

# Sign an image

*Estimated reading time: 5 minutes*

✔ These are the docs for DTR version 2.3.9

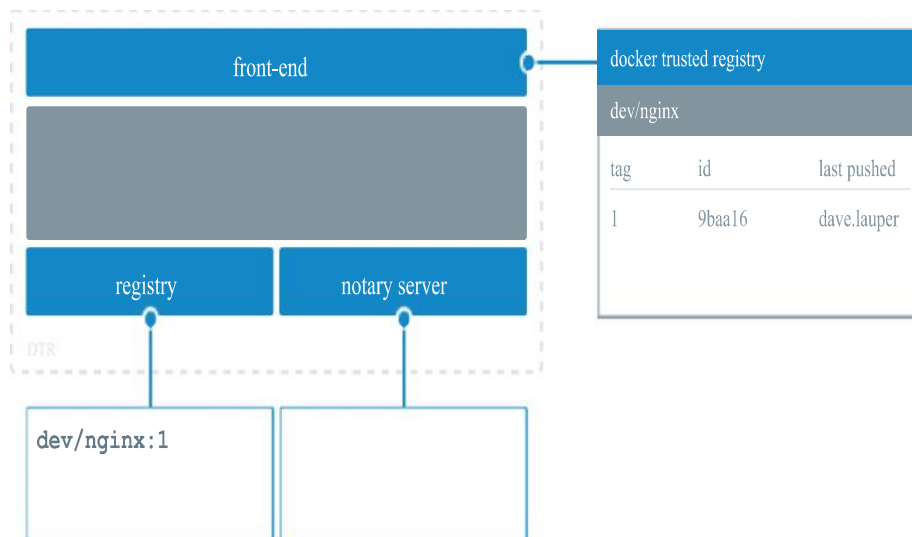
To select a different version, use the selector below.

2.3.9 ▼

By default, when you push an image to DTR, the Docker CLI client doesn't sign the image.



> `docker push`

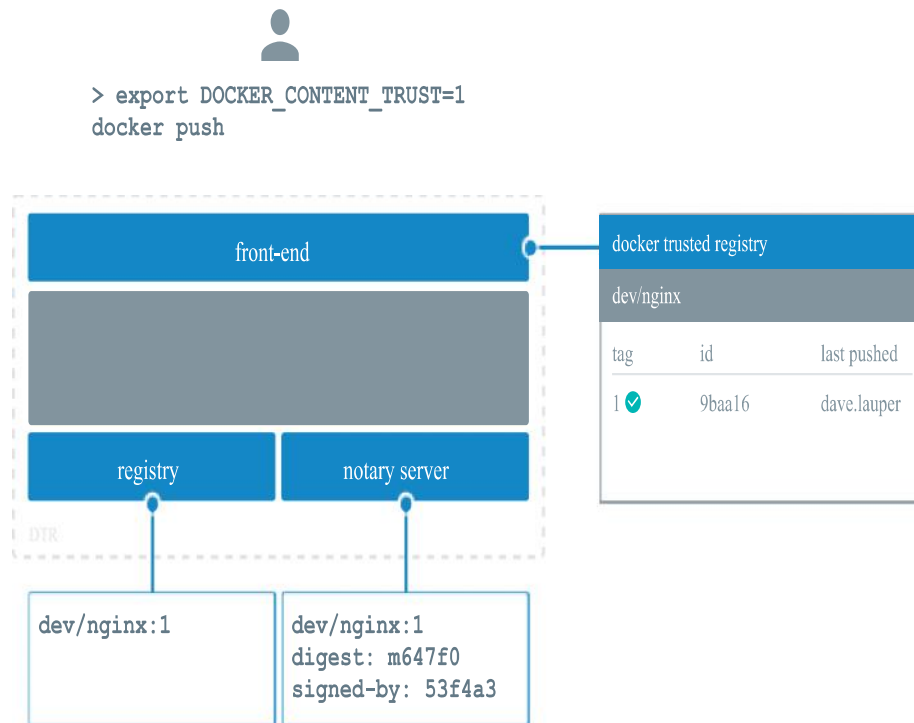


You can configure the Docker CLI client to sign the images you push to DTR. This allows whoever pulls your image to validate if they are getting the image you created, or a forged one.

To sign an image, you can run:

```
export DOCKER_CONTENT_TRUST=1
docker push <dtr-domain>/<repository>/<image>:<tag>
```

This pushes the image to DTR and creates trust metadata. It also creates public and private key pairs to sign the trust metadata, and pushes that metadata to the Notary Server internal to DTR.



## Sign images that UCP can trust

With the command above, you can sign your DTR images, but UCP doesn't trust them because it can't tie the private key you're using to sign the images to your UCP account.

To sign images in a way that UCP trusts them, you need to:

- Configure your Notary client
- Initialize trust metadata for the repository
- Delegate signing to the keys in your UCP client bundle

In this example we're going to pull an NGINX image from Docker Hub, re-tag it as `dtr.example.org/dev/nginx:1`, push the image to DTR and sign it in a way that is trusted by UCP. If you manage multiple repositories, you need to do the same procedure for every one of them.

## Configure your Notary client

Start by configuring your Notary client

(<https://docs.docker.com/datacenter/dtr/2.3/guides/user/access-dtr/configure-your-notary-client/>). This ensures the Docker and Notary CLI clients know about your UCP private keys.

## Initialize the trust metadata

Then you need to initialize the trust metadata for the new repository, and the easiest way to do it is by pushing an image to that repository. Navigate to the DTR web UI, and create a repository for your image. In this example we've created the `dev/nginx` repository.

From the Docker CLI client, pull an NGINX image from Docker Hub, re-tag it, sign and push it to DTR.

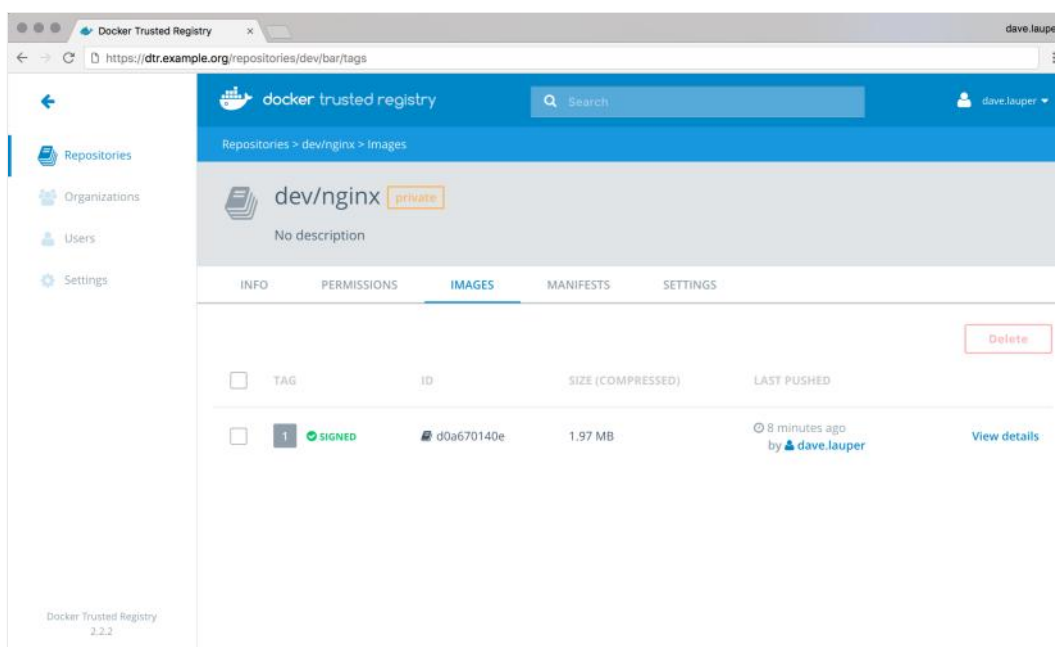
```
# Pull NGINX from Docker Hub
docker pull nginx:latest

# Re-tag NGINX
docker tag nginx:latest dtr.example.org/dev/nginx:1

# Log into DTR
docker login dtr.example.org

# Sign and push the image to DTR
export DOCKER_CONTENT_TRUST=1
docker push dtr.example.org/dev/nginx:1
```

This pushes the image to DTR and initializes the trust metadata for that repository.



DTR shows that the image is signed, but UCP doesn't trust the image because it doesn't have any information about the private keys used to sign the image.

## Delegate trust to your UCP keys

To sign images in a way that is trusted by UCP, you need to delegate trust, so that you can sign images with the private keys in your UCP client bundle.

When delegating trust you associate a public key certificate with a role name.

UCP requires that you delegate trust to two different roles:

- `targets/releases`
- `targets/<role>`, where `<role>` is the UCP team the user belongs to

In this example we delegate trust to `targets/releases` and `targets/admin`:

```
# Delegate trust, and add that public key with the role targets/releases
notary delegation add --publish \
  dtr.example.org/dev/nginx targets/releases \
  --all-paths <ucp-cert.pem>

# Delegate trust, and add that public key with the role targets/admin
notary delegation add --publish \
  dtr.example.org/dev/nginx targets/admin \
  --all-paths <ucp-cert.pem>
```

To push the new signing metadata to the Notary server, you need to push the image again:

```
docker push dtr.example.org/dev/nginx:1
```

## Under the hood

Both Docker and Notary CLI clients interact with the Notary server to:

- Keep track of the metadata of signed images
- Validate the signatures of the images you pull

This metadata is also kept locally in `~/.docker/trust`.

```
.
|-- private
|   |-- root_keys
|   |   `-- 993ad247476da081e45fdb6c28edc4462f0310a55da4acf1e08404c5
51d94c14.key
|   `-- tuf_keys
|       |-- dtr.example.org
|           |-- dev
|               |-- nginx
|                   |-- 98a93b2e52c594de4d13d7268a4a5f28ade5fc1cb5f4
4cc3a4ab118572a86848.key
|                   |-- f7917aef77d0d4bf8204af78c0716dac6649346ebea1
c4cde7a1bfa363c502ce.key
|-- tuf
    |-- dtr.example.org
        |-- dev
            |-- nginx
                |-- changelist
                |-- metadata
                    |-- root.json
                    |-- snapshot.json
                    |-- targets.json
                    |-- timestamp.json
```

The `private` directory contains the private keys the Docker CLI client uses to sign the images. Make sure you create backups of this directory so that you don't lose your signing keys.

The Docker and Notary CLI clients integrate with Yubikey. If you have a Yubikey plugged in when initializing trust for a repository, the root key is stored on the Yubikey instead of in the trust directory. When you run any command that needs the `root` key, Docker and Notary CLI clients look on the Yubikey first, and use the trust directory as a fallback.

The `tuf` directory contains the trust metadata for the images you've signed. For each repository there are four files.

File	Description
<code>root.json</code>	Has data about other keys and their roles. This data is signed by the root key.
<code>targets.json</code>	Has data about the digest and size for an image. This data is signed by the target key.
<code>snapshot.json</code>	Has data about the version number of the <code>root.json</code> and <code>targets.json</code> files. This data is signed by the snapshot key.
<code>timestamp.json</code>	Has data about the digest, size, and version number for the <code>snapshot.json</code> file. This data is signed by the timestamp key.

Learn more about trust metadata

([https://docs.docker.com/notary/service\\_architecture/](https://docs.docker.com/notary/service_architecture/)).

registry (<https://docs.docker.com/glossary/?term=registry>), sign  
(<https://docs.docker.com/glossary/?term=sign>), trust  
(<https://docs.docker.com/glossary/?term=trust>)

# Scan images for vulnerabilities

*Estimated reading time: 6 minutes*

✔ These are the docs for DTR version 2.3.9

To select a different version, use the selector below.

2.3.9 ▼



(<https://www.youtube.com/watch?v=121poCB0Nn8>)

Docker Trusted Registry can scan images in your repositories to verify that they are free from known security vulnerabilities or exposures, using Docker Security Scanning. The results of these scans are reported for each image tag.

Docker Security Scanning is available as an add-on to Docker Trusted Registry, and an administrator configures it for your DTR instance. If you do not see security scan results available on your repositories, your organization may not have purchased the Security Scanning feature or it may be disabled. See [Set up Security Scanning in DTR](#)

(<https://docs.docker.com/datacenter/dtr/2.3/guides/admin/configure/set-up-vulnerability-scans/>) for more details.

Tip: Only users with write access to a repository can manually start a scan. Users with read-only access can view the scan results, but cannot start a new scan.

## The Docker Security Scan process

Scans run either on demand when a user clicks the Start a Scan links or Scan button (see Manual scanning (/datacenter/dtr/2.3/guides/user/manage-images/scan-images-for-vulnerabilities/#manual-scanning) below), or automatically on any `docker push` to the repository.

First the scanner performs a binary scan on each layer of the image, identifies the software components in each layer, and indexes the SHA of each component in a bill-of-materials. A binary scan evaluates the components on a bit-by-bit level, so vulnerable components are discovered even if they are statically-linked or under a different name.

The scan then compares the SHA of each component against the US National Vulnerability Database that is installed on your DTR instance. When this database is updated, DTR reviews the indexed components for newly discovered vulnerabilities.

DTR scans both Linux and Windows images, but by default Docker doesn't push foreign image layers for Windows images so DTR can't scan them. If you want DTR to scan your Windows images, configure Docker to always push image layers (<https://docs.docker.com/datacenter/dtr/2.3/guides/user/manage-images/pull-and-push-images/>), and it will scan the non-foreign layers.

## Security scan on push

By default, Docker Security Scanning runs automatically on `docker push` to an image repository.

