Docker Registry

Table of Contents

Available in Nexus Repository OSS and Nexus Repository Pro

Docker containers and their usage have revolutionized the way applications and the underlying operating system are packaged and deployed to development, testing and production systems.

The creation of the Open Container Initiative[1], and the involvement of a large number of stakeholders, guarantees that the ecosystem of tools around the lightweight containers and their usage will continue to flourish.

Docker Hub[2] is the original registry for Docker container images and it is being joined by more and more other publicly available registries such as the Google Container Registry[3] and others.

Nexus Repository Manager Pro and Nexus Repository Manager OSS support Docker registries as the Docker repository format for hosted and proxy repositories. You can expose these repositories to the client-side tools directly or as a repository group, which is a repository that merges and exposes the contents of multiple repositories in one convenient URL. This allows you to reduce time and bandwidth usage for accessing Docker images in a registry as well as share your images within your organization in a hosted repository. Users can then launch containers based on those images, resulting in a completely private Docker registry with all the features available in the repository manager.

Docker Manifest Lists **NEW IN 3.22**

Docker manifest lists allow a manifest to represent support for multiple architectures while maintaining a single "image:tag" reference format. See https://docs.docker.com/registry/spec/manifest-v2-2/ for more information on docker's schema specification.

The minimum version of Docker that works with Nexus Repository is version 1.8. But please note that Docker is a fast moving project and requires usage of current operating system versions and tools. For example, usage of Red Hat Enterprise Linux 6 is simply not supported. Please use the official documentation[4] as reference and help for your usage.

Topics in this section:

1 https://www.opencontainers.org/
2 https://hub.docker.com/
3 https://cloud.google.com/container-registry/
4 https://docs.docker.com/

SSL and Repository Connector Configuration

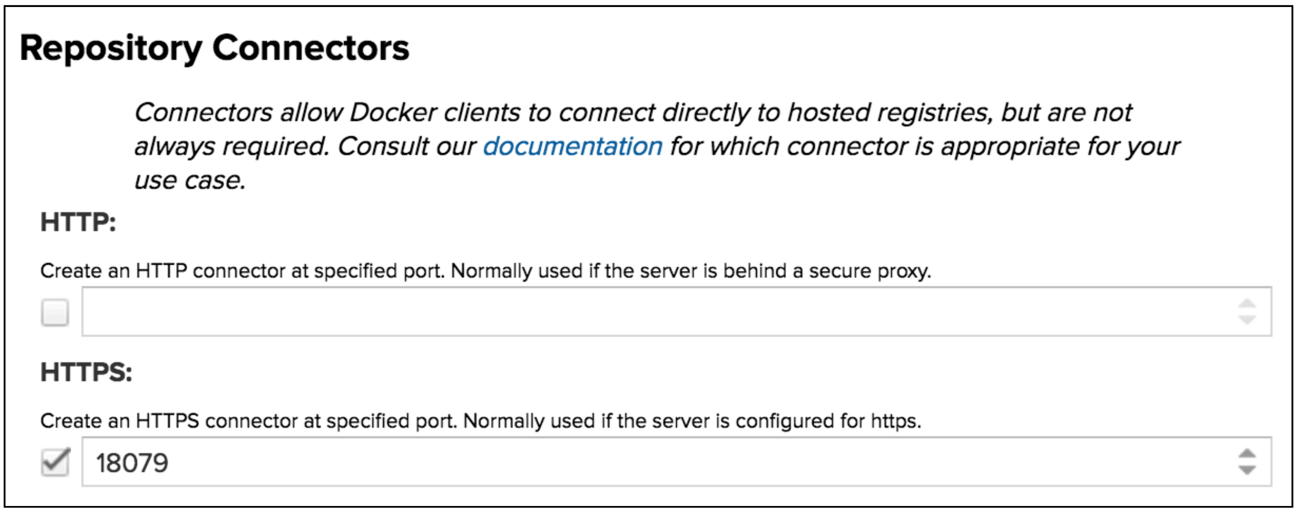
Docker relies on secure connections using SSL to connect to the repositories. You are therefore required to expose the repository manager to your client tools via HTTPS. This can be configured via an external proxy server, which can also be used to scale your repositories(see page 21), or directly with the repository manager. Further details can be found in Inbound SSL - Configuring to Serve Content via HTTPS[5].

Interaction of the `docker` client with repositories requires specific ports to be used. These can be configured in the repository configuration in the Repository Connectors section. In order for this to work on your network, you need to ensure that the chosen ports are available in your organization and not used by some other application, and that no firewall or other network configuration prevents connectivity.

The `docker` client does not allow a context as part of the path to a registry, as the namespace and image name are embedded in the URLs it uses. This is why requests to repositories on the repository manager are served on a specific and separate port from the rest of the application instead of how most other repositories serve content via a path i.e. `<nexus-hostname>/<repositoryName>/<path to content>`.

The recommended minimal configuration requires one port for a Docker repository group used for read access to all repositories and one port for each hosted Docker repository that will receive push events from your users. The Repository Connectors configuration, displayed in *Figure: "Repository Connector Configuration"*, is available in the configuration for proxy and hosted Docker repositories as well as Docker repository groups.

5 https://help.sonatype.com/display/NXRM3/Configuring+SSL#ConfiguringSSL-InboundSSL-ConfiguringtoServeContentviaHTTPS


Repository Connectors

Connectors allow Docker clients to connect directly to hosted registries, but are not always required. Consult our documentation for which connector is appropriate for your use case.

HTTP:

Create an HTTP connector at specified port. Normally used if the server is behind a secure proxy.

[ ]

HTTPS:

Create an HTTPS connector at specified port. Normally used if the server is configured for https.

[✓] 18079

Figure: Repository Connector Configuration

If you have configured the repository manager to use HTTPS directly, you have to configure a HTTPS repository connector. If an external proxy server translates incoming HTTPS requests to HTTP and forwards the request to the repository manager via HTTP you have to configure the respective HTTP port.

A configured context-path for the user interface does not affect the repository connector URLs used by Docker. E.g. if your repository manager instance is configured to be available at `http://localhost:8081/nexus` instead of the default root context `http://localhost:8081/`, the URLs for your Docker repositories will still only use the configured port for the repository and omit the context path in the URL. This is a side-effect of the the fact that Docker does not support context paths in the registry API.

Tips for SSL Certificate Usage

Nexus Repository Manager is not configured with HTTPS connectors by default as it requires an SSL certificate to be generated and configured manually.

The requirement of Docker to use HTTPS forces the usage of SSL certificates. By default, Docker looks up the validity of the certificate by checking with certificate authorities. If you purchased a certificate that is registered with these authorities, all functionality works as desired.

