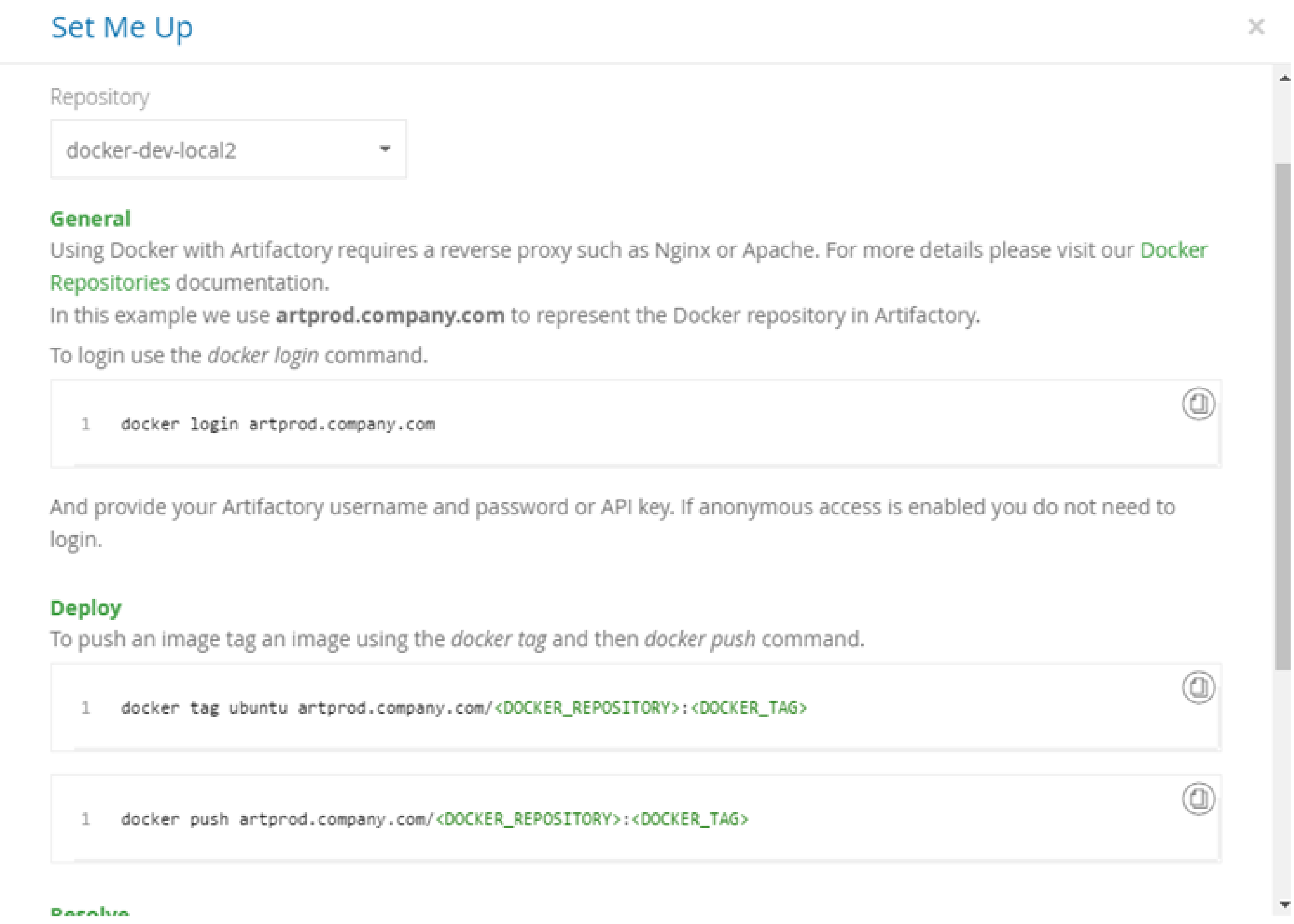
The following example adds retagging with a specific version of the "*jfrog/ubuntu"* image (4.9.0) being retagged to "latest" as it gets promoted:

```
curl -i -uadmin:password -X POST "https://artprod.company.com/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, select it in the Tree Browser and click **Set Me Up** button.



Set Me Up

Repository

docker-dev-local2

General

Using Docker with Artifactory requires a reverse proxy such as Nginx or Apache. For more details please visit our Docker Repositories documentation.

In this example we use **artprod.company.com** to represent the Docker repository in Artifactory.

To login use the *docker login* command.