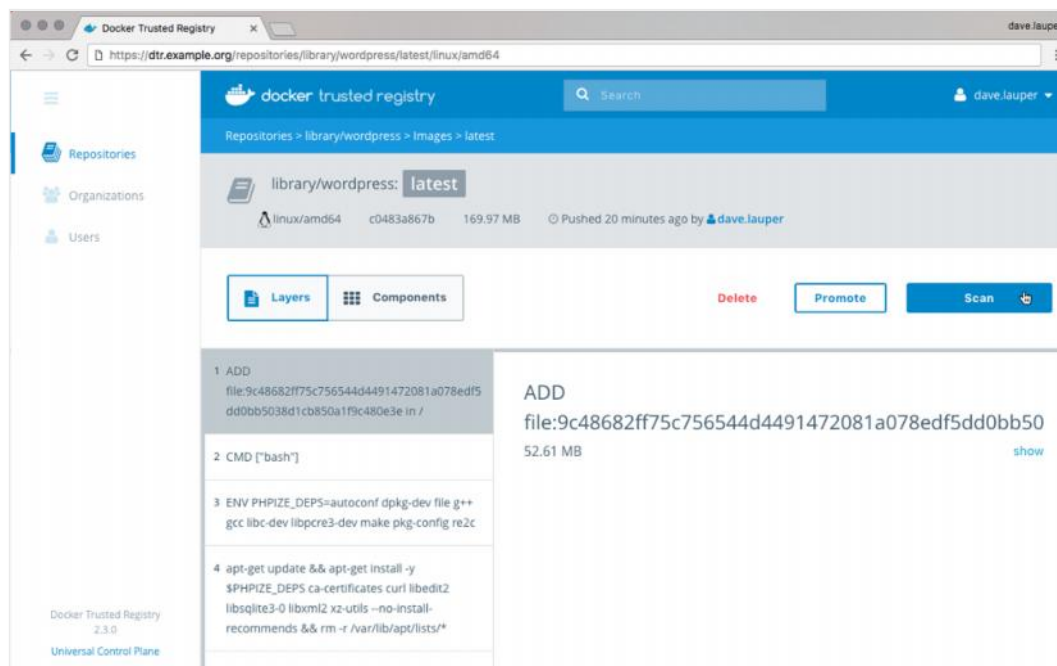
If your DTR instance is configured in this way, you do not need to do anything once your `docker push` completes. The scan runs automatically, and the results are reported in the repository's Images tab after the scan finishes.

## Manual scanning



If your repository owner enabled Docker Security Scanning but disabled automatic scanning, you can manually start a scan for images in repositories to which you have `write` access.

To start a security scan, navigate to the tag details, and click the Scan button.



DTR begins the scanning process. You will need to refresh the page to see the results once the scan is complete.

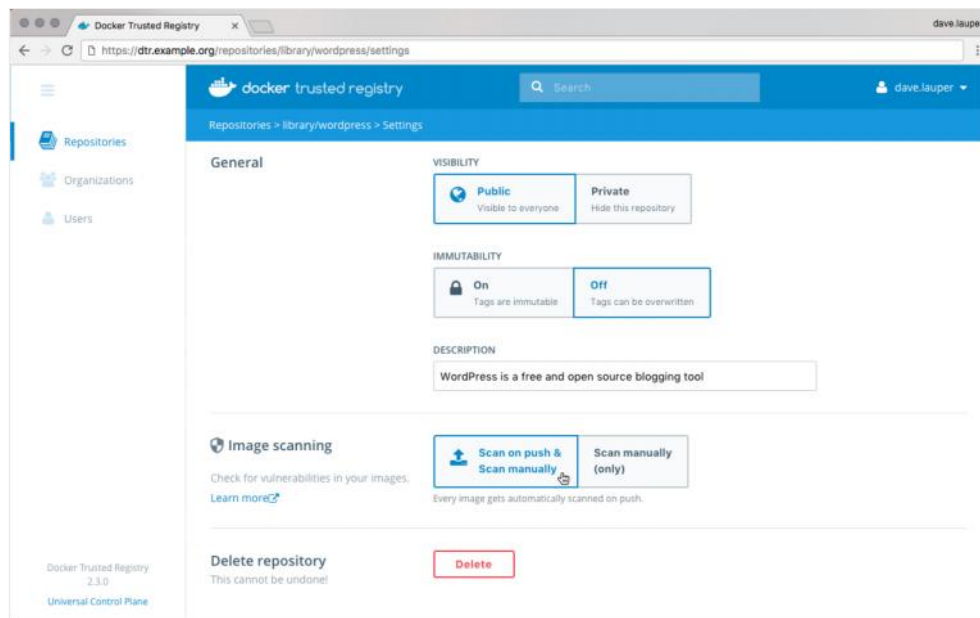
## Change the scanning mode

You can change the scanning mode for each individual repository at any time. You might want to disable scanning if you are pushing an image repeatedly during troubleshooting and don't want to waste resources scanning and re-scanning, or if a repository contains legacy code that is not used or updated frequently.

Note: To change an individual repository's scanning mode, you must have `write` or `administrator` access to the repo.

To change the repository scanning mode:

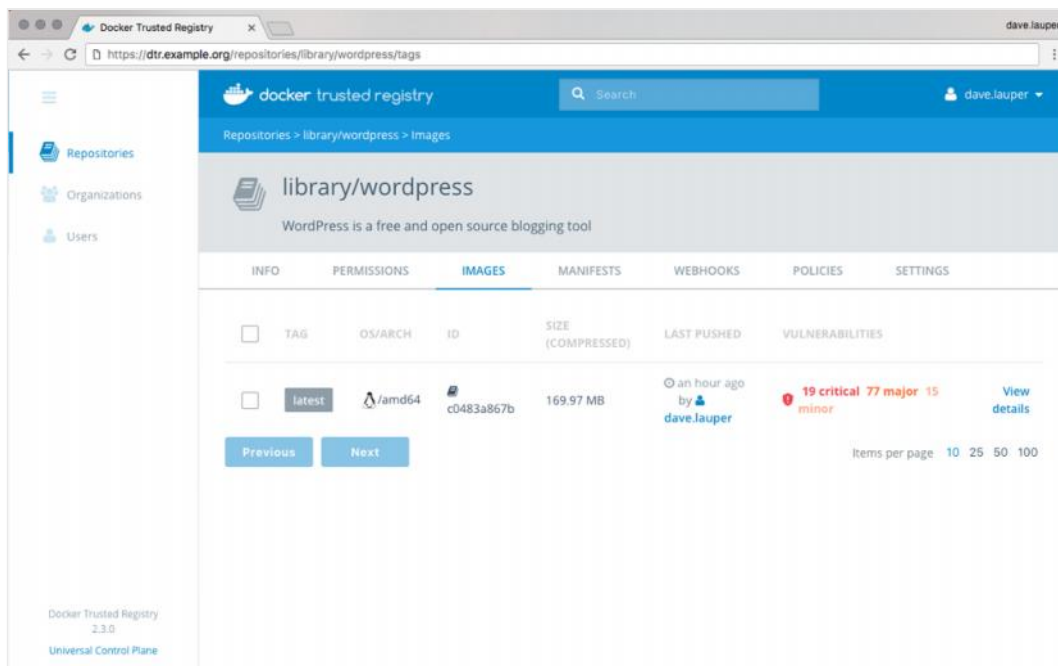
1. Navigate to the repository, and click the Settings tab.
2. Scroll down to the Image scanning section.
3. Select the desired scanning mode.



## View security scan results

Once DTR has run a security scan for an image, you can view the results.

The Images tab for each repository includes a summary of the most recent scan results for each image.



- A green shield icon with a check mark indicates that the scan did not find any vulnerabilities.
- A red or orange shield icon indicates that vulnerabilities were found, and the number of vulnerabilities is included on that same line.

If the vulnerability scan can't detect the version of a component, it reports the vulnerabilities for all versions of that component.

From the Images tab you can click View details for a specific tag to see the full scan results. The top of the page also includes metadata about the image, including the SHA, image size, date last pushed and user who last pushed, the security scan summary, and the security scan progress.

The scan results for each image include two different modes so you can quickly view details about the image, its components, and any vulnerabilities found.

- The Layers view lists the layers of the image in order as they are built by the Dockerfile.

This view can help you find exactly which command in the build introduced the vulnerabilities, and which components are associated with that single command. Click a layer to see a summary of its components. You can then click on a component to switch to the Component view and get more details about the specific item.

Tip: The layers view can be long, so be sure to scroll down if you don't immediately see the reported vulnerabilities.

The screenshot shows the Docker Trusted Registry (DTR) interface. The top navigation bar includes 'Repositories', 'Organizations', and 'Users'. The main content area displays the details for the 'library/wordpress:latest' image. The 'Layers' tab is active, showing a list of build layers. The 'ADD' layer is selected, displaying its components and associated vulnerabilities.

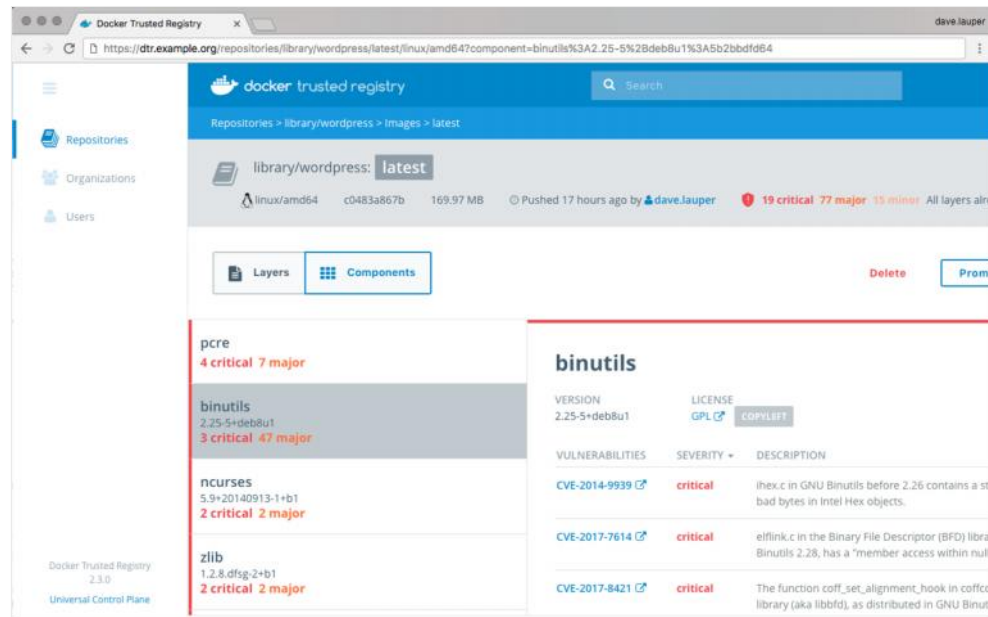
Layer	Command	Components	Vulnerabilities
1	ADD	file:9c48682ff75c756544d4491472081a078edf5dd0bb5038d1cb850a1f9c480e3e in /	19 critical 77 major 15 minor
2	CMD ["bash"]		
3	ENV PHPIZE_DEPS=autoconf dpkg-dev file g++ gcc libc-dev libpcres-dev make pkg-config re2c		
4	apt-get update && apt-get install -y \$PHPIZE_DEPS ca-certificates curl libedit2 libsqlite3-0 libxml2 xz-utils --no-install-recommends		

The 'ADD' layer details show the following components and vulnerabilities:

Component	Vulnerabilities
zlib 1.2.8.dfsg-2+b1	2 critical 2 major
ncurses 5.9+20140913-1+b1	2 critical 2 major
systemd 215-17+deb8u7	1 critical 2 major

- The Components view lists the individual component libraries indexed by the scanning system, in order of severity and number of vulnerabilities found, most vulnerable first.

Click on an individual component to view details about the vulnerability it introduces, including a short summary and a link to the official CVE database report. A single component can have multiple vulnerabilities, and the scan report provides details on each one. The component details also include the license type used by the component, and the filepath to the component in the image.



## What to do next

If you find that an image in your registry contains vulnerable components, you can use the linked CVE scan information in each scan report to evaluate the vulnerability and decide what to do.

If you discover vulnerable components, you should check if there is an updated version available where the security vulnerability has been addressed. If necessary, you might contact the component's maintainers to ensure that the vulnerability is being addressed in a future version or patch update.

If the vulnerability is in a **base layer** (such as an operating system) you might not be able to correct the issue in the image. In this case, you might switch to a different version of the base layer, or you might find an equivalent, less vulnerable base layer. You might also decide that the vulnerability or exposure is acceptable.

Address vulnerabilities in your repositories by updating the images to use updated and corrected versions of vulnerable components, or by using a different components that provide the same functionality. When you have

updated the source code, run a build to create a new image, tag the image, and push the updated image to your DTR instance. You can then re-scan the image to confirm that you have addressed the vulnerabilities.

registry (<https://docs.docker.com/glossary/?term=registry>), scan (<https://docs.docker.com/glossary/?term=scan>), vulnerability (<https://docs.docker.com/glossary/?term=vulnerability>)

# Content trust in Docker

*Estimated reading time: 13 minutes*

When transferring data among networked systems, *trust* is a central concern. In particular, when communicating over an untrusted medium such as the internet, it is critical to ensure the integrity and the publisher of all the data a system operates on. You use Docker Engine to push and pull images (data) to a public or private registry. Content trust gives you the ability to verify both the integrity and the publisher of all the data received from a registry over any channel.

## About trust in Docker

Docker Content Trust (DCT) allows operations with a remote Docker registry to enforce client-side signing and verification of image tags. DCT provides the ability to use digital signatures for data sent to and received from remote Docker registries. These signatures allow client-side verification of the integrity and publisher of specific image tags.

Once DCT is enabled, image publishers can sign their images. Image consumers can ensure that the images they use are signed. Publishers and consumers can either be individuals or organizations. DCT supports users and automated processes such as builds.