If you create a certificate yourself with tools such as `openssl`, it is self-signed and not registered. Using a self-signed certificate requires further configuration steps to ensure that Docker can explicitly trust it.

Docker Daemon can stand up instances with the `--insecure-registry` flag to skip validation of a self-signed certificate. But the repository manager does not support the use of the flag, as it generates known bugs and other implementation issues.

To generate a trustworthy self-signed certificate for the repository manager use `keytool`, a utility that lets you manage your own private key pairs and certificates. See our knowledge base article[6] to learn how to configure the utility.

Support for Docker Registry API

The Docker client tools interact with a repository via the registry API. It is available in version 1 (V1) and version 2 (V2). The newer V2 will completely replace the old V1 in the future. Currently Docker Hub and other registries as well as other tools use V2, but in many cases fall back to V1. E.g., search is currently only implemented in V1.

Nexus Repository Manager supports V1 as well as V2 of the API. All Docker repository configurations contain a section to configure Docker Registry API Support . If you activate Enable Docker V1 API for a repository it is enabled to use V1 as a fallback from V2. Without this option any V1 requests result in errors from the client tool.

Generally V1 support is only needed for repository groups that will be used for command line-based searches, when any client side tools in use require V1 or when a upstream proxy repository requires V1. If you are unsure if your setup uses these or V1, it is recommended to activate V1 support as there should be no harm if it is not needed.

Proxy Repository for Docker

Docker Hub is the common registry used by all image creators and consumers. To reduce duplicate downloads and improve download speeds for your developers and CI servers, you should proxy Docker Hub and any other registry you use for Docker images.

To proxy a Docker registry, you simply create a new *docker (proxy)* as documented in Repository Management[7].

Minimal configuration steps are:

- Define *Name*
- Define URL for *Remote* storage

---

6 https://support.sonatype.com/hc/en-us/articles/217542177
7 https://help.sonatype.com/display/NXRM3/Repository+Management

- Enable *Docker V1 API* support, if required by the remote repository
- Select correct *Docker index*, further configure *Location of Docker* index if needed
- Select *Blob store* for Storage

If you intend users to access the proxy repository directly, you may wish to configure a Repository Connector as explained in SSL and Repository Connector Configuration(see page 4). This is not necessary for proxy content that is accessed via a Docker group.

The *Remote Storage* has to be set to the URL of the remote registry you want to proxy. The configuration for proxying Docker Hub uses the URL `https://registry-1.docker.io` for the Remote storage URL.

The Proxy configuration for a Docker proxy repository includes a configuration URL to access the *Docker Index*. The index is used for requests related to searches, users, docker tokens and other aspects. The registry and the index are typically co-hosted by the same provider, but can use different URLs. E.g. the index for Docker Hub is exposed at `https://index.docker.io/`. The default option of *Use proxy registry (specified above)* will attempt to retrieve any index data from the same URL configured as the Remote storage URL. The option to *Use Docker Hub* fulfills any index related requests by querying the Docker Hub index at `https://index.docker.io/`. This configuration is desired when the proxy repository is Docker Hub itself or any of its mirrors. The option to use a *Custom index* allows you to specify the URL of the index for the remote repository. It is important to configure a correct pair of Remote Storage URL and Docker Index URL. In case of a mismatch, search results potentially do not reflect the content of the remote repository and other problems can occur.

Just to recap, in order to configure a proxy for Docker Hub you configure the Remote Storage URL to `https://registry-1.docker.io`, enable Docker V1 API support and for the choice of Docker Index select the *Use Docker Hub* option.

Hosted Repository for Docker (Private Registry for Docker)

A hosted repository using the Docker repository format is typically called a private Docker registry. It can be used to upload your own container images as well as third-party images. It is common practice to create two separate hosted repositories for these purposes.

To create a Docker hosted repository, simply create a new *docker (hosted)* repository as documented in Repository Management[8].

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for Storage

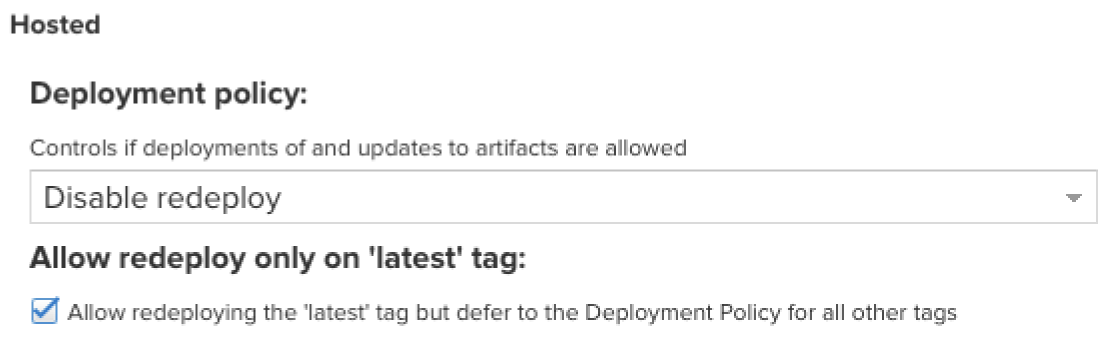
If you add a Repository Connectors configuration as documented in SSL and Repository Connector Configuration(see page 4) you can `push` images to this repository, and subsequently access them directly from the hosted repository or ideally from the Docker repository group as documented in Grouping Docker Repositories(see page 8).

8 https://help.sonatype.com/display/NXRM3/Repository+Management

Deployment Policy

Docker repositories support all the regular options for hosted repository deployment policies.

An additional configuration option named *Allow redeploy only on 'latest' tag* (**NEW IN 3.21**) appears when the *Deployment Policy* is set to *Disable redeploy* .


Hosted

Deployment policy:

Controls if deployments of and updates to artifacts are allowed

Disable redeploy

Allow redeploy only on 'latest' tag:

✓ Allow redeploying the 'latest' tag but defer to the Deployment Policy for all other tags

When enabled this allows pushing a new docker 'latest' tag for an existing image of the same name.

Grouping Docker Repositories

Available in Nexus Repository OSS and Nexus Repository Pro

A repository group is the recommended way to expose all your repositories for read access to your users. It allows you to pull images from all repositories in the group without needing any further client side configuration after the initial setup. A repository group allows you to expose the aggregated content of multiple proxy and hosted repositories with one URL to your tools.

To create a Docker repository group, simply create a new *docker (group)* repository as documented in Repository Management[9].