```
docker login artprod.company.com
```

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

Deploy

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

```
docker tag ubuntu artprod.company.com/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```

```
docker push artprod.company.com/<DOCKER_REPOSITORY>:<DOCKER_TAG>
```

Resolve

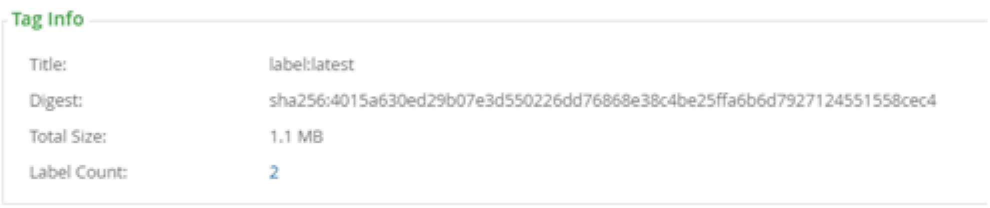
Browsing Docker Repositories

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.



Tag Info

Title:	label:latest
Digest:	sha256:4015a630ed29b07e3d550226dd76868e38c4be25ffa6b6d7927124551558cec4
Total Size:	1.1 MB
Label Count:	2

Title	The Docker tag name.
Digest	The tag's SHA 256 digest.
Total Size	The total size of the image

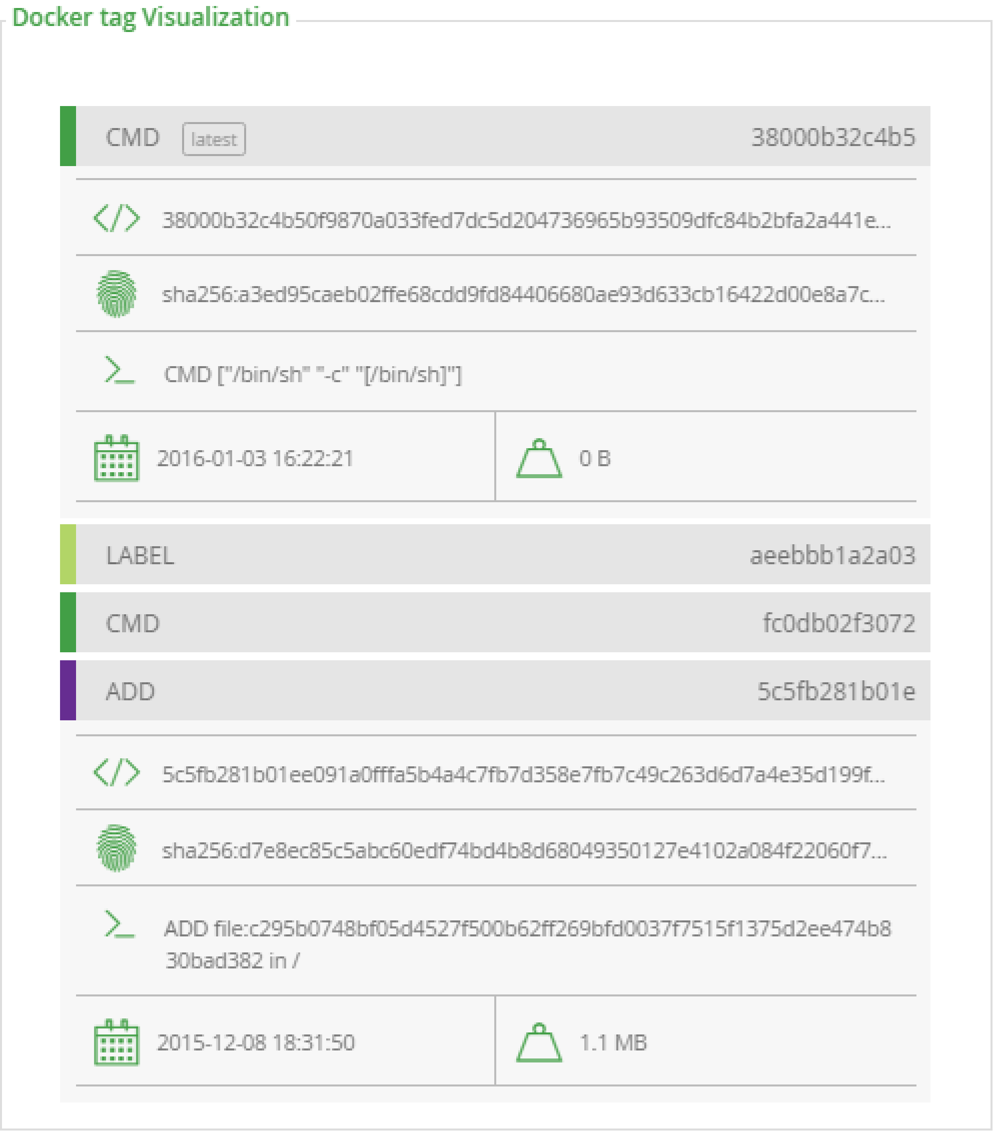
Label Count	The number of labels attached to this tag. ✓ Click the label count to view the attached labels at the bottom of the screen.