When you enable DCT, signing occurs on the client after push and verification happens on the client after pull if you use Docker CE. If you use UCP, and you have configured UCP to require images to be signed before deploying, signing is verified by UCP.

## Image tags and DCT

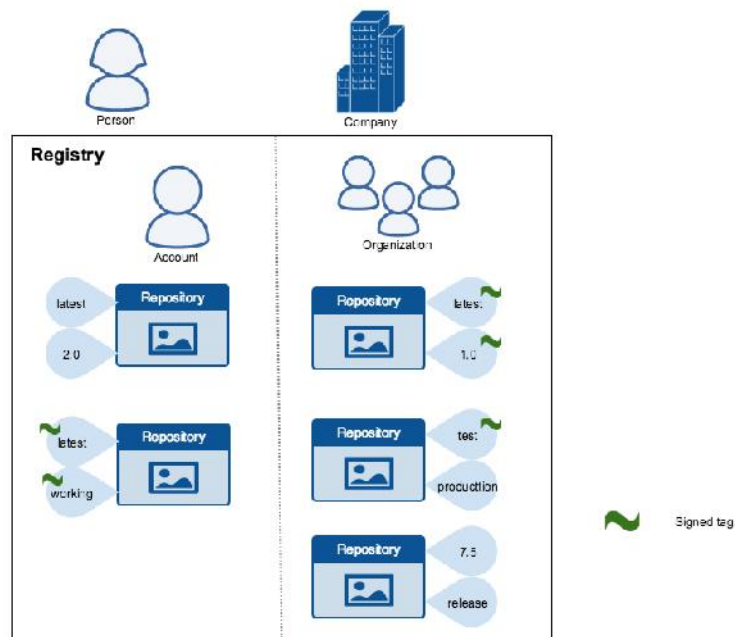
An individual image record has the following identifier:

```
[REGISTRY_HOST[:REGISTRY_PORT]/]REPOSITORY[:TAG]
```

A particular image **REPOSITORY** can have multiple tags. For example, `latest` and `3.1.2` are both tags on the `mongo` image. An image publisher can build an image and tag combination many times changing the image with each build.

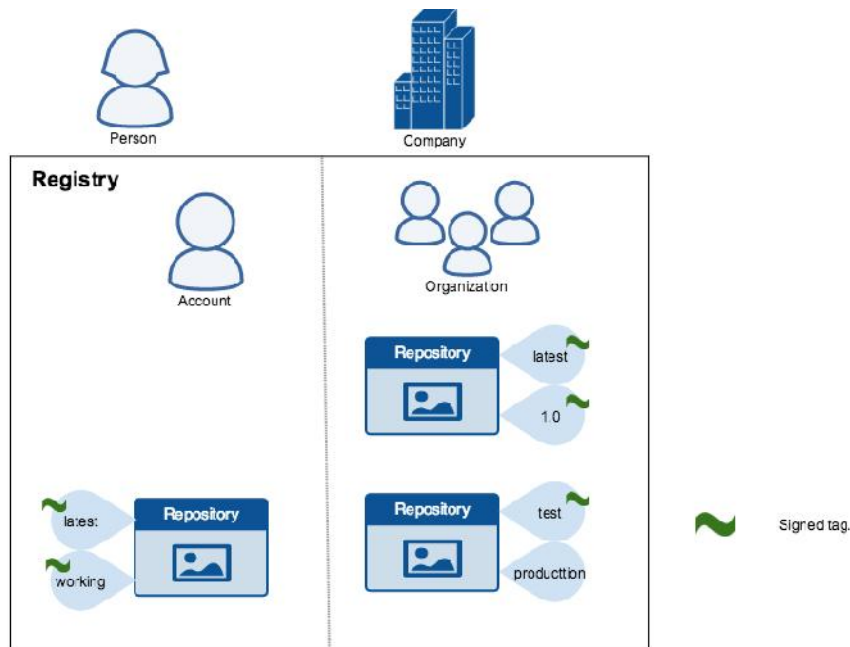
DCT is associated with the **TAG** portion of an image. Each image repository has a set of keys that image publishers use to sign an image tag. Image publishers have discretion on which tags they sign.

An image repository can contain an image with one tag that is signed and another tag that is not. For example, consider the Mongo image repository (<https://hub.docker.com/r/library/mongo/tags/>). The `latest` tag could be unsigned while the `3.1.6` tag could be signed. It is the responsibility of the image publisher to decide if an image tag is signed or not. In this representation, some image tags are signed, others are not:



Publishers can choose to sign a specific tag or not. As a result, the content of an unsigned tag and that of a signed tag with the same name may not match. For example, a publisher can push a tagged image `someimage:latest` and sign it. Later, the same publisher can push an unsigned `someimage:latest` image. This second push replaces the last unsigned tag `latest` but does not affect the signed `latest` version. The ability to choose which tags they can sign, allows publishers to iterate over the unsigned version of an image before officially signing it.

Image consumers can enable DCT to ensure that images they use were signed. If a consumer enables DCT, they can only pull, run, or build with trusted images. Enabling DCT is like wearing a pair of rose-colored glasses. Consumers “see” only signed image tags and the less desirable, unsigned image tags are “invisible” to them.



To the consumer who has not enabled DCT, nothing about how they work with Docker images changes. Every image is visible regardless of whether it is signed or not.

## DCT operations and keys

When DCT is enabled, `docker` CLI commands that operate on tagged images must either have content signatures or explicit content hashes. The commands that operate with DCT are:

- `push`
- `build`
- `create`
- `pull`
- `run`

For example, with DCT enabled a `docker pull someimage:latest` only succeeds if `someimage:latest` is signed. However, an operation with an explicit content hash always succeeds as long as the hash exists:

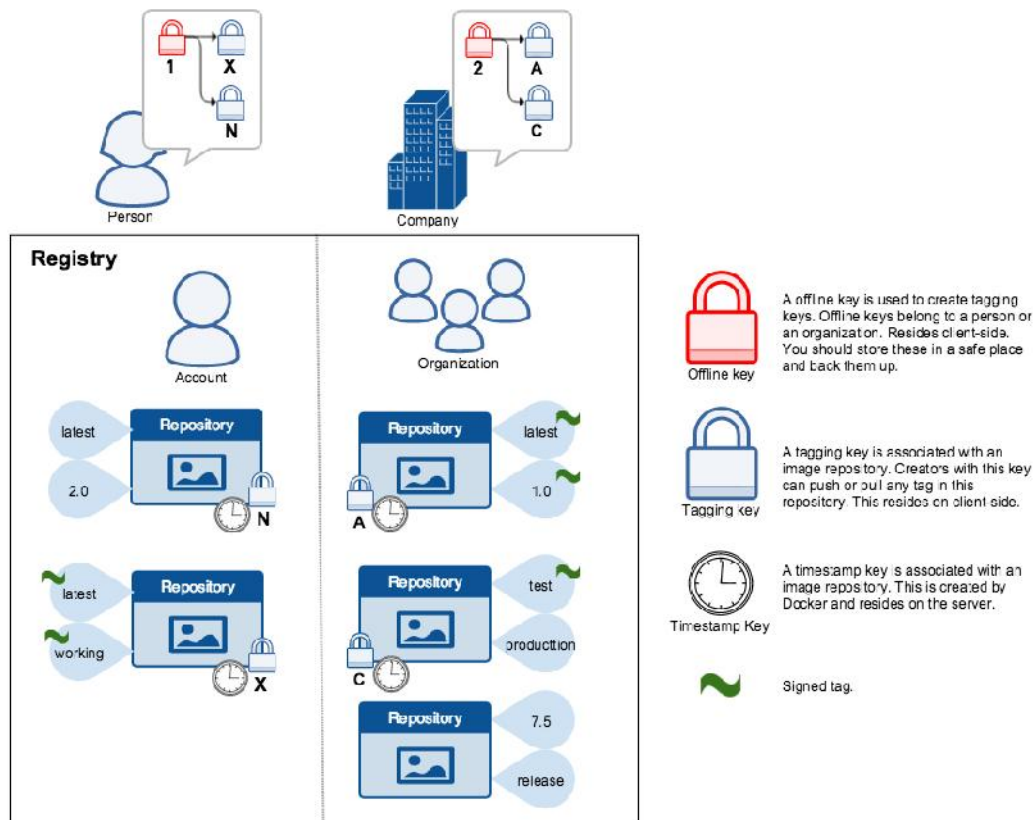


```
$ docker pull someimage@sha256:d149ab53f8718e987c3a3024bb8aa0e2caadf6c0328f1d9d850b2a2a67f2819a
```

Trust for an image tag is managed through the use of signing keys. A key set is created when an operation using DCT is first invoked. A key set consists of the following classes of keys:

- an offline key that is the root of DCT for an image tag
- repository or tagging keys that sign tags
- server-managed keys such as the timestamp key, which provides freshness security guarantees for your repository

The following image depicts the various signing keys and their relationships:



⊗ **WARNING:** Loss of the root key is very difficult to recover from. Correcting this loss requires intervention from Docker Support (<https://support.docker.com>) to reset the repository state. This loss also requires manual intervention from every consumer that used a signed tag from this repository prior to the loss.

You should backup the root key somewhere safe. Given that it is only required to create new repositories, it is a good idea to store it offline in hardware. For details on securing, and backing up your keys, make sure you read how to manage keys for DCT ([https://docs.docker.com/engine/security/trust/trust\\_key\\_mng/](https://docs.docker.com/engine/security/trust/trust_key_mng/)).

## Survey of typical DCT operations

This section surveys the typical trusted operations users perform with Docker images. Specifically, we go through the following steps to help us exercise these various trusted operations:

- Build and push an unsigned image
- Pull an unsigned image
- Build and push a signed image
- Pull the signed image pushed above
- Pull unsigned image pushed above

## Enabling DCT in Docker Engine Configuration

Engine Signature Verification prevents the following behaviors on an image:

- Running a container to build an image (the base image must be signed, or must be scratch)
- Creating a container from an image that is not signed

DCT does not verify that a running container's filesystem has not been altered from what was in the image. For example, it does not prevent a container from writing to the filesystem, nor the container's filesystem from being altered on disk.

It will also pull and run signed images from registries, but will not prevent unsigned images from being imported, loaded, or created.

The image name, digest, or tag must be verified if DCT is enabled. The latest DCT metadata for an image must be downloaded from the trust server associated with the registry:

- If an image tag does not have a digest, the DCT metadata translates the name to an image digest
- If an image tag has an image digest, the DCT metadata verifies that the name matches the provided digest
- If an image digest does not have an image tag, the DCT metadata does a reverse lookup and provides the image tag as well as the digest.

The signature verification feature is configured in the Docker daemon

configuration file `daemon.json` .

```
{
  ...
  "content-trust": {
    "trust-pinning": {
      "root-keys": {
        "myregistry.com/myorg/*": ["keyID1", "keyID2"],
        "myregistry.com/otherorg/repo": ["keyID3"]
      },
      "official-images": true,
    },
    "mode": "disabled" | "permissive" | "enforced",
    "allow-expired-trust-cache": true,
  }
}
```

Stanza	Description
<code>trust-pinning:root-keys</code>	<p>Root key IDs are canonical IDs that sign the root metadata of the image trust data. In Docker Certified Trust (DCT), the root keys are unique certificates tying the name of the image to the repo metadata. The private key ID (the canonical key ID) corresponding to the certificate does not depend on the image name. If an image's name matches more than one glob, then the most specific (longest) one is chosen.</p>
<code>trust-pinning:library-images</code>	<p>This option pins the official libraries ( <code>docker.io/library/*</code> ) to the hard-coded Docker official images root key. DCT trusts the official images by default. This is in addition to whatever images are specified by <code>trust-pinning:root-keys</code> . If <code>trustpinning:root-keys</code> specifies a key mapping for <code>docker.io/library/*</code> , those keys will be preferred for trust pinning. Otherwise, if a more general <code>docker.io/*</code> or <code>*</code> are specified, the official images key will be preferred.</p>

Stanza	Description
<code>allow-expired-trust-cache</code>	Specifies whether cached locally expired metadata validates images if an external server is unreachable or does not have image trust metadata. This is necessary for machines which may be often offline, as may be the case for edge. This does not provide mitigations against freeze attacks, which is a necessary to provide availability in low-connectivity environments.
<code>mode</code>	<p>Specifies whether DCT is enabled and enforced. Valid modes are: <code>disabled</code> : Verification is not active and the remainder of the content-trust related metadata will be ignored. <i>NOTE</i> that this is the default configuration if <code>mode</code> is not specified.</p> <p><code>permissive</code> : Verification will be performed, but only failures will only be logged and remain unenforced. This configuration is intended for testing of changes related to content-trust.</p> <p><code>enforced</code> : DCT will be enforced and an image that cannot be verified successfully will not be pulled or run.</p>

*Note:* The DCT configuration defined here is agnostic of any policy defined in UCP (<https://docs.docker.com/v17.09/datacenter/ucp/2.0/guides/content-trust/#configure-ucp>). Images that can be deployed by the UCP trust policy but are disallowed by the Docker Engine configuration will not successfully be deployed or run on that engine.

## Enable and disable DCT per-shell or per-invocation

Instead of enabling DCT through the system-wide configuration, DCT can be enabled or disabled on a per-shell or per-invocation basis.

To enable on a per-shell basis, enable the `DOCKER_CONTENT_TRUST` environment variable. Enabling per-shell is useful because you can have one shell configured for trusted operations and another terminal shell for untrusted operations. You can also add this declaration to your shell profile to have it enabled by default.

To enable DCT in a `bash` shell enter the following command:

```
export DOCKER_CONTENT_TRUST=1
```

Once set, each of the “tag” operations requires a key for a trusted tag.

In an environment where `DOCKER_CONTENT_TRUST` is set, you can use the `--disable-content-trust` flag to run individual operations on tagged images without DCT on an as-needed basis.