Minimal configuration steps are:

- Define *Name*
- Select *Blob store* for Storage
- Add Docker repositories to the *Members* list in the desired order

Typically the member list includes a mixture of proxy and hosted repositories to allow access to public as well as private images.

9 https://help.sonatype.com/display/NXRM3/Repository+Management

Using the Repository Connectors port of the repository group and the URL of the repository manager in your client tool gives you access to the container images in all repositories from the group. Any new images added as well as any new repositories added to the group will automatically be available.

Check out this repository configuration demonstrated in this video[10] .

NEW IN 3.27

Using a Pro version, you can push images to the group repository as documented in Pushing Images(see page 19).

Content Selectors and Docker

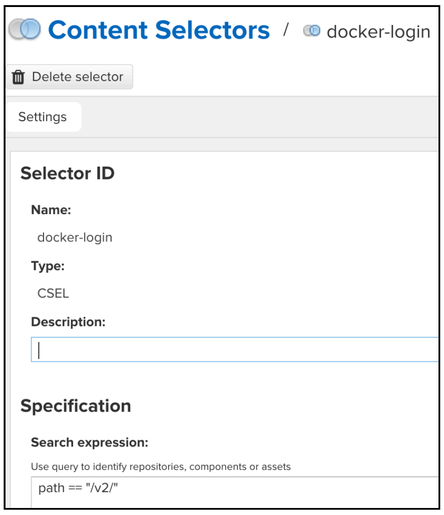
Administrators of the Docker format may desire to not share all Docker components across all teams. In general, Content Selectors[11] provide you with a way to select specific content from a whole and, in conjunction with Access Control[12] (specifically the privileges created by Content Selectors), are the recommended way to manage this desire. However, the Docker format has some non-standard complexity to setting up security against the CLI and UI which is communicated here.

A basic assumption this page makes is that you are able to sort the security of your docker images by name. Thus below we use the word `team` in our examples where `team` is the representation of that which you are sorting. For example, it might be `sonatype/nexus` for this product/project team, or further `sonatype/nexus/docker` for the docker specific parts. If you are unable to do this, this page likely will not help you or your solution will be much more complex than the scenarios given.

docker login

This command requires access to the /v2/ path and can be established by creating a content selector with the search expression `path == "/v2/"` as shown here:

10 https://www.youtube.com/watch?v=oxCztw5MfAw
11 https://help.sonatype.com/display/NXRM3/Repository+Management#RepositoryManagement-ContentSelectors
12 https://help.sonatype.com/display/NXRM3/Access+Control



Selector ID

Name:
docker-login

Type:
CSEL

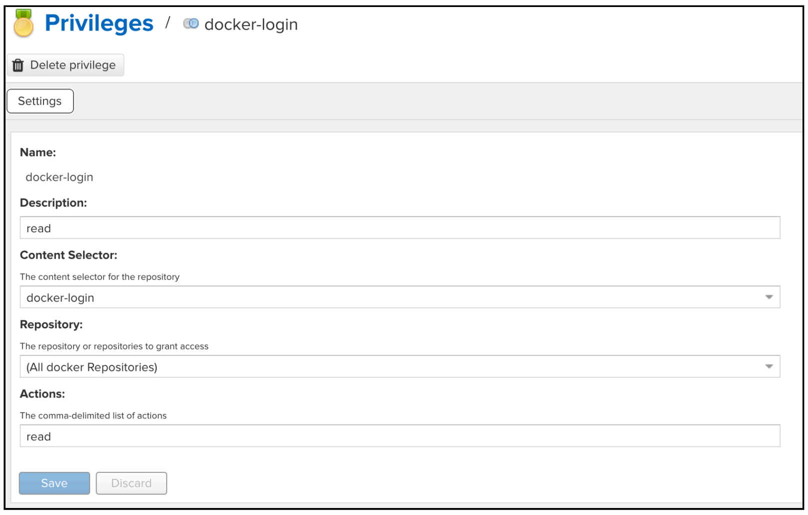
Description:

Specification

Search expression:
Use query to identify repositories, components or assets

```
path == "/v2/"
```

Once the content selector is created, a *Repository Content Selector* privilege with *Actions* defined with `read` should be created, as shown below, then assigned to a role and subsequently users.



Name:
docker-login

Description:
read

Content Selector:
The content selector for the repository
docker-login

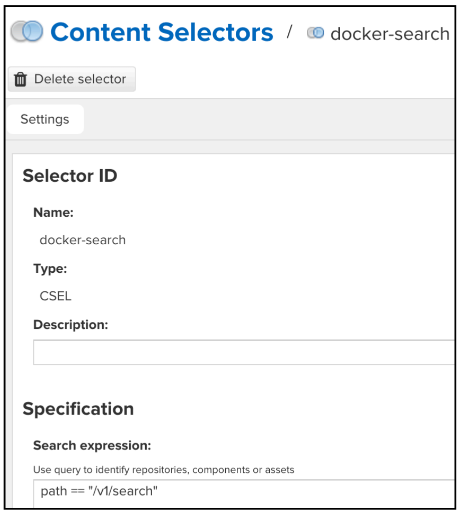
Repository:
The repository or repositories to grant access
(All docker Repositories)

Actions:
The comma-delimited list of actions
read

Access to login likely will need to be combined with other privileges to be useful.

docker search

This command requires access to the /v1/search path and can be established by creating a content selector with the search expression `path == "/v1/search"` as shown here:


Content Selectors / docker-search

Delete selector

Settings

Selector ID

Name:
docker-search

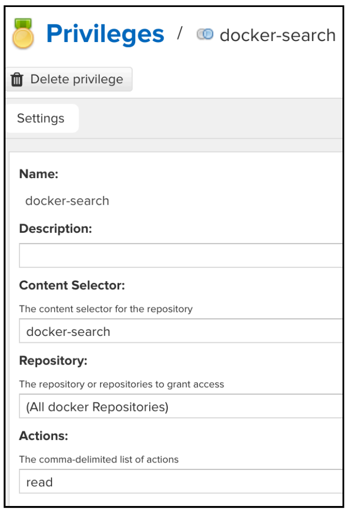
Type:
CSEL

Description:

Specification

Search expression:
Use query to identify repositories, components or assets
path == "/v1/search"

Once the content selector is created, a *Repository Content Selector* privilege with *Actions* defined as `read` sho uld be created, as shown below, then assigned to a role and subsequently users.


Privileges / docker-search

Delete privilege

Settings

Name:
docker-search

Description:

Content Selector:
The content selector for the repository
docker-search

Repository:
The repository or repositories to grant access
(All docker Repositories)

Actions:
The comma-delimited list of actions
read

Disclaimers

This privilege gives those assigned access to search the entire repository which can be more than you have access to pull/push.

