# Advanced Topics

## Overview

This page provides some advanced topics for using Docker with Artifactory.

**Page Contents**



## Using a Self-signed SSL Certificate

You can use self-signed SSL certificates with `docker push/pull` commands, however for this to work, you need to specify the `--insecure-registry` daemon flag for each insecure registry.

For full details please refer to the Docker documentation.

For example, if you are running Docker as a service, edit the */etc/default/docker* file, and append the `--insecure-registry` flag with your registry URL to the DOCKER_OPTS variable as in the following example:

**Edit the DOCKER_OPTS variable**

```
DOCKER_OPTS="-H unix:///var/run/docker.sock --insecure-registry artprod.company.com"
```

For this to take effect, you need to restart the Docker service.

If you are using **Boot2Docker**, please refer to the **Boot2Docker** documentation for Insecure Registry.

If you do not make the required modifications to the `--insecure-registry` daemon flag, you should get the following error:

**Error message**

```
v2 ping attempt failed with error: Get https://artprod.company.com/v2/: x509: cannot validate certificate
for artprod.company.com because it doesn't contain any IP SANs
```

## Using Your Own Certificate

The NGINX configuration provided with Artifactory out-of-the-box references the internally bundled certificate and key which you may replace with your own certificate and key.

For details, please refer to Using Your Own Certificate.

---

## Setting Your Credentials Manually

If you are unable to log in to Docker, you may need to set your credentials manually.

The Docker command line tool supports authenticating sensitive operations, such as push, with the server using basic HTTP authentication.

To enforce authenticated access to docker repositories you need to provide the following parameters to the Docker configuration file.

- The Docker endpoint URL (must use HTTPS for basic authentication to work)
- Your Artifactory username and password (formatted `username:password`) as Base64 encoded strings
- Your email address

You can use the following command to get these strings directly from Artifactory and copy/paste them into your *~/.dockercfg* file:



**sudo**

If you are using Docker commands with "sudo" or as a root user (for example after installing the Docker client), note that the Docker configuration file should be placed under */root/.dockercfg*

**Getting .dockercfg entries directly from Artifactory**

```
$ curl -uadmin:password "https://artprod.company.com/<v1|v2>/auth"
{
 "https://artprod.company.com" : {
   "auth" : "YWRtaW46QVA1N05OaHZTMnM5Qk02RkR5RjNBVmF4TVFl",
   "email" : "admin@email.com"
 }
}
```

The Docker configuration file may contain a separate authentication block for each registry that you wish to access.

Below is an example with two URL endpoints:

```
{
        "https://artprod.company.com": {
                "auth":"YWRtaW46cGFzc3dvcmQ=",
                "email":"myemail@email.com"
        },
        "https://artprod2.company.com": {
                "auth":"YWRtaW46cGFzc3dvcmQ=",
                "email":"myemail@email.com"
        }
}
```

---

## Authenticating via OAuth

From version 4.4, Artifactory supports authentication of the Docker client using OAuth through the default GitHub OAuth provider. When authenticating using OAuth you will not need to provide additional credentials to execute `docker login` with Artifactory.

To set up OAuth authentication for your Docker client, execute the following steps:

- Under **General OAuth Settings**, make sure **Auto Create Artifactory Users** is checked to make sure a user record is created for you first time you log in to Artifactory with OAuth.
- Log in to Artifactory with OAuth using your Git Enterprise account

Once you are logged in to Artifactory through your Git Enterprise OAuth account, your Docker client will automatically detect this and use OAuth for authentication, so you do not need to provide additional credentials.

---

## Docker Manifest V2 Schema 1 Deprecation

To align with the Docker manifest V2 Schema 1 deprecation, from Artifactory version 6.15.0, Artifactory by default supports blocking Schema 1 requests. Only Docker images with the latest Docker manifest V2 Schema 2 are supported for:

- Push requests, for new local repositories.
- Pull requests, for new remote repositories.

Existing local and remote repositories continue to support both schemas.

Configuration can be changed at any time via REST API (using the `blockPushingSchema1` flag) or the UI.

When pushing a new Docker image, make sure you are using the latest Docker client versions, which will automatically convert your images accordingly. Artifactory will continue to allow Scheme 1 pull requests.

### AQL Example: Identify Schema 1 Docker manifest

From Artifactory version 6.15, to align with the Docker manifest V2 Schema 1 deprecation, Artifactory by default supports blocking Schema 1 requests. Only Docker images with the latest manifest V2 Schema 2 will be supported for:

- Push requests, for new local repositories.
- Pull requests, for new remote repositories.

This can be configured using REST API (with the `blockPushingSchema1` flag) or from within the UI.

A simple way to convert an image from schema1 to schema2 is to **use a recent Docker client, pull the image and then re-push**. The Docker client will convert the manifest format, but will not update the contents within the image.

**To identify outdated images using AQL**:

Run the following AQL command on the remote or local repository.

**Request**

```
items.find({"$and": [{"name":{"$eq":"manifest.json"}}, {"type":"file"}, {"repo":<REPOSITORY_KEY>}, {"@docker.
manifest.type":{"$ne":"application/vnd.docker.distribution.manifest.v2+json"}}]}).include("repo","path","
@docker.manifest","@docker.repoName")
```

**Response**

```
Property @docker.repoName is <IMAGE>
Property @docker.manifest is <TAG>
```

For remote repositories, this query will indicate whether the cache needs to be cleaned.

For local repositories, convert the outdated images found by running `docker pull` and `docker push`.

```
docker pull/push <DOCKER_DOMAIN>/<IMAGE>:<TAG>
```