Consider the following Dockerfile that uses an untrusted parent image:

```
$ cat Dockerfile
FROM docker/trusttest:latest
RUN echo
```

To build a container successfully using this Dockerfile, one can do:

```
$ docker build --disable-content-trust -t <username>/nottrusttest:latest .
Sending build context to Docker daemon 42.84 MB
...
Successfully built f21b872447dc
```

The same is true for all the other commands, such as `pull` and `push` :

```
$ docker pull --disable-content-trust docker/trusttest:latest
...
$ docker push --disable-content-trust <username>/nottrusttest:latest
...

```

To invoke a command with DCT enabled regardless of whether or how the `DOCKER_CONTENT_TRUST` variable is set:

```
$ docker build --disable-content-trust=false -t <username>/trusttest:testing .
```

All of the trusted operations support the `--disable-content-trust` flag.

## Push trusted content

To create signed content for a specific image tag, simply enable DCT and push a tagged image. If this is the first time you have pushed an image using DCT on your system, the session looks like this:

```
$ docker push <username>/trusttest:testing
The push refers to a repository [docker.io/<username>/trusttest] (len: 1)
9a61b6b1315e: Image already exists
902b87aaec9: Image already exists
latest: digest: sha256:d02adacee0ac7a5be140adb94fa1dae64f4e71a68696e7f8e7cbf9db8dd49418 size: 3220
Signing and pushing trust metadata
You are about to create a new root signing key passphrase. This passphrase
will be used to protect the most sensitive key in your signing system. Please
choose a long, complex passphrase and be careful to keep the password and the
key file itself secure and backed up. It is highly recommended that you use a
password manager to generate the passphrase and keep it safe. There will be no
way to recover this key. You can find the key in your config directory.
Enter passphrase for new root key with id a1d96fb:
Repeat passphrase for new root key with id a1d96fb:
Enter passphrase for new repository key with id docker.io/<username>/trusttest (3a932f1):
Repeat passphrase for new repository key with id docker.io/<username>/trusttest (3a932f1):
Finished initializing "docker.io/<username>/trusttest"
```

When you push your first tagged image with DCT enabled, the `docker` client recognizes this is your first push and:

- alerts you that it is creating a new root key
- requests a passphrase for the root key
- generates a root key in the `~/.docker/trust` directory
- requests a passphrase for the repository key
- generates a repository key in the `~/.docker/trust` directory

The passphrase you chose for both the root key and your repository key-pair should be randomly generated and stored in a *password manager*.

NOTE: If you omit the `testing` tag, DCT is skipped. This is true even if DCT is enabled and even if this is your first push.

```
$ docker push <username>/trusttest
The push refers to a repository [docker.io/<username>/trusttest] (len: 1)
9a61b6b1315e: Image successfully pushed
902b87aaec9: Image successfully pushed
latest: digest: sha256:a9a9c4402604b703bed1c847f6d85faac97686e48c579bd9c3b0fa6694a398fc size: 3220
No tag specified, skipping trust metadata push
```

It is skipped because as the message states, you did not supply an image `TAG` value. In DCT, signatures are associated with tags.

Once you have a root key on your system, subsequent images repositories you create can use that same root key:

```
$ docker push docker.io/<username>/otherimage:latest
The push refers to a repository [docker.io/<username>/otherimage] (len: 1)
a9539b34a6ab: Image successfully pushed
b3dbab3810fc: Image successfully pushed
latest: digest: sha256:d2ba1e603661a59940bfad7072eba698b79a8b20ccbb4e3bfb6f9e367ea43939 size: 3346
Signing and pushing trust metadata
Enter key passphrase for root key with id a1d96fb:
Enter passphrase for new repository key with id docker.io/<username>/otherimage (bb045e3):
Repeat passphrase for new repository key with id docker.io/<username>/otherimage (bb045e3):
Finished initializing "docker.io/<username>/otherimage"
```

The new image has its own repository key and timestamp key. The `latest` tag is signed with both of these.

## Pull image content

A common way to consume an image is to `pull` it. With DCT enabled, the Docker client only allows `docker pull` to retrieve signed images. Let's try to pull the image you signed and pushed earlier:

```
$ docker pull <username>/trusttest:testing
Pull (1 of 1): <username>/trusttest:testing@sha256:d149ab53f871
...
Tagging <username>/trusttest@sha256:d149ab53f871 as docker/trusttest:testing
```

In the following example, the command does not specify a tag, so the system uses the `latest` tag by default again and the `docker/trusttest:latest` tag is not signed.

```
$ docker pull docker/trusttest
Using default tag: latest
no trust data available
```

Because the tag `docker/trusttest:latest` is not trusted, the `pull` fails.

## Related information

- Manage keys for content trust  
([https://docs.docker.com/engine/security/trust/trust\\_key\\_mng/](https://docs.docker.com/engine/security/trust/trust_key_mng/))
- Automation with content trust  
([https://docs.docker.com/engine/security/trust/trust\\_automation/](https://docs.docker.com/engine/security/trust/trust_automation/))
- Delegations for content trust  
([https://docs.docker.com/engine/security/trust/trust\\_delegation/](https://docs.docker.com/engine/security/trust/trust_delegation/))
- Play in a content trust sandbox  
([https://docs.docker.com/engine/security/trust/trust\\_sandbox/](https://docs.docker.com/engine/security/trust/trust_sandbox/))

content (<https://docs.docker.com/glossary/?term=content>), trust  
(<https://docs.docker.com/glossary/?term=trust>), security  
(<https://docs.docker.com/glossary/?term=security>), docker  
(<https://docs.docker.com/glossary/?term=docker>), documentation  
(<https://docs.docker.com/glossary/?term=documentation>)



# Access control model

*Estimated reading time: 5 minutes*

✔ These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

2.2.14 ▼

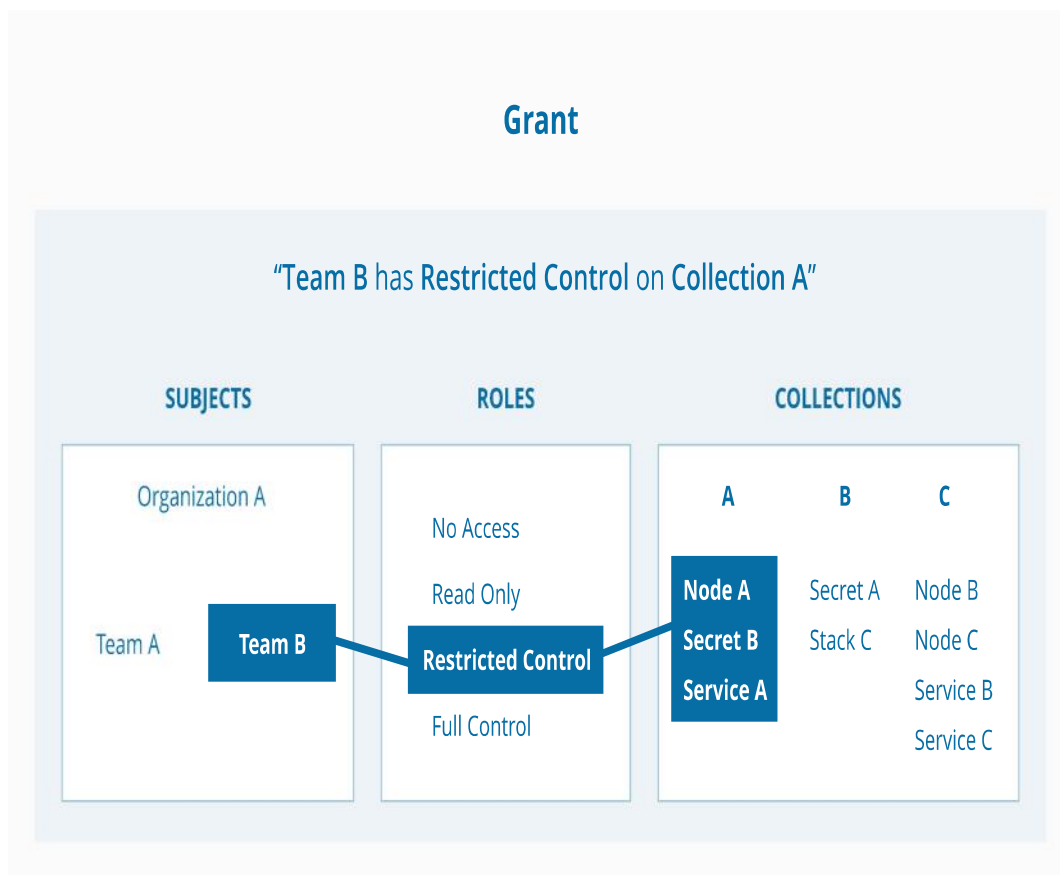
With Docker Universal Control Plane, you get to control who can create and edit container resources in your swarm, like services, images, networks, and volumes. You can grant and manage permissions to enforce fine-grained access control as needed.

## Grant access to swarm resources

If you're a UCP administrator, you can create *grants* to control how users and organizations access swarm resources.

A grant is made up of a *subject*, a *role*, and a *resource collection*. A grant defines who (subject) has how much access (role) to a set of resources (collection). Learn how to grant permissions to users based on roles

(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/grant-permissions/>).



An administrator is a user who can manage grants, subjects, roles, and collections. An administrator identifies which operations can be performed against specific resources and who can perform these actions. An administrator can create and manage role assignments against subject in the system. Only an administrator can manage subjects, grants, roles, and collections.

## Subjects

A subject represents a user, team, or organization. A subject is granted a role for a collection of resources.

- **User:** A person that the authentication backend validates. You can assign users to one or more teams and one or more organizations.
- **Organization:** A group of users that share a specific set of permissions, defined by the roles of the organization.
- **Team:** A group of users that share a set of permissions defined in the team itself. A team exists only as part of an organization, and all of its members must be members of the organization. Team members share organization permissions. A team can be in one organization only.

## Roles

A role is a set of permitted API operations that you can assign to a specific subject and collection by using a grant. UCP administrators view and manage roles by navigating to the Roles page. Learn more about roles and permissions (<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/permission-levels/>).

## Resource collections

Docker EE enables controlling access to swarm resources by using *collections*. A collection is a grouping of swarm cluster resources that you access by specifying a directory-like path.

Swarm resources that can be placed in to a collection include:

- Physical or virtual nodes
- Containers
- Services
- Networks
- Volumes
- Secrets
- Application configs

## Collection architecture

Grants tie together who has which kind of access to what resources. Grants are effectively ACLs, which grouped together, can provide full and comprehensive access policies for an entire organization. However, before grants can be implemented, collections need to be designed to group resources in a way that makes sense for an organization.

The following example shows a potential access policy of an organization. Consider an organization with two application teams, Mobile and Payments, that will share cluster hardware resources, but still need to segregate access to the applications. Collections should be designed to map to the organizational structure desired, in this case the two application teams. Their collection architecture for a production UCP cluster might look something like this:

```
prod
├─ mobile
└─ payments
```

A subject that has access to any level in a collection hierarchy will have that same access to any collections below it.

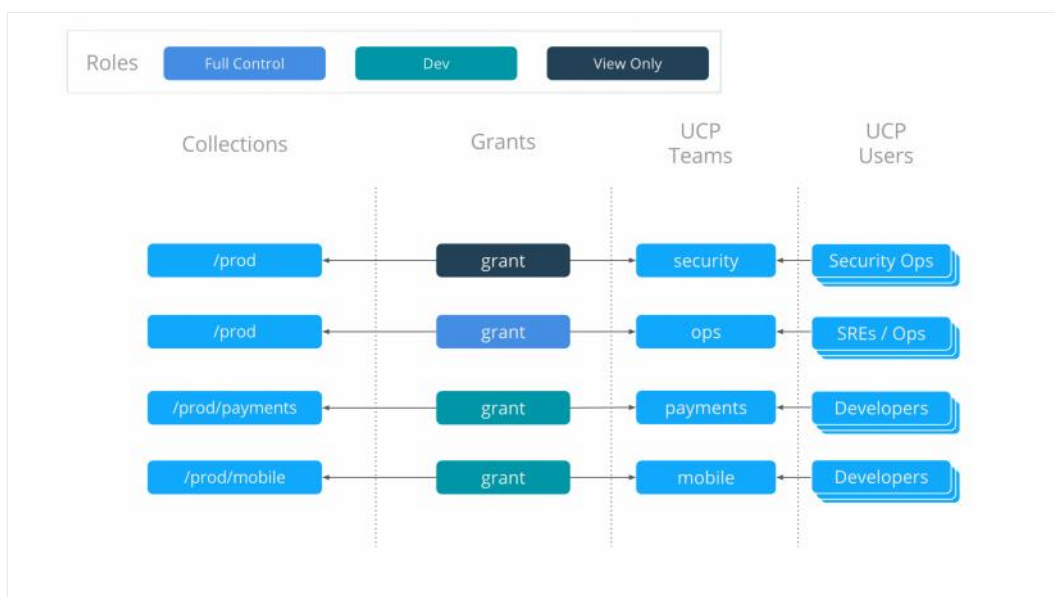
## Role composition

Roles define what operations can be done against cluster resources. An organization will likely use several different kinds of roles to give the right kind of access. A given team or user may have different roles provided to them depending on what resource they are accessing. There are default roles provided by UCP and also the ability to build custom roles. In this example three different roles are used:

- Full Control - This is a default role that provides the full list of operations against cluster resources.
- View Only - This is also a default role that allows a user to see resources, but not to edit or delete.
- Dev - This is not a default role, but a potential custom role. In this example “Dev” includes the ability to view containers and also `docker exec`. This allows developers to run a shell inside their container process but not see or change any other cluster resources.