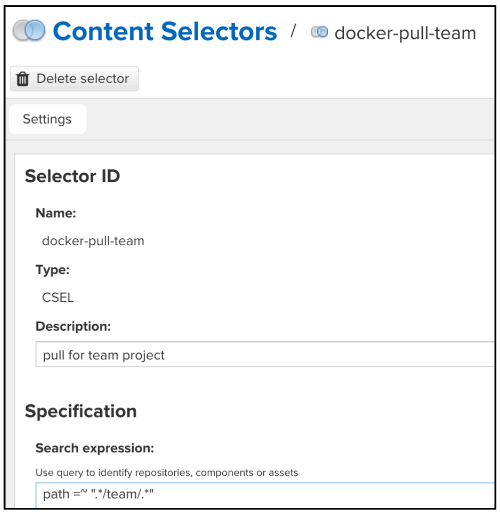
Further, this privilege does not give you access to search anything via UI, instead it is delegating search via the CLI. How to give permission to search via the UI is defined below(see page 14).

docker pull (by team)

This command requires that the content selector path be restricted by `team`. This can be done loosely (e.g. `path =~ ".*/team/.*"`) with "contains team" or strictly (e.g. `path =^ "/v2/team/"`) with "starts with team". Note that for simpler docker packages, you can restrict using `library` too (e.g. `path =^ "/v2/library/hello-world/"`).

Any of these methods give you the ability to pull any version of components from `team`.

Here's an example content selector:



Selector ID

Name:
docker-pull-team

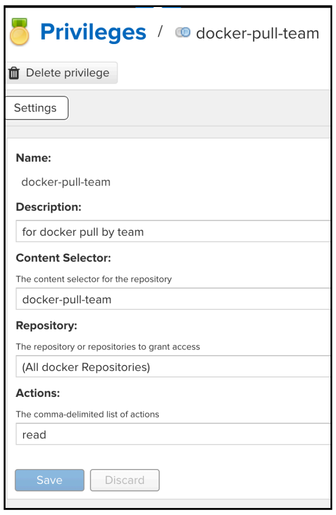
Type:
CSEL

Description:
pull for team project

Specification

Search expression:
Use query to identify repositories, components or assets
path =~ ".*/team/.*"

Once the content selector is created, a *Repository Content Selector* privilege with *Actions* defined as `read` sho uld be created, as shown below, then assigned to a role and subsequently users.



Name:
docker-pull-team

Description:
for docker pull by team

Content Selector:
The content selector for the repository
docker-pull-team

Repository:
The repository or repositories to grant access
(All docker Repositories)

Actions:
The comma-delimited list of actions
read

Content selectors and privileges will need created for each "team".

With security setup in this way, users trying docker pull commands by hash will get unauthorized responses because the hash does not fall in the pattern of the path defined above.

Tip: Browse UI and Content Selector Preview

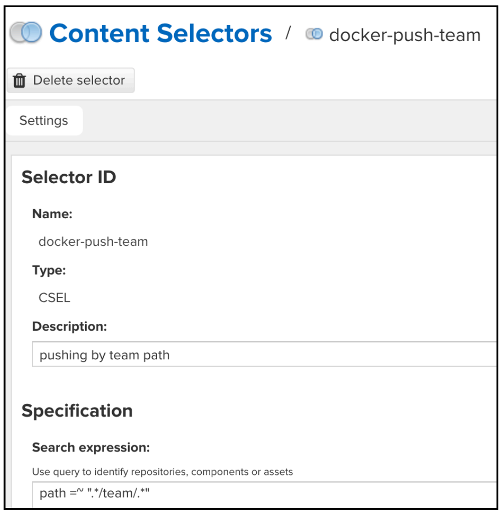
`read` action grants permission to the Browse UI but only to manifests and tags (which match the path defined). Individual layers will not be shown.

The same will reflect in Content Selector preview results.

docker push (by team)

Like *docker pull*, this command also requires that the content selector path be restricted by `team`. This can be done loosely (e.g. `path =~ ".*/team/.*"`) with "contains team" or strictly (e.g. `path =^ "/v2/team/"`) with "starts with team". Using the simpler `library` method from *docker pull* (e.g. `path =^ "/v2/library/hello-world/"`) will not work.

Because the exact same paths can be used if you have *docker pull* content selectors too, you can reuse them if you've implemented *docker pull*. However you need them distinct, you'd create a new content selector like this:



Selector ID

Name:
docker-push-team

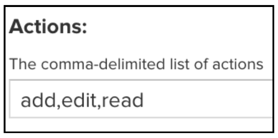
Type:
CSEL

Description:
pushing by team path

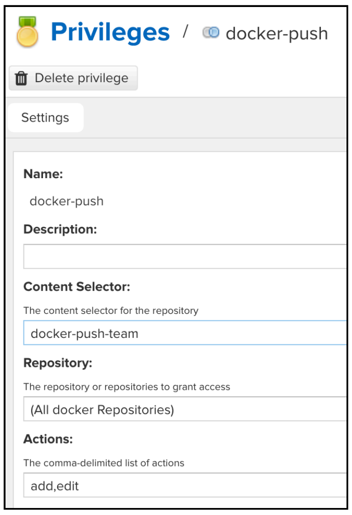
Specification

Search expression:
Use query to identify repositories, components or assets
path =~ ".*/team/.*"

If you are combining permissions, edit your existing privilege *Actions* to put (comma delinated) `add` and `edit` (in addition to `read`) like so:


Actions:
The comma-delimited list of actions
add,edit,read

If you are not combining, create a new a *Repository Content Selector* privilege with *Actions* defined as `add,edit` like so:



Privileges / docker-push

Delete privilege

Settings

Name: docker-push

Description:

Content Selector: The content selector for the repository: docker-push-team

Repository: The repository or repositories to grant access: (All docker Repositories)

Actions: The comma-delimited list of actions: add,edit

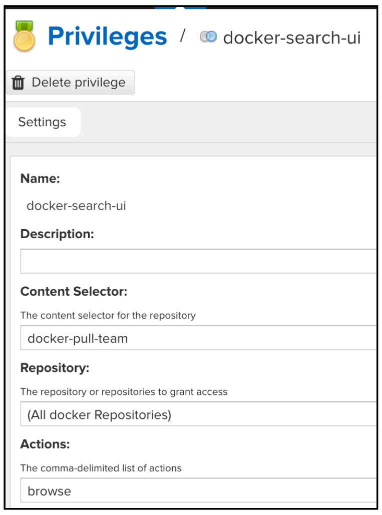
Then assign the privilege to roles and subsequently users.