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



Docker tag Visualization

CMD latest 38000b32c4b5

38000b32c4b50f9870a033fed7dc5d204736965b93509dfc84b2bfa2a441e...

sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c...

CMD ["/bin/sh" "-c" "[/bin/sh]"]

2016-01-03 16:22:21

0 B

LABEL aeebbb1a2a03

CMD fc0db02f3072

ADD 5c5fb281b01e

5c5fb281b01ee091a0fffa5b4a4c7fb7d358e7fb7c49c263d6d7a4e35d199f...

sha256:d7e8ec85c5abc60edf74bd4b8d68049350127e4102a084f22060f7...

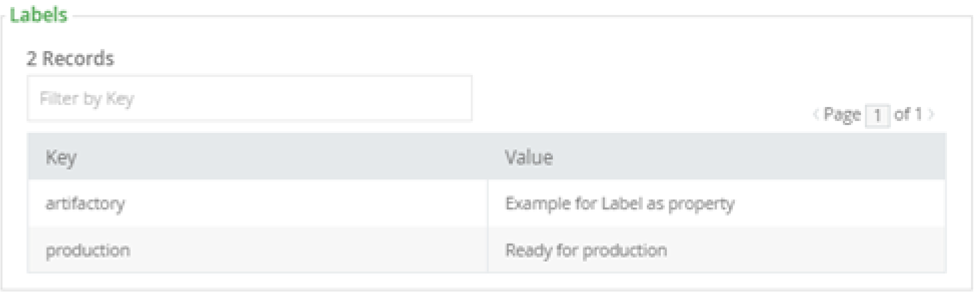
ADD file:c295b0748bf05d4527f500b62ff269bfd0037f7515f1375d2ee474b8 30bad382 in /

2015-12-08 18:31:50

1.1 MB

Labels

This section displays the labels attached to the image.



Labels

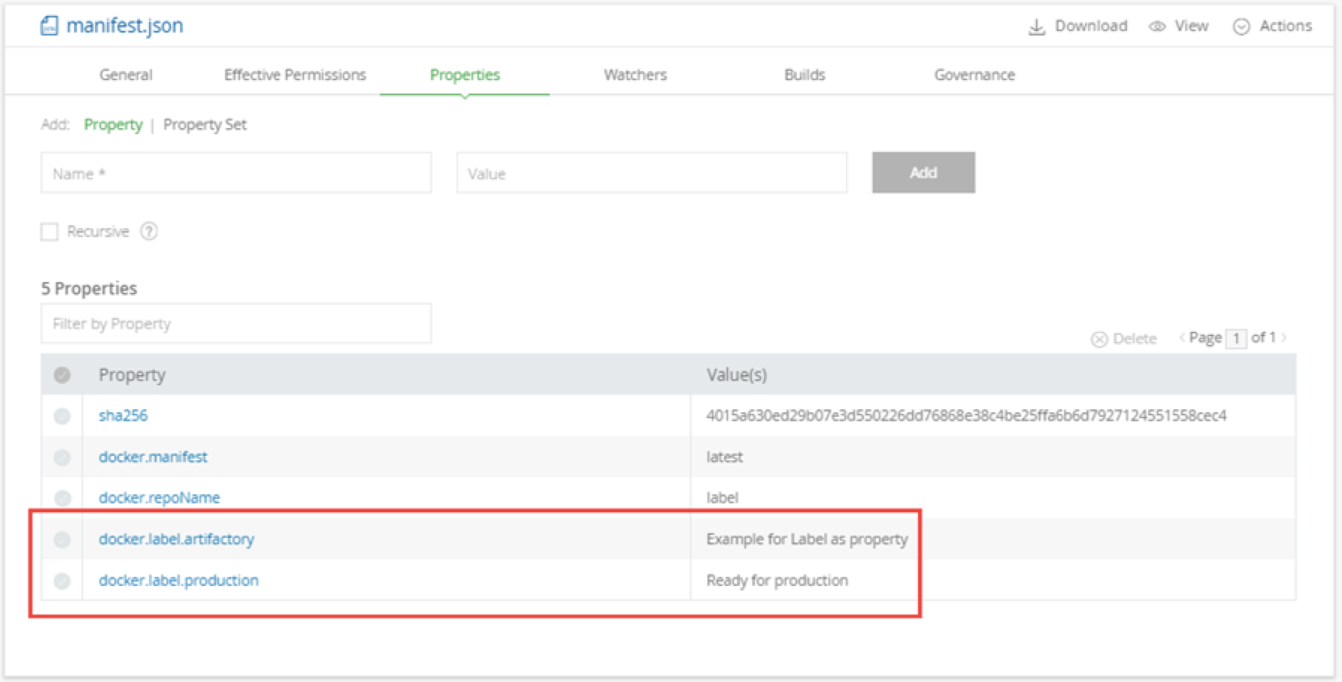
2 Records

Filter by Key

‹ Page 1 of 1 ›

Key	Value
artifactory	Example for Label as property
production	Ready for production

Note also, that from version 4.4.0, 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.