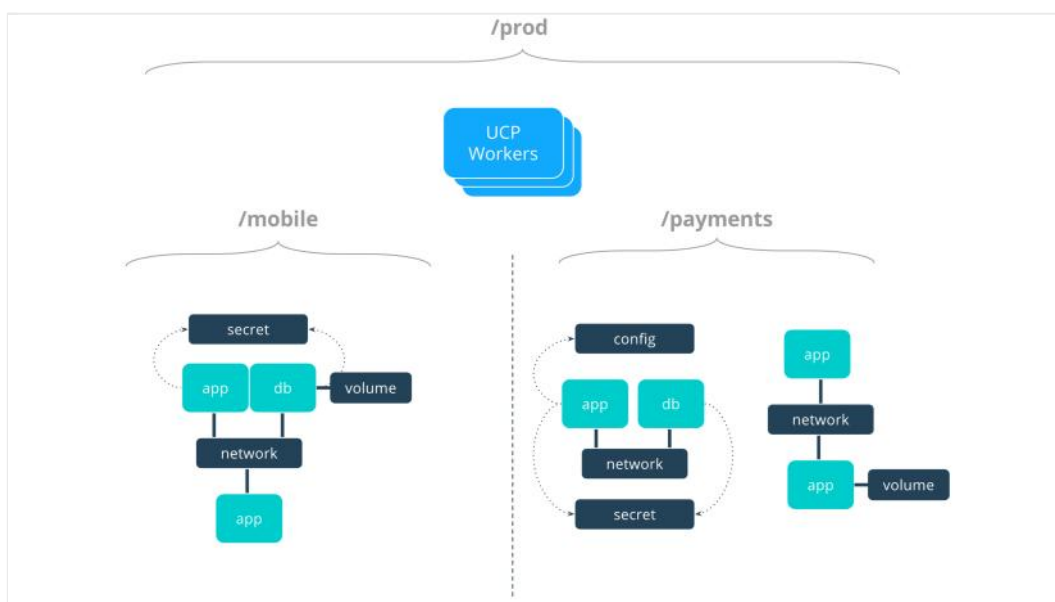
## Grant composition

The following four grants define the access policy for the entire organization for this cluster. They tie together the collections that were created, the default and custom roles, and also teams of users that are in UCP.



## Access architecture

The resulting access architecture defined by these grants is depicted below.



There are four teams that are given access to cluster resources:

- `security` can see, but not edit, all resources shown, as it has `View Only` access to the entire `/prod` collection.
- `ops` has `Full Control` against the entire `/prod` collection, giving it the capability to deploy, view, edit, and remove applications and application resources.
- `mobile` has the `Dev` role against the `/prod/mobile` collection. This team can see and `exec` in to their own applications, but can't see any of the `payments` applications.

- `payments` has the same type of access but for the `/prod/payments` collection.

See a deeper tutorial on how to design access control architectures.

(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/access-control-design-ee-standard/>)

Manage access to resources by using collections

(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/manage-access-with-collections/>).

## Transition from UCP 2.1 access control

- Your existing access labels and permissions are migrated automatically during an upgrade from UCP 2.1.x.
- Unlabeled “user-owned” resources are migrated into the user’s private collection, in `/Shared/Private/<username>` .
- Old access control labels are migrated into `/Shared/Legacy/<labelname>` .
- When deploying a resource, choose a collection instead of an access label.
- Use grants for access control, instead of unlabeled permissions.

## Where to go next

- Create and manage users  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/create-and-manage-users/>)
- Create and manage teams  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/create-and-manage-teams/>)
- Deploy a service with view-only access across an organization  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/deploy-view-only-service/>)
- Isolate volumes between two different teams  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/isolate-volumes-between-teams/>)
- Isolate swarm nodes between two different teams  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/isolate-nodes-between-teams/>)

ucp (<https://docs.docker.com/glossary/?term=ucp>), grant

(<https://docs.docker.com/glossary/?term=grant>), role

(<https://docs.docker.com/glossary/?term=role>), permission

(<https://docs.docker.com/glossary/?term=permission>), authentication

(<https://docs.docker.com/glossary/?term=authentication>), authorization  
(<https://docs.docker.com/glossary/?term=authorization>)

[\(/\)](#)[Create Docker ID \(https://cloud.docker.com/\)](https://cloud.docker.com/)[Sign In \(https://cloud.docker.com/login\)](https://cloud.docker.com/login)[All \(/\)](#)[Engineering \(/category/engineering\)](#)[Curated \(/curated/\)](#)[Docker Weekly \(/docker-weekly-archives/\)](#)[What is Docker? \(https://www.docker.com/what-docker\)](https://www.docker.com/what-docker)[Product \(https://www.docker.com/get-docker\)](https://www.docker.com/get-docker)[Community \(https://www.docker.com/docker-community\)](https://www.docker.com/docker-community)[Support \(https://success.docker.com/support\)](https://success.docker.com/support)

## TUTORIAL: ROLE BASED ACCESS CONTROL IN UNIVERSAL CONTROL PLANE

By [Vivek Saraswat \(https://blog.docker.com/author/vivek-saraswat/\)](https://blog.docker.com/author/vivek-saraswat/) March 18, 2016

<https://www.linkedin.com/shareArticle?mini=true&url=https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/>  
<https://www.reddit.com/submit?url=https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/&title=Tutorial%3A%20Role%20Based%20Access%20Control%20in%20Universal%20Control%20Plane>  
<https://plus.google.com/share?url=https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/>  
<http://news.ycombinator.com/submitlink?u=https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/&t=Tutorial%3A%20Role%20Based%20Access%20Control%20in%20Universal%20Control%20Plane>

[datacenter \(https://blog.docker.com/tag/datacenter/\)](https://blog.docker.com/tag/datacenter/), [docker \(https://blog.docker.com/tag/docker/\)](https://blog.docker.com/tag/docker/), [docker datacenter \(https://blog.docker.com/tag/docker-datacenter/\)](https://blog.docker.com/tag/docker-datacenter/), [Docker tutorial \(https://blog.docker.com/tag/docker-tutorial/\)](https://blog.docker.com/tag/docker-tutorial/), [tutorial \(https://blog.docker.com/tag/tutorial/\)](https://blog.docker.com/tag/tutorial/), [ucp \(https://blog.docker.com/tag/ucp/\)](https://blog.docker.com/tag/ucp/), [universal control plane \(https://blog.docker.com/tag/universal-control-plane/\)](https://blog.docker.com/tag/universal-control-plane/)

It's been an exciting time for us since we released [Docker Datacenter](https://www.docker.com/products/docker-datacenter) (<https://www.docker.com/products/docker-datacenter>) a couple of week ago, which included the GA of [Universal Control Plane](https://www.docker.com/products/docker-universal-control-plane) (<https://www.docker.com/products/docker-universal-control-plane>) (UCP), our on-premises or VPC deployable management and orchestration solution for containers. Thank you for the enthusiastic response, and we'll continue to deliver enterprise-grade feature sets into the Docker Datacenter platform.

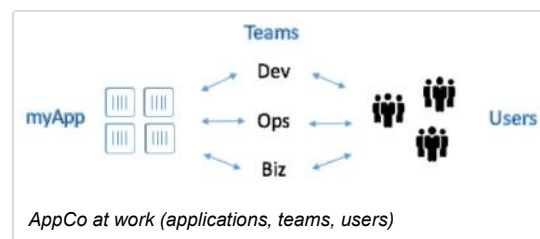
One such feature we delivered in UCP 1.0 was Role Based Access Control (RBAC)—deciding and enforcing who gets access to which resources. This is a really important and complex component of commercial IT infrastructure. This tutorial walks through how the feature is implemented in UCP and demonstrating it with an example.

### Why RBAC?

RBAC is important because it addresses the following questions for your infrastructure:

1. Which users get access to a resource?
2. What level of access do these users get to that resource?
3. How do I enforce this access within my environment?

The easiest way to understand RBAC is through an example. Let's say you are the system administrator for AppCo, which is developing a containerized application called myApp. You have three teams within your organization—a Dev team, an Ops team, and a Biz team—each of which interacts with myApp in a different way. Dev needs to build the app and test it on the cluster. Ops needs to deploy the application and access kernel-level resources on hosts. Biz wants to take a look at myApp from time to time.

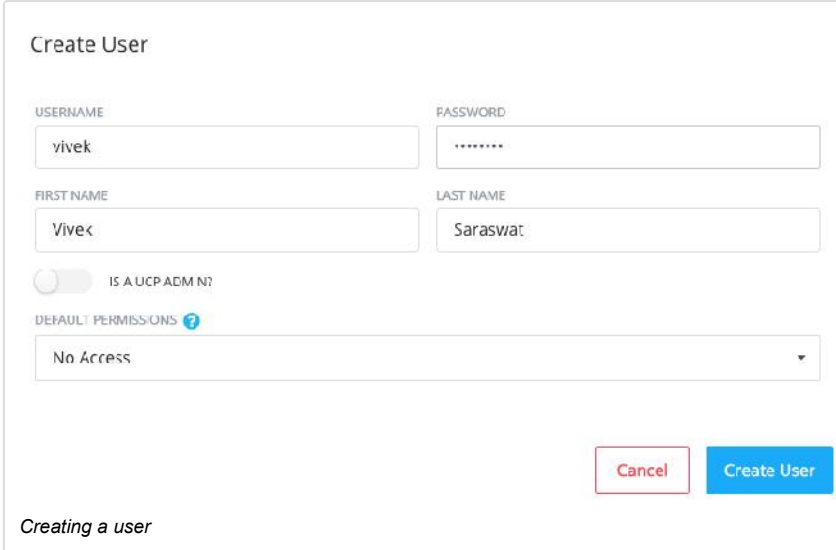




All of these teams need to access myApp, but how do you ensure that each team only gets the specific level of access it needs to get the job done? Let's walk through how RBAC works in UCP, and use AppCo along the way as an example.

## Users and Teams


The first step to setting up RBAC is to create your users. UCP users can either be created manually or imported via LDAP/AD integration, and come in two flavors—admins, and users. Admins have the ability to do everything within a UCP environment: They have the ability to access all resources, change UCP settings, manage users accounts, and set access permissions. Non-admin users, by default, have no permissions beyond what is granted by admins. They can change their password, but that's about it. In the case of AppCo, you are the plucky admin (the first admin account is created as part of installation) so you create non-admin user accounts for everybody else in the organization.



The 'Create User' form contains the following fields and controls:

- USERNAME:** Text input with the value 'yivek'.
- PASSWORD:** Password input field with masked characters '\*\*\*\*\*'.
- FIRST NAME:** Text input with the value 'Vivek'.
- LAST NAME:** Text input with the value 'Saraswat'.
- IS A UCP ADMIN?:** A toggle switch currently turned off.
- DEFAULT PERMISSIONS:** A dropdown menu showing 'No Access'.
- Buttons:** 'Cancel' (red outline) and 'Create User' (blue solid).
- Footer:** The text 'Creating a user'.

Now that you've created your users, you'll want to put them in teams. A team is a grouping of users for access control purposes. Teams can be manually created via UCP or can be synced through LDAP/AD integration. For AppCo, you've created three teams in UCP to match the real world ones: Dev, Ops, and Biz. You can then assign each user to their given teams. Note that a user can be a part of multiple teams at the same time.



The 'Adding users to a team' interface shows a table of users and their team assignments:

USERNAME	FIRST NAME	LAST NAME	Teams
yivek	Vivek	Saraswat	Dev, Ops, Biz
...	...	...	...

Below the table, there are buttons for 'Add User to Team' and 'Remove User from Team'.

Adding users to a team

## Levels of Permissions

Next, you want to figure out what level of permission to give to your users. UCP provides four distinct sets of permissions. This makes it easier for you to assign and understand user access without having to set individual permissions for each and every action possible. These four sets of permissions are:

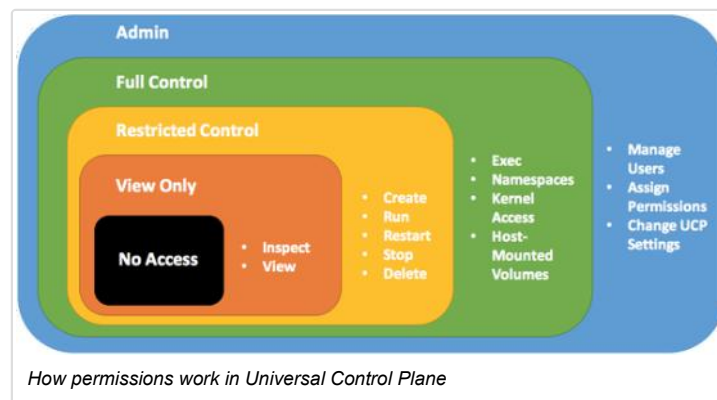
**Full Control:** Can do anything possible to resources. Create, restart, kill, view containers, etc.

This is the highest level of access a non-admin user can have.

**Restricted Control:** Similar to Full Control, but with restrictions around container exec, privileged containers, host-mounted volumes, and other particularly sensitive operations. This is best suited for when you want a group to run containers in production but not access kernel capabilities or modify a container using exec privileges.