Content selectors and privileges will need created (or combined/edited) for each "team".

Search UI

The permissions described in Docker Search(see page 10) do not grant permission to the *Search* UI. To gain permission to the *Search* UI the role needs to have the *nx-search-read* privilege assigned. In addition to this to see the items restricted by path, the Action of `browse` needs to be added. You can do this by adding (comma delinated) to an existing privilege with the path for your team (e.g. `path =^ "/v2/team/"`, `path =~ ".*/team/.*"` or `path =^ "/v2/library/hello-world/"`) or by creating a new privilege then assigning that to roles.

Here's an example of the new privilege you would create:



Privileges / docker-search-ui

Delete privilege

Settings

Name: docker-search-ui

Description:

Content Selector: The content selector for the repository: docker-pull-team

Repository: The repository or repositories to grant access: (All docker Repositories)

Actions: The comma-delimited list of actions: browse

If creating a new privilege, you would assign this privilege to roles and then the roles to users as needed. If editing the existing privilege, permissions will be granted when you save.

Docker Authentication

Authenticated Access to Docker Repositories

Configuration Guidance

In some cases in order to `docker login` and in order to access docker anonymously, you also need to enable the *Docker Bearer Token Realm* as generally outlined in Realms[13]. This realm is inactive by default.

If access to a repository requires the user to be authenticated, `docker` will check for authentication access in the `.docker/config.json` file. If authentication is not found, some actions will prompt for authentication but otherwise a docker login command will be required before the actions can be performed. Typically this is required when anonymous access to the repository manager is disabled or the operation requires authentication.

The `docker login` command observes the following syntax for the desired repository or repository group:

```
docker login <nexus-hostname>:<repository-port>
```

Provide your repository manager credentials of username and password as well as an email address. This authentication is persisted in `~/.docker/config.json` and reused for any subsequent interactions against that repository. Individual login operations must be performed for each repository and repository group you want to access in an authenticated manner.

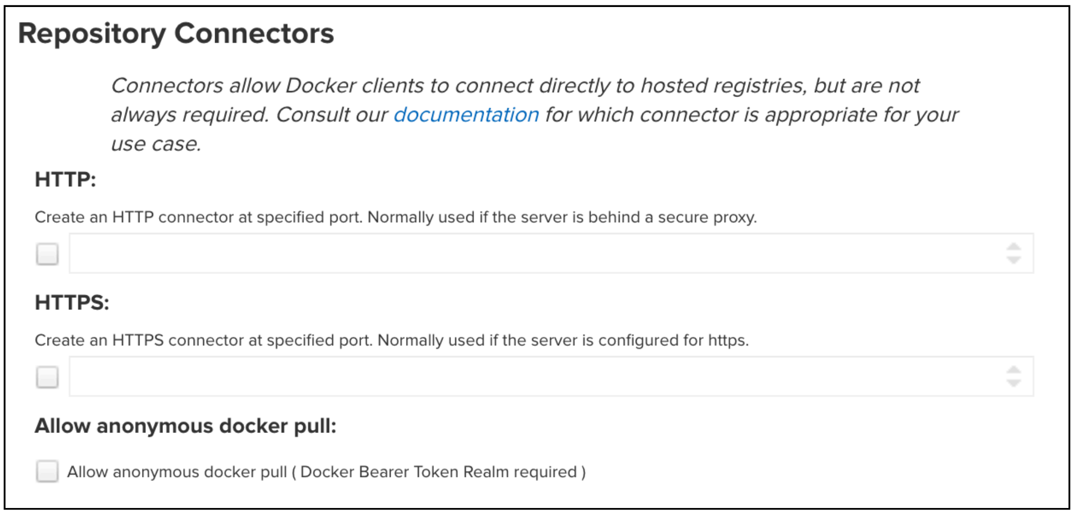
Specifically when planning to push to a repository a preemptive login operation is advisable as it removes the need for use interaction and is therefore suitable for continuous integration server setups and automated scenarios.

Unauthenticated Access to Docker Repositories

By default when using Nexus Repository Manager, all docker repositories require authentication to be read from using the command line tools regardless of any permissions granted by the Anonymous[14] user (if enabled) or, in the case of proxy repositories, the remotes' settings. For Docker in NXRM, this can be bypassed on a per repository basis by editing the repository settings and enabling the *Allow anonymous*

13 https://help.sonatype.com/display/NXRM3/User+Authentication#UserAuthentication-security-realms
14 https://help.sonatype.com/display/NXRM3/Anonymous+Access

docker pull checkbox under the *Repository Connectors* section shown at the bottom of *Figure: "Repository Connectors Configuration including Allow anonymous docker pull"*.



Repository Connectors

Connectors allow Docker clients to connect directly to hosted registries, but are not always required. Consult our documentation for which connector is appropriate for your use case.

HTTP:

Create an HTTP connector at specified port. Normally used if the server is behind a secure proxy.

HTTPS:

Create an HTTPS connector at specified port. Normally used if the server is configured for https.

Allow anonymous docker pull:

Allow anonymous docker pull (Docker Bearer Token Realm required)

Figure: Repository Connectors Configuration including Allow anonymous docker pull

The Anonymous[15] user must be enabled and granted read access to the docker repositories.

Each repository must have the *Allow anonymous docker pull* configuration enabled individually. Enabling this for a group, just allows the anonymous read when utilizing the group connector. If you utilize one of the member connectors, it will use whatever setting it has for that member even if it differs from the group.

Only read settings are affected by this configuration and all other actions on the docker repositories require authentication or lack thereof regardless if this option is on or off.

Accessing Repositories

You can browse Docker repositories in the user interface and inspect the components and assets and their details as documented in Browsing Repositories and Repository Groups[16].

When using the docker command line client, or any other tools using the repository manager indirectly, the common structure for commands can be:

```
docker <command> <nexus-hostname>:<https-repository-port>/<namespace>/<image>:<tag>
docker search <nexus-hostname>:<https-repository-port>/<search-term>
```

15 https://help.sonatype.com/display/NXRM3/Anonymous+Access
16 https://help.sonatype.com/display/NXRM3/Browsing+Repositories+and+Repository+Groups

with

command

a docker command such as `push` or `pull`

nexus-hostname

the IP number or hostname of your repository manager

https-repository-port

https port which ultimately directs to the NXRM docker registry https port in the repository connector config, or that of a reverse proxy https connector which proxies the docker registry plain http port

namespace

the optional namespace of the specific image reflecting the owner, if left out this will silently default to */library* and utilize Docker Hub

image

the name of the Docker image

tag

the optional tag of the image, defaulting to *latest* when omitted

search-term

the search term or name of the image to search for

The most important aspects are to know and use the correct hostname for the repository manager and the port for the desired repository or repository group.