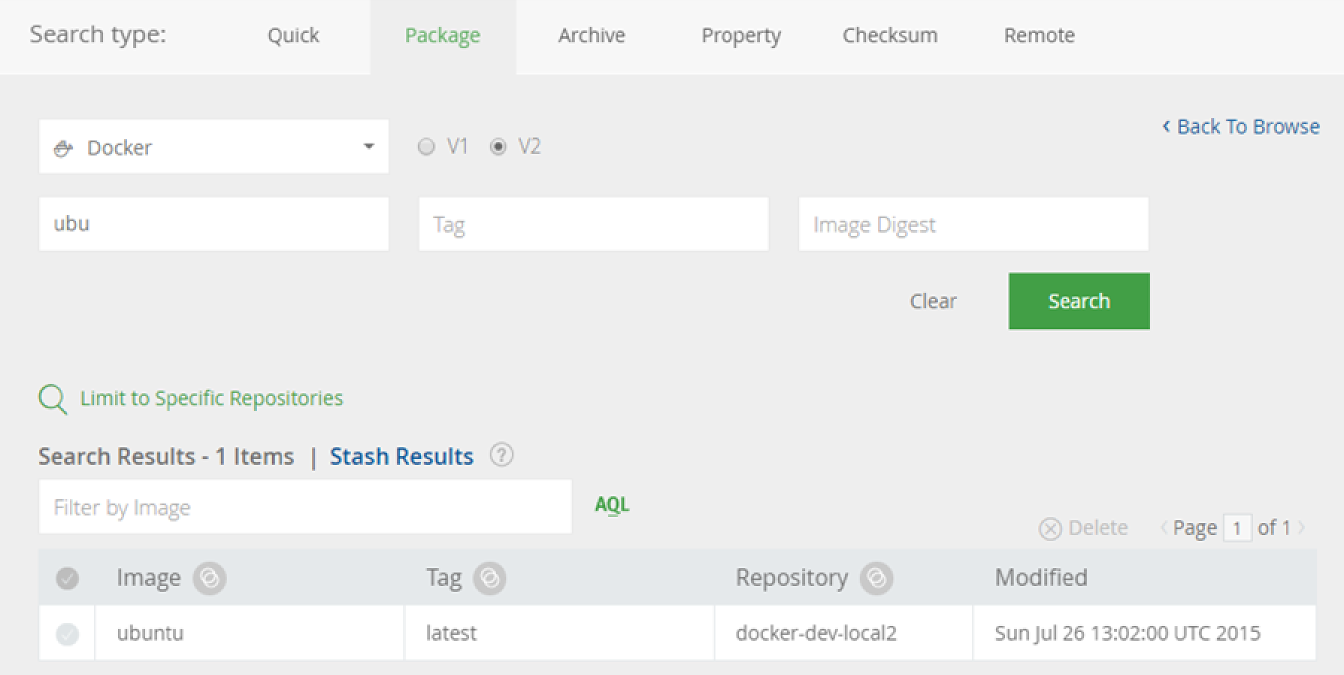


5 Properties

Property	Value(s)
sha256	4015a630ed29b07e3d550226dd76868e38c4be25ffa6b6d7927124551558cec4
docker.manifest	latest
docker.repoName	label
docker.label.artifactory	Example for Label as property
docker.label.production	Ready for production

Searching for Docker Images

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



ubu

Search Results - 1 Items

Image	Tag	Repository	Modified
ubuntu	latest	docker-dev-local2	Sun Jul 26 13:02:00 UTC 2015

Listing Docker Images

From version 4.4.3, 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.

From version 5.4.6, Artifactory also supports pagination for this endpoint.

From version 6.8, 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.

Pushing Images to Bintray

Through Artifactory's close integration with JFrog Bintray, you can push Docker images from your Artifactory Docker Registries directly to Bintray. To enable this, make sure your Bintray credentials are properly configured in your User Profile page.

To push an image to Bintray, use the Distribution Repository.

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 Building 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.

Support Matrix

This matrix provides information on features supported as the versions of Artifactory progress.

Artifactory Version	Docker Client Version	Docker V1 API	Docker V2 API	Remote Repositories*	Virtual Repositories*
4.9+	1.12	✓	✓	✓	✓
4.8+	1.11	✓	✓	✓	✓
4.4.3+	1.10	✓	✓	✓	✓
4.1+	1.8+	✓	✓	✓	✓
4.0.2+	1.8	✓	✓	✓	✗
4.0.0+	1.6+	✓	✓	✓	✗
4.0.0+	<1.6	✓	✗	✗	✗

* Supported for Docker V2 API only