**View Only:** Look, but don't touch. Can view and inspect resources, but nothing else.

**No Access:** Cannot view or otherwise access resources.



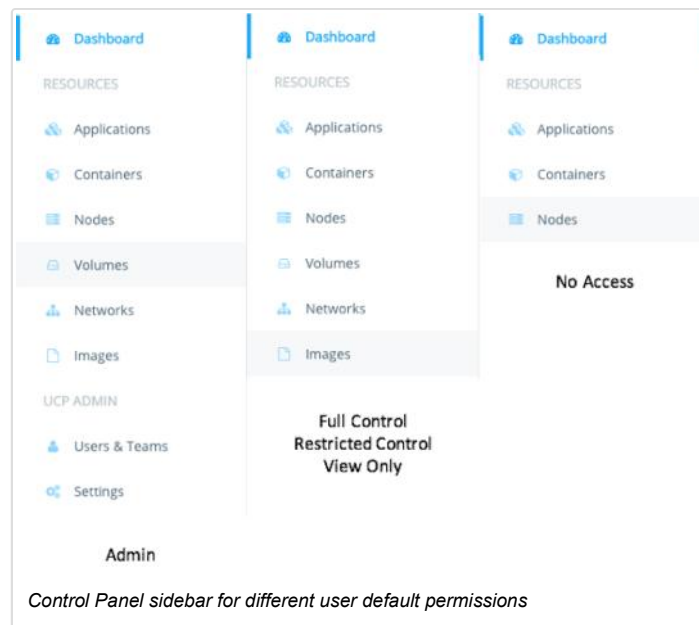
You can find more detailed information on permissions in the [UCP documentation](https://docs.docker.com/ucp/manage/monitor-manage-users/) (<https://docs.docker.com/ucp/manage/monitor-manage-users/>).

## Enforcing Coarse-Grained Access through User Default Permissions

UCP allows you to set and enforce resource access through two distinct methods: user-assigned default permissions, and team-assigned container labels. First we'll discuss default permissions.

Every user has a default permissions setting which is assigned at account creation, and can be edited by an admin at any time. The default permissions are enforced for all non-container resources such as images, networks, and volumes.

In AppCo, when you created the users they had the default setting of "No Access." Thus, each user won't even be able to see the tabs for images, networks, and volumes. Thinking again, you realize that the folks on the Ops team will need the ability to create and destroy non-container resources, so you change all of their user default permissions to "Full Control." The folks on the Dev and Biz team should be able to see these resources to understand the cluster deployment, so you change all of their user default permissions to "View Only."



## Enforcing Fine-Grained Access through Team Container Labels

Default permissions allows you to provide broad, coarse-grained access to resources. However, in most cases you will want to provide finer-grained access to specific containers. In addition, you may want to set preferences at a team level rather than individuals for easier organization and convenience. This is where label permissions come in.

In the case of AppCo, you (as an admin) realize that various teams are going to need different levels of access to any of the containerized components of the myApp application. Your first step is to ensure that anyone starting containers as a part of myApp uses an access label with the key `com.docker.ucp.access.label` and the value `myApp`. This can be done either through the UCP GUI (as shown below) or when running containers (e.g. `docker run -label com.docker.ucp.access.label="myApp"`).

Dashboard / Containers / Deploy

### Basic Settings

IMAGE NAME

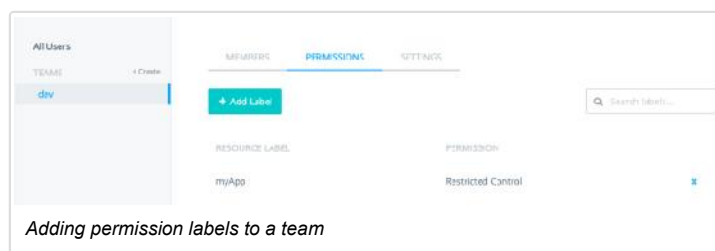
CONTAINER NAME

PERMISSIONS LABEL (COM.DOCKER.UCP.ACCESS.LABEL)

*Deploying an app with a permissions label*

Next, you go to Team UI screen and start changing the permissions of each team:

1. The Biz Team wants to see the myApp containers in action, but you don't think they need permission to change the application. So, you give them "View Only" access to the "myApp" label.
2. The Dev Team needs to be able to start and stop myApp containers, but you don't want them to have edit currently running containers or get kernel-level access to the hosts. So, you give them "Restricted Control" access to the "myApp" label.
3. The Ops Team needs the freedom to make changes to the application as necessary. So, you give them "Full Control" access to the "myApp" label.



Keep in mind that any user who is a part of multiple teams with access to the same label gets the best possible access out of those teams. From the AppCo example, let's say the admin added user Jane to both the Dev and Ops teams. The Ops team has Full Control and Dev team has Restricted Control access to the myApp label. Thus, Jane has Full Control to the myApp label, which is the higher access out of those two teams. In addition, a non-labeled container will be visible only to the user who created that container, as well as to all admins.

## Putting It All Together

Now, all the users of AppCo have both user-assigned default permissions and well as team-assigned label permissions. If a user tries to do an action that they do not have permission for, he or she will get an "access denied" error. This occurs whether the user tries this in the UCP GUI or in the CLI via Docker Client.



We hope you've found this overview of Role-Based Access Control useful. We're very excited about RBAC and look forward to building more enterprise-grade features for Universal Control Plane in the future. If you have any questions please feel free to discuss [on the UCP forums](https://forums.docker.com/c/commercial-products/ucp). (<https://forums.docker.com/c/commercial-products/ucp>)

## Additional Resources:

- Try a [free 30-day trial](https://hub.docker.com/enterprise/trial/) (<https://hub.docker.com/enterprise/trial/>) of Docker Datacenter
- Get [training](https://training.docker.com/instructor-led-training/docker-datacenter-remote-training-series) (<https://training.docker.com/instructor-led-training/docker-datacenter-remote-training-series>) on Docker Datacenter

- Register (<https://goto.docker.com/Webinar-Docker-Datacenter-Demo.html>) for the next Docker Datacenter demo

## Learn More about Docker

New to Docker? Try our 10 min [online tutorial](#)

(<https://docs.docker.com/engine/understanding-docker/>)

Share images, automate builds, and more with a [free Docker Hub account](#)

(<http://hub.docker.com/>)

Read the Docker [1.10 Release Notes](#) (<http://docs.docker.com/release-notes/>)

Subscribe to [Docker Weekly](#) ([https://www.docker.com/subscribe\\_newsletter/](https://www.docker.com/subscribe_newsletter/))

Sign up for upcoming [Docker Online Meetups](#) (<http://www.meetup.com/Docker-Online-Meetup/>)

Attend upcoming [Docker Meetups](#) (<https://www.docker.com/community/meetups/>)

Register for [DockerCon 2016](#) (<http://2016.dockercon.com/>)

Watch [DockerCon EU 2015 videos](#) ([https://www.youtube.com/playlist?](https://www.youtube.com/playlist?list=PLkA60AVN3hh87OoVra6MHf2L4UR9xwIkv)

[list=PLkA60AVN3hh87OoVra6MHf2L4UR9xwIkv](https://www.youtube.com/playlist?list=PLkA60AVN3hh87OoVra6MHf2L4UR9xwIkv))

Start [contributing to Docker](#) (<https://docs.docker.com/contributing/contributing/>)

[datacenter](https://blog.docker.com/tag/datacenter/) (<https://blog.docker.com/tag/datacenter/>), [docker](https://blog.docker.com/tag/docker/)

(<https://blog.docker.com/tag/docker/>), [docker datacenter](https://blog.docker.com/tag/docker-datacenter/) (<https://blog.docker.com/tag/docker-datacenter/>),

[Docker tutorial](https://blog.docker.com/tag/docker-tutorial/) (<https://blog.docker.com/tag/docker-tutorial/>), [tutorial](https://blog.docker.com/tag/tutorial/)

(<https://blog.docker.com/tag/tutorial/>), [ucp](https://blog.docker.com/tag/ucp/) (<https://blog.docker.com/tag/ucp/>), [universal control plane](https://blog.docker.com/tag/universal-control-plane/)

(<https://blog.docker.com/tag/universal-control-plane/>)



### TUTORIAL: ROLE BASED ACCESS CONTROL IN UNIVERSAL CONTROL PLANE

By [Vivek Saraswat](https://blog.docker.com/author/vivek-saraswat/) (<https://blog.docker.com/author/vivek-saraswat/>)

Vivek works in product management at Docker, where he helps enterprise customers to manage their containerized applications. Prior to Docker, Vivek held product roles at VMware and AWS and was an engineer in a past life. He also enjoys singing and gaming in his spare time. Vivek tweets at [@theVSaraswat](https://twitter.com/theVSaraswat) (<https://twitter.com/theVSaraswat>).

## 4 Responses to "Tutorial: Role Based Access Control in Universal Control Plane"



### Ravi Upad

[January 6, 2017](https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/#comment-380184) (<https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/#comment-380184>)

Do we need to create the tag com.docker.ucp.access.label even at services level?

[Reply](#)



### Dhaval

[June 27, 2017](https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/#comment-402684) (<https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/#comment-402684>)

It's not available for open source ?

[Reply](#)



**Victor Coisne**

July 10, 2017 (<https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/#comment-404261>)

No this is part of Docker Enterprise Edition (Docker EE)

[Reply](#)



**Chris**

October 26, 2017 (<https://blog.docker.com/2016/03/role-based-access-control-docker-ucp-tutorial/#comment-414928>)

Why would a user who has access to a collection like /SharedProd/app1/secrets-and-volumes not be able to create a service with a local volume? I have given the ops team in UCP for app1 restricted control to that collection. Thanks

[Reply](#)

## Leave a Reply





Submit Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

## Related Posts



[Docker Weekly | Roundup](https://blog.docker.com/2016/08/docker-weekly-round-up/)

(<https://blog.docker.com/2016/08/docker-weekly-round-up/>)

By [Cari Scardina](https://blog.docker.com/author/cari-scardina/)

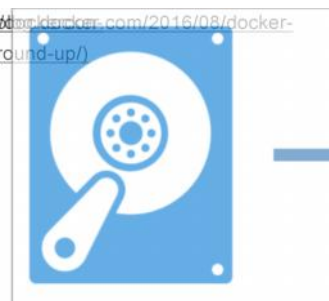
(<https://blog.docker.com/author/cari-scardina/>)

| August 12, 2016

[Docker weekly roundup](https://blog.docker.com/tag/docker-weekly-roundup/)

(<https://blog.docker.com/tag/docker-weekly-roundup/>), [roundup](https://blog.docker.com/tag/docker-weekly-roundup/)

(<https://blog.docker.com/2015/12/dockercon-eu-2015-wild-card-day1/>) (<https://blog.docker.com/2016/08/docker-weekly-round-up/>)



[Docker for Windows Server and Image2Docker](https://blog.docker.com/2017/01/docker-for-windows-server-image2docker/)

(<https://blog.docker.com/2017/01/docker-for-windows-server-image2docker/>)

By [Elton Stoneman](https://blog.docker.com/author/elton-stoneman/)

(<https://blog.docker.com/author/elton-stoneman/>)

| January 17, 2017

[ASP.NET](https://blog.docker.com/tag/asp-net/)

(<https://blog.docker.com/tag/asp-net/>)

[DockerCon EU 2015 Videos: Day 1](https://blog.docker.com/2015/12/dockercon-eu-2015-wild-card-day1/)

(<https://blog.docker.com/2015/12/dockercon-eu-2015-wild-card-day1/>)

By [Victor Coisne](https://blog.docker.com/author/victor-coisne/)

(<https://blog.docker.com/author/victor-coisne/>)

| December 7, 2015

[dockercon](https://blog.docker.com/tag/dockercon/)

(<https://blog.docker.com/tag/dockercon/>),

[\(https://blog.docker.com/tag/roundup/\)](https://blog.docker.com/tag/roundup/),  
[weekly roundup](https://blog.docker.com/tag/weekly-roundup/)  
[\(https://blog.docker.com/tag/weekly-roundup/\)](https://blog.docker.com/tag/weekly-roundup/)

[dockercon eu](https://blog.docker.com/tag/dockercon-eu/)  
[\(https://blog.docker.com/tag/dockercon-eu/\)](https://blog.docker.com/tag/dockercon-eu/), [DockerCon Europe](https://blog.docker.com/tag/dockercon-europe/)  
[\(https://blog.docker.com/tag/dockercon-europe/\)](https://blog.docker.com/tag/dockercon-europe/), [Dockercon Speakers](https://blog.docker.com/tag/dockercon-speakers/)  
[\(https://blog.docker.com/tag/dockercon-speakers/\)](https://blog.docker.com/tag/dockercon-speakers/), [videos](https://blog.docker.com/tag/videos/)  
[\(https://blog.docker.com/tag/videos/\)](https://blog.docker.com/tag/videos/)

[net/\), docker for windows](https://blog.docker.com/tag/docker-for-windows/)  
[\(https://blog.docker.com/tag/docker-for-windows/\)](https://blog.docker.com/tag/docker-for-windows/), [docker on windows](https://blog.docker.com/tag/docker-on-windows/)  
[\(https://blog.docker.com/tag/docker-on-windows/\)](https://blog.docker.com/tag/docker-on-windows/), [image2docker](https://blog.docker.com/tag/image2docker/)  
[\(https://blog.docker.com/tag/image2docker/\)](https://blog.docker.com/tag/image2docker/), [Microsoft windows server](https://blog.docker.com/tag/microsoft-windows-server/)  
[\(https://blog.docker.com/tag/microsoft-windows-server/\)](https://blog.docker.com/tag/microsoft-windows-server/), [SQL server](https://blog.docker.com/tag/sql-server/)  
[\(https://blog.docker.com/tag/sql-server/\)](https://blog.docker.com/tag/sql-server/), [Windows Server Containers](https://blog.docker.com/tag/windows-server-containers/)  
[\(https://blog.docker.com/tag/windows-server-containers/\)](https://blog.docker.com/tag/windows-server-containers/)

## Get the Latest Docker News by Email

Docker Weekly is a newsletter with the latest content on Docker and the agenda for the upcoming weeks.

Subscribe to our newsletter

Company Email	Select Country...
<div>Submit</div>	

Sign up for our newsletter.