Searching

Searching for Docker images can be performed in the user interface as described in Searching for Components[17]. This search will find all Docker images that are currently stored in repositories, either because they have been pushed to a hosted repository or they have been proxied from an upstream repository and cached in the repository manager.

The more common use case for a Docker user is to search for images on the command line:

```
$ docker search postgres
NAME      DESCRIPTION                                  STARS  OFFICIAL  AUTOMATED
postgres  The PostgreSQL object-relational database... 1025   [OK]      ...
```

By default this search uses Docker Hub as preconfigured in `docker` and will only find images available there. A more powerful search is provided by the repository manager when searching against a repository group.

17 https://help.sonatype.com/display/NXRM3/Searching+for+Components

An example looking for a `postgres` image on Nexus Repository Manager OSS running on the host `nexus.example.com` and exposing a repository group with a repository connector port of `18443` looks like this:

```
docker search nexus.example.com:18443/postgres
```

The results include all images found in the repositories that are part of the repository group. This includes any private images you have pushed to your hosted repositories. In addition it includes all results returned from the remote repositories configured as proxy repositories in the group. Searching in a specific repository can be achieved by using the repository connector port for the specific repository.

> Searching can be done from the command line anonymously by setting up your individual repositories as described in the earlier subsection(see page 0).

Pulling Images

Downloading images, also known as pulling, from the repository manager can be performed with the `docker pull` command. The only necessary additions are the hostname or IP address of the repository manager as well as the repository connector port for the repository or repository group to download from:

```
docker pull <nexus-hostname>:<repository-port>/<image>
```

The preferred setup is to proxy all relevant sources of public/private images you want to use, with Docker Hub being the most common choice. Then configure one or more hosted repositories to contain your own images, and expose these repositories through one repository group.

Examples for various images from Nexus Repository Manager running on the host `nexus.example.com` and exposing a repository group with a repository connector port of `18443` are:

```
docker pull nexus.example.com:18443/ubuntu
docker pull nexus.example.com:18443/bitnami/node
docker pull nexus.example.com:18443/postgres:9.4
```

These snippets download the official ubuntu image, the node image from the user bitnami and the version 9.4 of the postgres image. Official images such as ubuntu or postgres belong to the library user on Docker Hub and will therefore show up as `library/ubuntu` and `library/postgres` in the repository manager.

After a successful `pull` you can start the container with `run`.

Pulling can be configured to be done from the command line anonymously by setting up your individual repositories as described in the earlier subsection[18].

Pushing Images

Available in Nexus Repository OSS and Nexus Repository Pro

Sharing an image can be achieved by publishing it to a hosted repository. This is completely private and requires you to `tag` and `push` the image. When tagging an image, you can use the image identifier (`imageId`). It is listed when showing the list of all images with `docker images`. Syntax and an example (using `imageId`) for creating a tag are:

```
docker tag <imageId or imageName> <nexus-hostname>:<repository-port>/<image>:<tag>
docker tag af340544ed62 nexus.example.com:18444/hello-world:mytag
```

Once the tag, which can be equivalent to a version, is created successfully, you can confirm its creation with `docker images` and issue the `push` with the syntax:

```
docker push <nexus-hostname>:<repository-port>/<image>:<tag>
```

Note that the port needs to be the repository connector port configured for the hosted repository to which you want to push to. You can not push to a proxy repository.

A sample output could look like this:

```
$ docker push nexus.example.com:18444/hello-world:labeltest
The push refers to a repository [nexus.example.com:18444/hello-world] (len: 1)
Sending image list
Pushing repository nexus.example.com:18444/hello-world (1 tags)
535020c3e8ad: Image successfully pushed
af340544ed62: Image successfully pushed
Pushing tag for rev [af340544ed62] on
{https://nexus.example.com:18444/repository/docker-internal/v1/repositories/hello-world/tags/labeltest}
```

18 https://help.sonatype.com/display/NXRM3M/Authentication#Authentication-AnonymousReadAccess

Now, this updated image is available in the repository manager and can be pulled by anyone with access to the repository, or the repository group, containing the image. Pulling the image from the repository group exposed at port `18443` can be done with:

```
docker pull nexus.example.com:18443/hello-world:labeltest
```

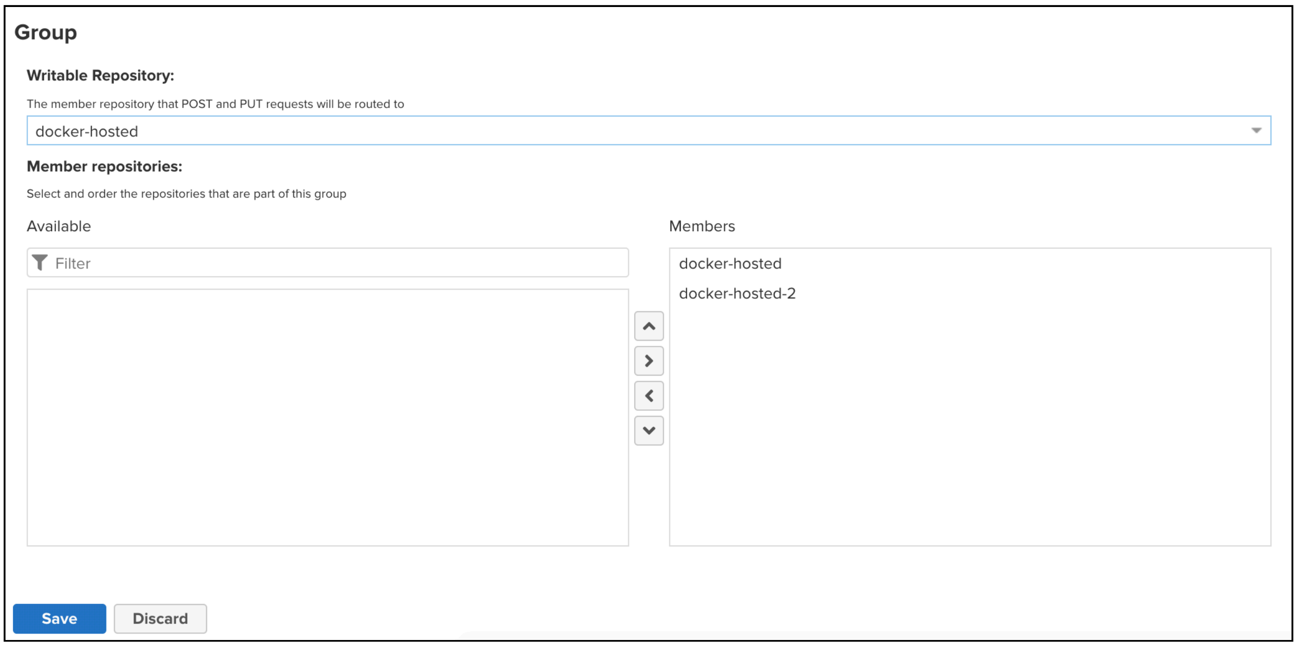
Prior to `push`, and depending on your configuration, repository manager login credentials may be required before a `push` or `pull` can occur.