Company Email	Select Country...	Submit
---------------	-------------------	--------

### What is Docker

[\(https://www.docker.com/what-docker/\)](https://www.docker.com/what-docker/)

### What is a Container

[\(https://www.docker.com/what-container/\)](https://www.docker.com/what-container/)

### Use Cases

[\(https://www.docker.com/use-cases/\)](https://www.docker.com/use-cases/)

### Customers

[\(https://www.docker.com/customers/\)](https://www.docker.com/customers/)

### For Government

[\(https://www.docker.com/industry-government/\)](https://www.docker.com/industry-government/)

### For IT Pros

[\(https://www.docker.com/itpro/\)](https://www.docker.com/itpro/)

### Product

[\(https://www.docker.com/get-docker/\)](https://www.docker.com/get-docker/)

### Pricing

[\(https://www.docker.com/pricing/\)](https://www.docker.com/pricing/)

### Community Edition

[\(https://www.docker.com/community-edition/\)](https://www.docker.com/community-edition/)

### Enterprise Edition

[\(https://www.docker.com/enterprise-edition/\)](https://www.docker.com/enterprise-edition/)

### Docker Cloud

[\(https://cloud.docker.com/\)](https://cloud.docker.com/)

### Docker Store

[\(https://store.docker.com/\)](https://store.docker.com/)

### Documentation

[\(https://docs.docker.com/\)](https://docs.docker.com/)

### Blog (/)

[\(https://www.docker.com/blog/\)](https://www.docker.com/blog/)

### RSS Feed (/feed/)

[\(https://www.docker.com/feed/\)](https://www.docker.com/feed/)

### Knowledge Base

[\(https://success.docker.com/kbase/\)](https://success.docker.com/kbase/)

### Resources

[\(https://www.docker.com/products/resources/\)](https://www.docker.com/products/resources/)

### Community

[\(https://www.docker.com/docker-community/\)](https://www.docker.com/docker-community/)

### Open Source

[\(https://www.docker.com/technologies/overview/\)](https://www.docker.com/technologies/overview/)

[Forums \(https://forums.docker.com/\)](https://forums.docker.com/)

### Docker Captains

[\(https://www.docker.com/community/docker-captains/\)](https://www.docker.com/community/docker-captains/)

### Scholarships

[\(https://www.docker.com/docker-community/scholarships/\)](https://www.docker.com/docker-community/scholarships/)

[Community News \(/curated/\)](https://www.docker.com/community/news/)

[Find a Partner](#)

(<https://www.docker.com/find-partner>)

[Become a Partner](#)

(<https://www.docker.com/partners/partner-program>)

[About Docker](#)

(<https://www.docker.com/company>)

[Management](#)

(<https://www.docker.com/company/management>)

[Press & News](#)

(<https://www.docker.com/company/news-and-press>)

[Careers](#)

(<https://www.docker.com/careers>)

[Status \(http://status.docker.com/\)](http://status.docker.com/) [Security \(https://docker.com/docker-security\)](https://docker.com/docker-security) [Legal \(https://www.docker.com/legal\)](https://www.docker.com/legal)

[Contact \(https://www.docker.com/company/contact\)](https://www.docker.com/company/contact)

---

Copyright © 2018 Docker Inc. All rights reserved.



4



# Integrate with an LDAP Directory

*Estimated reading time: 12 minutes*

✔ These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

2.2.14 ▼

Docker UCP integrates with LDAP directory services, so that you can manage users and groups from your organization's directory and it will automatically propagate that information to UCP and DTR.

If you enable LDAP, UCP uses a remote directory server to create users automatically, and all logins are forwarded to the directory server.

When you switch from built-in authentication to LDAP authentication, all manually created users whose usernames don't match any LDAP search results are still available.

When you enable LDAP authentication, you can choose whether UCP creates user accounts only when users log in for the first time. Select the Just-In-Time User Provisioning option to ensure that the only LDAP accounts that exist in UCP are those that have had a user log in to UCP.

## How UCP integrates with LDAP

You control how UCP integrates with LDAP by creating searches for users. You can specify multiple search configurations, and you can specify multiple LDAP servers to integrate with. Searches start with the `Base DN`, which is the *distinguished name* of the node in the LDAP directory tree where the search starts looking for users.

Access LDAP settings by navigating to the Authentication & Authorization page in the UCP web UI. There are two sections for controlling LDAP searches and servers.

- LDAP user search configurations: This is the section of the Authentication & Authorization page where you specify search parameters, like `Base DN`, `scope`, `filter`, the `username` attribute, and the `full name` attribute. These searches are stored in a list, and the ordering may be important, depending on your search configuration.
- LDAP server: This is the section where you specify the URL of an LDAP server, TLS configuration, and credentials for doing the search requests. Also, you provide a domain for all servers but the first one. The first server is considered the default domain server. Any others are associated with the domain that you specify in the page.

Here's what happens when UCP synchronizes with LDAP:

1. UCP creates a set of search results by iterating over each of the user search configs, in the order that you specify.
2. UCP chooses an LDAP server from the list of domain servers by considering the `Base DN` from the user search config and selecting the domain server that has the longest domain suffix match.
3. If no domain server has a domain suffix that matches the `Base DN` from the search config, UCP uses the default domain server.
4. UCP combines the search results into a list of users and creates UCP accounts for them. If the Just-In-Time User Provisioning option is set, user accounts are created only when users first log in.

The domain server to use is determined by the `Base DN` in each search config. UCP doesn't perform search requests against each of the domain servers, only the one which has the longest matching domain suffix, or the default if there's no match.

Here's an example. Let's say we have three LDAP domain servers:

Domain	Server URL
<i>default</i>	ldaps://ldap.example.com
<code>dc=subsidiary1,dc=com</code>	ldaps://ldap.subsidiary1.com
<code>dc=subsidiary2,dc=subsidiary1,dc=com</code>	ldaps://ldap.subsidiary2.com

Here are three user search configs with the following `Base DN`s :

- baseDN= `ou=people,dc=subsidiary1,dc=com`

For this search config, `dc=subsidiary1,dc=com` is the only server with a domain which is a suffix, so UCP uses the server

`ldaps://ldap.subsidiary1.com` for the search request.

- baseDN= `ou=product,dc=subsidiary2,dc=subsidiary1,dc=com`

For this search config, two of the domain servers have a domain which is a suffix of this base DN, but `dc=subsidiary2,dc=subsidiary1,dc=com` is the longer of the two, so UCP uses the server `ldaps://ldap.subsidiary2.com` for the search request.

- baseDN= `ou=eng,dc=example,dc=com`

For this search config, there is no server with a domain specified which is a suffix of this base DN, so UCP uses the default server,

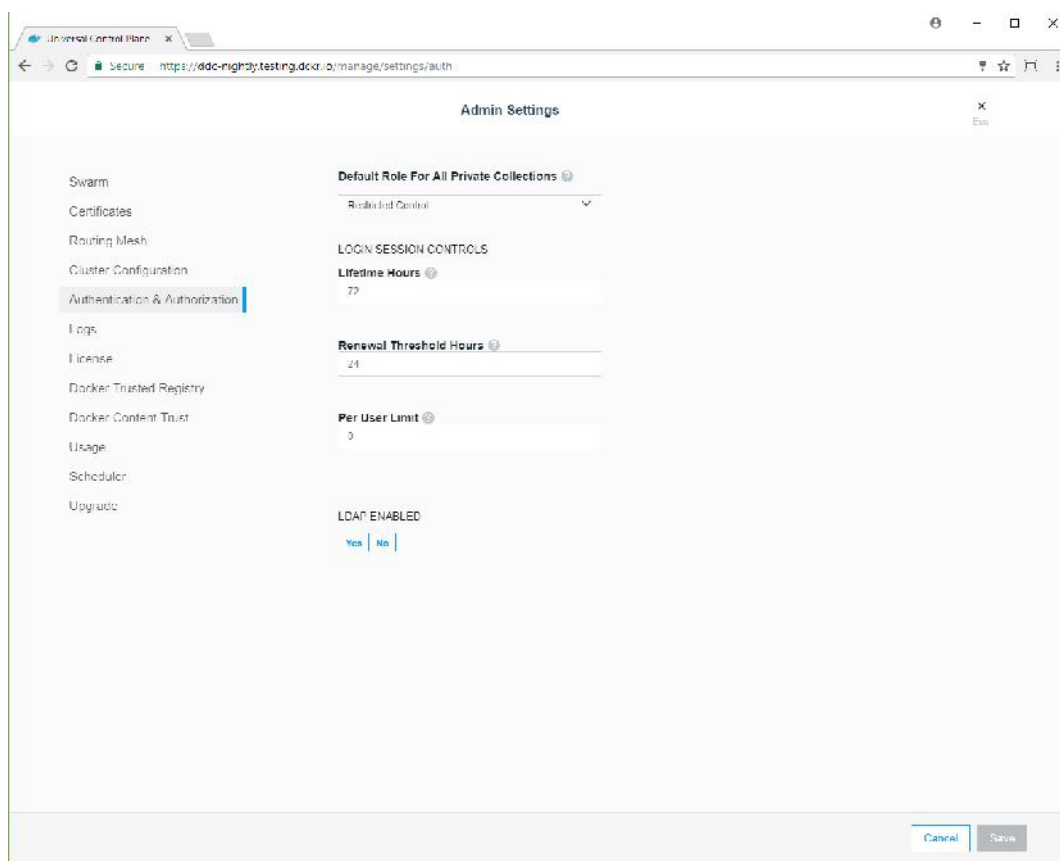
`ldaps://ldap.example.com` , for the search request.

If there are `username` collisions for the search results between domains, UCP uses only the first search result, so the ordering of the user search configs may be important. For example, if both the first and third user search configs result in a record with the username `jane.doe` , the first has higher precedence and the second is ignored. For this reason, it's important to choose a `username` attribute that's unique for your users across all domains.

Because names may collide, it's a good idea to use something unique to the subsidiary, like the email address for each person. Users can log in with the email address, for example, `jane.doe@subsidiary1.com` .

## Configure the LDAP integration

To configure UCP to create and authenticate users by using an LDAP directory, go to the UCP web UI, navigate to the Admin Settings page and click Authentication & Authorization to select the method used to create and authenticate users.



In the LDAP Enabled section, click Yes to The LDAP settings appear. Now configure your LDAP directory integration.

## Default role for all private collections

Use this setting to change the default permissions of new users.

Click the dropdown to select the permission level that UCP assigns by default to the private collections of new users. For example, if you change the value to **View Only**, all users who log in for the first time after the setting is changed have **View Only** access to their private collections, but permissions remain unchanged for all existing users. Learn more about permission levels (<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/permission-levels/>).

## LDAP enabled

Click Yes to enable integrating UCP users and teams with LDAP servers.

## LDAP server

Field	Description
LDAP server URL	The URL where the LDAP server can be reached.
Reader DN	The distinguished name of the LDAP account used for searching entries in the LDAP server. As a best practice, this should be an LDAP read-only user.
Reader password	The password of the account used for searching entries in the LDAP server.
Use Start TLS	Whether to authenticate/encrypt the connection after connecting to the LDAP server over TCP. If you set the LDAP Server URL field with <code>ldaps://</code> , this field is ignored.
Skip TLS verification	Whether to verify the LDAP server certificate when using TLS. The connection is still encrypted but vulnerable to man-in-the-middle attacks.
No simple pagination	If your LDAP server doesn't support pagination.
Just-In-Time User Provisioning	Whether to create user accounts only when users log in for the first time. The default value of <code>true</code> is recommended. If you upgraded from UCP 2.0.x, the default is <code>false</code> .

Click Confirm to add your LDAP domain.

To integrate with more LDAP servers, click Add LDAP Domain.

## LDAP user search configurations

Field	Description
Base DN	The distinguished name of the node in the directory tree where the search should start looking for users.
Username attribute	The LDAP attribute to use as username on UCP. Only user entries with a valid username will be created. A valid username is no longer than 100 characters and does not contain any unprintable characters, whitespace characters, or any of the following characters: <code>/ \ [ ] : ;   = , + * ? &lt; &gt; ' " .</code>
Full name attribute	The LDAP attribute to use as the user's full name for display purposes. If left empty, UCP will not create new users with a full name value.

Field	Description
Filter	The LDAP search filter used to find users. If you leave this field empty, all directory entries in the search scope with valid username attributes are created as users.
Search subtree instead of just one level	Whether to perform the LDAP search on a single level of the LDAP tree, or search through the full LDAP tree starting at the Base DN.
Select Group Members	Whether to further filter users by selecting those who are also members of a specific group on the directory server. This feature is helpful if the LDAP server does not support <code>memberOf</code> search filters.
Iterate through group members	If <code>Select Group Members</code> is selected, this option searches for users by first iterating over the target group's membership, making a separate LDAP query for each member, as opposed to first querying for all users which match the above search query and intersecting those with the set of group members. This option can be more efficient in situations where the number of members of the target group is significantly smaller than the number of users which would match the above search filter, or if your directory server does not support simple pagination of search results.
Group DN	If <code>Select Group Members</code> is selected, this specifies the distinguished name of the group from which to select users.
Group Member Attribute	If <code>Select Group Members</code> is selected, the value of this group attribute corresponds to the distinguished names of the members of the group.

Universal Control Plane x

Secure | <https://doc-nightly.qe.aws.cn/ucp/manage/settings/auth>

### LDAP USER SEARCH CONFIGURATIONS [Add LDAP User Search Configuration +](#)

**Base DN**

**Username Attribute**

**Fullname Attribute**

**Filter**

☐ Search subtree instead of just one level

☐ Select Group Members

☐ Iterate through group members

**Group DN**

**Group Member Attribute**

[Confirm](#) [Cancel](#)

No LDAP User Search Configurations defined yet

**LDAP TEST LOGIN**

[Cancel](#) [Save](#)

To configure more user search queries, click **Add LDAP User Search Configuration** again. This is useful in cases where users may be found in multiple distinct subtrees of your organization's directory. Any user entry which matches at least one of the search configurations will be synced as a user.

## LDAP test login

Field	Description
Username	An LDAP username for testing authentication to this application. This value corresponds with the Username Attribute specified in the LDAP user search configurations section.
Password	The user's password used to authenticate (BIND) to the directory server.