Searching, Browsing, Pushing and Pulling are all showcased in this video[19].

Pushing large images can result in failures due to network interruptions and other issues. These partial uploads result in temporary storage for these transfers in the repository manager filling up. The task Purge incomplete docker uploads can be configured to delete these files. If you also tend to upload images to the same tag repeatedly, this can leave a lot of dangling images around, consuming a lot of space. The task *Docker - Delete unused manifests and images* can be configured to remove these files. Further information about these tasks can be found in Configuring and Executing Tasks[20].

NEW IN 3.27

Using a **Pro** version, you can push images to the group repository. In that case, you have to select a *Writable Repository* in the configuration menu.



Group

Writable Repository:

The member repository that POST and PUT requests will be routed to

docker-hosted

Member repositories:

Select and order the repositories that are part of this group

Available	Members
	docker-hosted
	docker-hosted-2

19 https://www.youtube.com/watch?v=Z2jH9LgeeI8
20 https://help.sonatype.com/display/NXRM3/Nodes

The procedure of pushing images is otherwise the same as for a hosted repository.

When pushing to a group repository, existing layers of all members are checked to avoid pushing those layers which will save on storage and bandwidth.

For example, if you try to push the `hello-world:latest` image which was pulled from the proxy repository you will get the output:

```
9c27e219663c: Layer already exists
```

Scaling Repositories

When there is a requirement to have many Docker registries it potentially leads to configuring many repositories using many different connector ports. These connector ports are setup within NXRM and use the connector thread allocation logic of the underlying Eclipse Jetty. The default thread pool size, that Eclipse Jetty draws from to preallocate threads for the new connector, is 400. Our support article Understanding Eclipse Jetty 9.4.8 Thread Allocation[21] gives really good details on how allocation works and why at certain amount of repositories with connectors the pool can get maxed out and `IllegalStateExceptions` maybe seen indicating that there are `Insufficient configured threads.`

Reverse Proxy (recommended)

Placing a reverse proxy in front of NXRM dynamically maps docker requests to NXRM Docker repositories. Using a reverse proxy does not require port numbers to be specified and therefore no new connectors on your docker repositories are required. There are two strategies that can be used, the *Port Method* and the *Subdomain Method.* In the support article Docker Repository Reverse Proxy Strategies[22] we give more details on these strategies.

Changing the thread pool size

Knowing that there is a limit to the amount of connectors that can be used in conjunction with how threads are being allocated for them from the Eclipse Jetty thread pool gives us a clear indication that we are able to make changes to the configuration of the thread pool. This can be achieved by editing the configuration file ($install-dir/etc/jetty/jetty.xml) and add a new `maxThreads` setter (a restart of NXRM is required to pick up changes to the file)

21 https://support.sonatype.com/hc/en-us/articles/360000744687
22 https://support.sonatype.com/hc/en-us/articles/360000761828-Docker-Repository-Reverse-Proxy-Strategies

```
<Arg name="threadpool">
    <New id="threadpool" class="org.sonatype.nexus.bootstrap.jetty.InstrumentedQueuedThreadPool">
        <Set name="maxThreads">400</Set>
    </New>
</Arg>
```

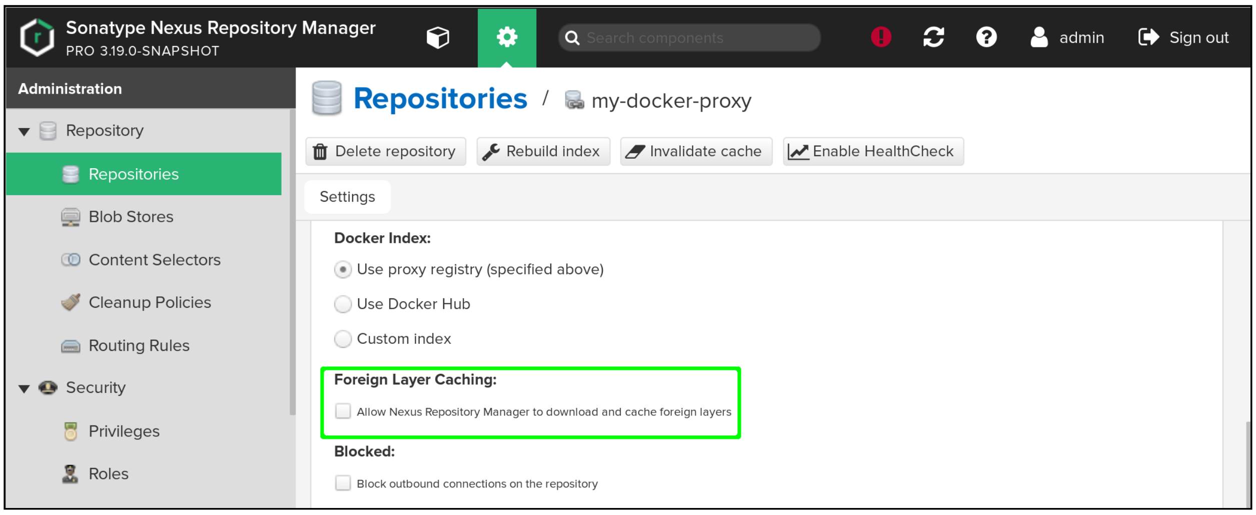
> Changing the thread pool size can help with small scaling efficiencies but is not a sustainable solution for scaling performance and we suggest being conservative increasing the maximum threads as each new thread in the pool has the potential to increase workload inside of NXRM for concurrent request threads.

Foreign Layers

NEW IN 3.19

You can enable NXRM to cache the foreign layers of Docker images so that the client will not need to communicate to a Docker registry other than your proxied Docker repository. This can be useful in air-gapped environments in which client hosts may not have full access to the internet where the layer may not be accessible.

To enable this functionality, in your Docker proxy repository configuration check the *Foreign Layer Caching* option:



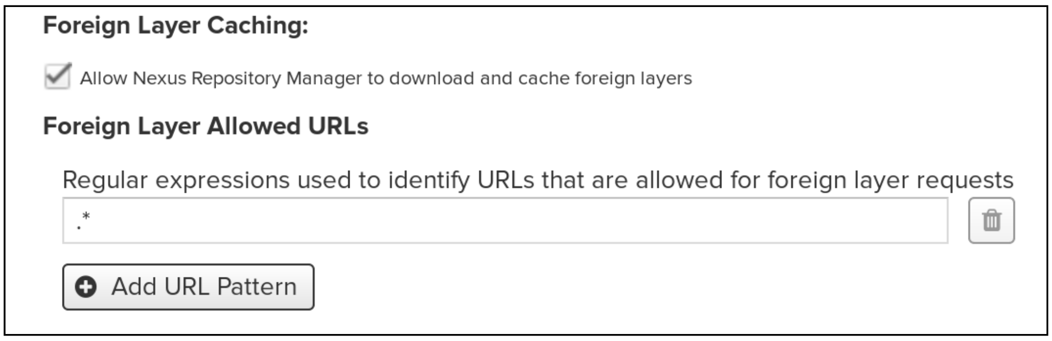
Docker Index:
[✓ Use proxy registry (specified above)]
Use Docker Hub
Custom index

Foreign Layer Caching:
☐ Allow Nexus Repository Manager to download and cache foreign layers

Blocked:
☐ Block outbound connections on the repository

This will expand the area to display the additional configuration of allowed URL patterns. In order to provide fine grained control over where your proxy repository communicates you will need to add a URL pattern (a regular expression) to the configuration. These patterns act as a white list to determine if your proxy repository is allowed to retrieve and cache foreign layers from a particular location. The list is inclusive such that a URL that matches **any** configured pattern will be allowed. When caching is enabled a default pattern of `.*` will be added which will match any URL; modify the pattern as desired, and click *Add URL Pattern* to set additional values. At least one value must be set.

Single, Defaulted Entry allowing all URLs:


Foreign Layer Caching:

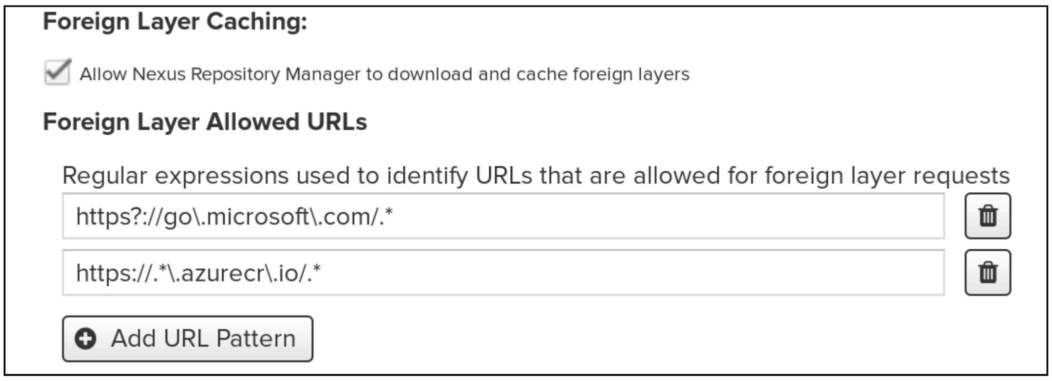
✓ Allow Nexus Repository Manager to download and cache foreign layers

Foreign Layer Allowed URLs

Regular expressions used to identify URLs that are allowed for foreign layer requests

```
.*
```

Multiple entries:


Foreign Layer Caching:

✓ Allow Nexus Repository Manager to download and cache foreign layers

Foreign Layer Allowed URLs

Regular expressions used to identify URLs that are allowed for foreign layer requests

```
https?://go\.microsoft\.com/.*
https://.*\.azurecr\.io/.*
```

For the later example the two patterns would allow foreign layer retrieval from any URL path in the
`go.microsoft.com` domain for either http or https protocols, as well as any URL path for any subdomain of the
`azurecr.io` domain for only the https protocol.

It should be noted that the regular expressions used here must match the URL entirely, not just a portion. The examples accomplish this by adding the

`.*`

to the end of the pattern in order to match any path.

Once setup your docker proxy repository will be able to fetch foreign layer instances, providing a single source of data for the corresponding images.

Docker Content Trust

Docker Content Trust (DCT) allows docker image tags to be cryptographically signed. This allows users to verify the integrity and the publisher of docker data provided by the registry. DCT is enforced at two levels: by the docker client (supported by Docker Community and Enterprise) and by the docker engine (Enterprise only).

Docker Content Trust is not directly handled by NXRM3. You can use Docker Notary in conjuction with NXRM3 to publish and manage trusted Docker content.

Docker Notary Service

Docker Notary can be obtained in binary form the release page on github: https://github.com/theupdateframework/notary/releases. Please refer to the Docker documentation[23] to install and configure Notary.

Client Configuration

Docker Content Trust is configured by setting the following environment variables:

```
export DOCKER_CONTENT_TRUST=1
export DOCKER_CONTENT_TRUST_SERVER=https://<notary-server-hostname>
```

With these variables set, docker will enforce content trust and use your notary service to store trust information.

ⓘ The docker client requires a valid HTTPS certificate. If you use a certificate signed by another certificate authority (a self-signed certificate for example), place the certificate in `$HOME/.docker/tls/<notary-server-hostname>/ca.crt`

With DCT enabled, docker images will be automatically signed on push. A sample output could look like this:

23 https://docs.docker.com/notary/getting_started/

```
$ docker push nexus.example.com:18079/hello-world:test
The push refers to repository [nexus.example.com:18079/hello-world]
test: digest: sha256:92c7f9c92844bbbb5d0a101b22f7c2a7949e40f8ea90c8b3bc396879d95e899a size: 524
Signing and pushing trust metadata
```

Pulling a docker image by tag will automatically validate the signature. For example:

```
$ docker pull nexus.example.com:18079/hello-world:test-unsigned
Error: remote trust data does not exist for nexus.example.com:18079/hello-world: nexus.example.com:18079
 does not have trust data for nexus.example.com:18079/hello-world

$ docker pull nexus.example.com:18079/hello-world:test-unsigned
test: Pulling from hello-world
Digest: sha256:92c7f9c92844bbbb5d0a101b22f7c2a7949e40f8ea90c8b3bc396879d95e899a
Status: Image is up to date for nexus.example.com:18079/hello-world:test
nexus.example.com:18079/hello-world:test
```