Before you save the configuration changes, you should test that the integration is correctly configured. You can do this by providing the credentials of an LDAP user, and clicking the **Test** button.



## LDAP sync configuration

Field	Description
Sync interval	The interval, in hours, to synchronize users between UCP and the LDAP server. When the synchronization job runs, new users found in the LDAP server are created in UCP with the default permission level. UCP users that don't exist in the LDAP server become inactive.
Enable sync of admin users	This option specifies that system admins should be synced directly with members of a group in your organization's LDAP directory. The admins will be synced to match the membership of the group. The configured recovery admin user will also remain a system admin.

Once you've configured the LDAP integration, UCP synchronizes users based on the interval you've defined starting at the top of the hour. When the synchronization runs, UCP stores logs that can help you troubleshoot when something goes wrong.

You can also manually synchronize users by clicking Sync Now.

## Revoke user access

When a user is removed from LDAP, the effect on the user's UCP account depends on the Just-In-Time User Provisioning setting:

- Just-In-Time User Provisioning is `false` : Users deleted from LDAP become inactive in UCP after the next LDAP synchronization runs.
- Just-In-Time User Provisioning is `true` : Users deleted from LDAP can't authenticate, but their UCP accounts remain active. This means that they can use their client bundles to run commands. To prevent this, deactivate their UCP user accounts.

## Data synced from your organization's LDAP directory

UCP saves a minimum amount of user data required to operate. This includes the value of the username and full name attributes that you have specified in the configuration as well as the distinguished name of each synced user. UCP does not store any additional data from the directory server.

## Sync teams

UCP enables syncing teams with a search query or group in your organization's LDAP directory. Sync team members with your organization's LDAP directory (<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/create-and-manage-teams/>).

## Where to go next

- Create and manage users (<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/create-and-manage-users/>)
- Create and manage teams (<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/create-and-manage-teams/>)
- UCP permission levels (<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/permission-levels/>)
- Enable LDAP integration by using a configuration file (<https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/external-auth/enable-ldap-config-file/>)

LDAP (<https://docs.docker.com/glossary/?term=LDAP>), UCP (<https://docs.docker.com/glossary/?term=UCP>), authentication (<https://docs.docker.com/glossary/?term=authentication>), user management ([https://docs.docker.com/glossary/?term=user management](https://docs.docker.com/glossary/?term=user%20management))

# CLI-based access

*Estimated reading time: 3 minutes*

✔ These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

2.2.14 ▼

Docker UCP secures your swarm by using role-based access control, so that only authorized users can perform changes to the cluster.

For this reason, when running docker commands on a UCP node, you need to authenticate your request with client certificates. When trying to run docker commands without a valid certificate, you get an authentication error:

```
docker ps
```

```
x509: certificate signed by unknown authority
```

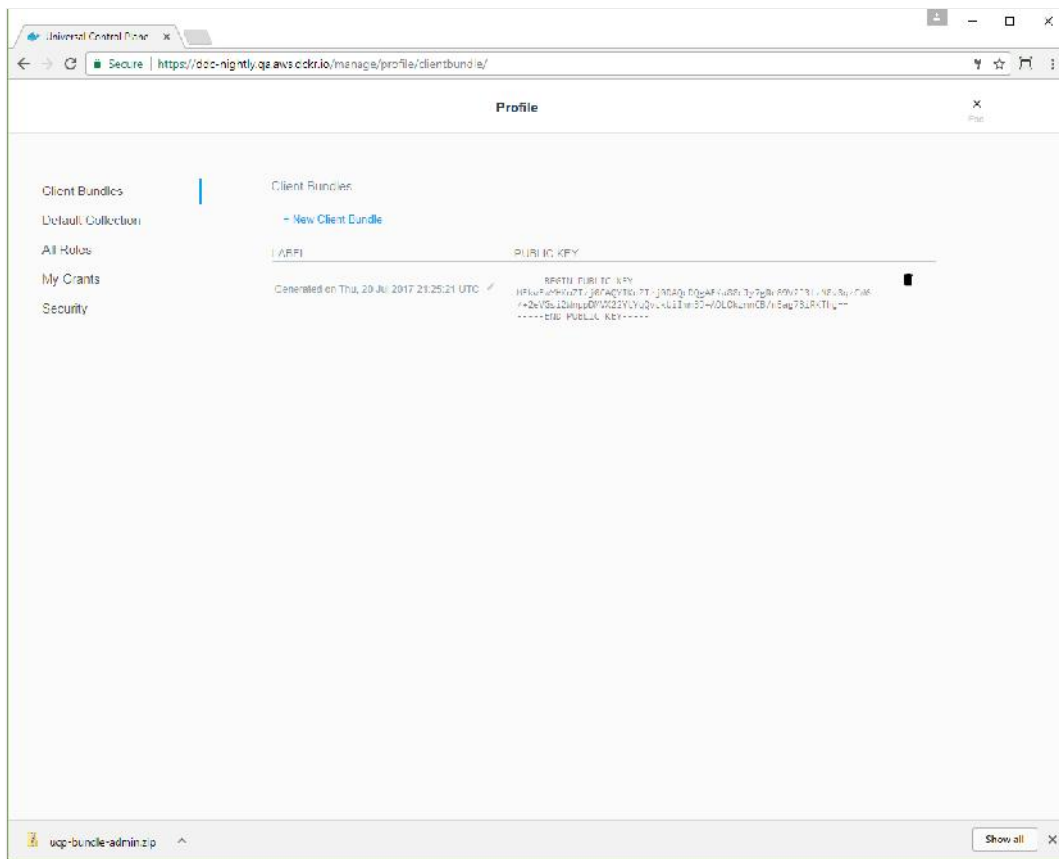
There are two different types of client certificates:

- Admin user certificate bundles: allow running docker commands on the Docker Engine of any node,
- User certificate bundles: only allow running docker commands through a UCP manager node.

## Download client certificates

To download a client certificate bundle, log in to the UCP web UI and navigate to your My Profile page.

In the left pane, click Client Bundles and click New Client Bundle to download the certificate bundle.



## Use client certificates

Once you've downloaded a client certificate bundle to your local computer, you can use it to authenticate your requests.

Navigate to the directory where you downloaded the user bundle, and unzip it. Then source the `env.sh` script.

```
unzip ucp-bundle-dave.lauper.zip
eval "$(<env.sh)"
```

The `env.sh` script updates the `DOCKER_HOST` environment variable to make your local Docker CLI communicate with UCP. It also updates the `DOCKER_CERT_PATH` environment variable to use the client certificates that are included in the client bundle you downloaded.

Note: The bundle includes scripts for setting up Windows nodes. To set up a Windows environment, run `env.cmd` in an elevated command prompt, or run `env.ps1` in an elevated PowerShell prompt.

To verify a client certificate bundle has been loaded and the client is successfully communicating with UCP, look for `ucp` in the `Server Version` returned by `docker version`.

```
docker version --format '{{.Server.Version}}'
ucp/2.2.14
```

From now on, when you use the Docker CLI client, it includes your client certificates as part of the request to the Docker Engine. You can now use the Docker CLI to create services, networks, volumes, and other resources on a swarm that's managed by UCP.

## Download client certificates by using the REST API

You can also download client bundles by using the UCP REST API (<https://docs.docker.com/datacenter/ucp/2.2/reference/api/>). In this example, we use `curl` to make the web requests to the API, and `jq` to parse the responses.

To install these tools on a Ubuntu distribution, you can run:

```
sudo apt-get update && sudo apt-get install curl jq
```

Then you get an authentication token from UCP, and use it to download the client certificates.

```
# Create an environment variable with the user security token
AUTHTOKEN=$(curl -sk -d '{"username": "<username>", "password": "<password>"}' https://<ucp-ip>/auth/login | jq -r .auth_token)

# Download the client certificate bundle
curl -k -H "Authorization: Bearer $AUTHTOKEN" https://<ucp-ip>/api/clientbundle -o bundle.zip
```

On Windows Server 2016, open an elevated PowerShell prompt and run:

```
$AUTHTOKEN=((Invoke-WebRequest -Body '{"username": "<username>", "password": "<password>"}' -Uri https://`<ucp-ip>/auth/login -Method POST).Content)|ConvertFrom-Json|select auth_token -ExpandProperty auth_token

[i o. file]::WriteAllBytes("ucp-bundle.zip", ((Invoke-WebRequest -Uri https://`<ucp-ip>/api/clientbundle -Headers @{"Authorization"="Bearer $AUTHTOKEN"}).Content))
```

## Where to go next

- Access the UCP web UI  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/user/access-ucp/>)

ucp (<https://docs.docker.com/glossary/?term=ucp>), cli  
(<https://docs.docker.com/glossary/?term=cli>), administration  
(<https://docs.docker.com/glossary/?term=administration>)

[\(/\)](#) [Create Docker ID \(https://cloud.docker.com/\)](https://cloud.docker.com/)

[All \(/\)](#) [Engineering \(/category/engineering\)](/category/engineering) [Curated \(/curated/\)](/curated/)  
[Sign In \(https://cloud.docker.com/login\)](https://cloud.docker.com/login)

[Docker Weekly \(/docker-weekly-archives/\)](/docker-weekly-archives/)

[What is Docker? \(https://www.docker.com/what-docker\)](https://www.docker.com/what-docker)

[Product \(https://www.docker.com/get-docker\)](https://www.docker.com/get-docker)

[Community \(https://www.docker.com/docker-community\)](https://www.docker.com/docker-community)



[Support \(https://success.docker.com/support\)](https://success.docker.com/support)

## GET FAMILIAR WITH DOCKER ENTERPRISE EDITION CLIENT BUNDLES

By [Brian Kaufman \(https://blog.docker.com/author/brian-kaufman/\)](https://blog.docker.com/author/brian-kaufman/)

September 20, 2017

 [in](http://www.linkedin.com/shareArticle?mini=true&url=https://blog.docker.com/2017/09/get-familiar-with-docker-enterprise-edition-client-bundles/) (http://www.linkedin.com/shareArticle?mini=true&url=https://blog.docker.com/2017/09/get-familiar-with-docker-enterprise-edition-client-bundles/)  [\(https://www.facebook.com/sharer/sharer.php?u=https://blog.docker.com/2017/09/get-familiar-with-docker-enterprise-edition-client-bundles/\)](https://www.facebook.com/sharer/sharer.php?u=https://blog.docker.com/2017/09/get-familiar-with-docker-enterprise-edition-client-bundles/)  [\(https://plus.google.com/share?url=https://blog.docker.com/2017/09/get-familiar-with-docker-enterprise-edition-client-bundles/\)](https://plus.google.com/share?url=https://blog.docker.com/2017/09/get-familiar-with-docker-enterprise-edition-client-bundles/)  [Y](http://news.ycombinator.com/submitlink?u=https://blog.docker.com/2017/09/get-familiar-docker-enterprise-edition-client-bundles/&t=Get%20Familiar%20with%20Docker%20Enterprise%20Edition%20Client%20Bundles) (http://news.ycombinator.com/submitlink?u=https://blog.docker.com/2017/09/get-familiar-docker-enterprise-edition-client-bundles/&t=Get%20Familiar%20with%20Docker%20Enterprise%20Edition%20Client%20Bundles)

[Client Bundles \(https://blog.docker.com/tag/client-bundles/\)](https://blog.docker.com/tag/client-bundles/), [Docker EE](#)

[\(https://blog.docker.com/tag/docker-ee/\)](https://blog.docker.com/tag/docker-ee/), [Docker Enterprise Edition](#)

[\(https://blog.docker.com/tag/docker-enterprise-edition/\)](https://blog.docker.com/tag/docker-enterprise-edition/), [ucp \(https://blog.docker.com/tag/ucp/\)](https://blog.docker.com/tag/ucp/),

[universal control plane \(https://blog.docker.com/tag/universal-control-plane/\)](https://blog.docker.com/tag/universal-control-plane/)

Docker Enterprise Edition (EE) is the only Containers as a Service (CaaS) Platform for IT that manages and secures diverse applications across disparate infrastructure, both on-premises and in the cloud.

There's a little mentioned big feature in Docker Enterprise Edition (EE) that seems to always bring smiles to the room once it's displayed.

~~All (/) Engineering (/category/engineering) - Curated (/curated)~~  
~~Before I tell you about it, let me first describe the use case. You're a~~  
sysadmin managing a Docker cluster and you have the following  
~~Docker Weekly (/docker-weekly-archives/)~~  
requirements:

- Different individuals in your LDAP/AD need various levels of access to the containers/services in your cluster
- Some users need to be able to go inside the running containers.
- Some users just need to be able to see the logs
- You do NOT want to give SSH access to each host in your cluster.

Now, how do you achieve this? The answer, or feature rather, is a client bundle. When you do a *docker version* command you will see two entries. The client portion of the engine is able to connect to a local server AND a remote once a client bundle is invoked.

```
Documents $docker version
Client:
Version:      17.06.2-ce
API version:  1.30
Go version:   go1.8.3
Git commit:   cec0b72
Built:        Tue Sep  5 20:12:06 2017
OS/Arch:      darwin/amd64

Server:
Version:      17.06.2-ce
API version:  1.30 (minimum version 1.12)
Go version:   go1.8.3
Git commit:   cec0b72
Built:        Tue Sep  5 19:59:19 2017
OS/Arch:      linux/amd64
Experimental: true
```

## What is a client bundle?



A client bundle is a group of certificates downloadable directly from the [Docker Universal Control Plane](https://docs.docker.com/datacenter/ucp/2.2/guides/)

All (/) (Engineering (/category/engineering) - Curated (/curated)) UCP

(<https://docs.docker.com/datacenter/ucp/2.2/guides/>)

Docker Weekly (/docker-weekly-archives/)

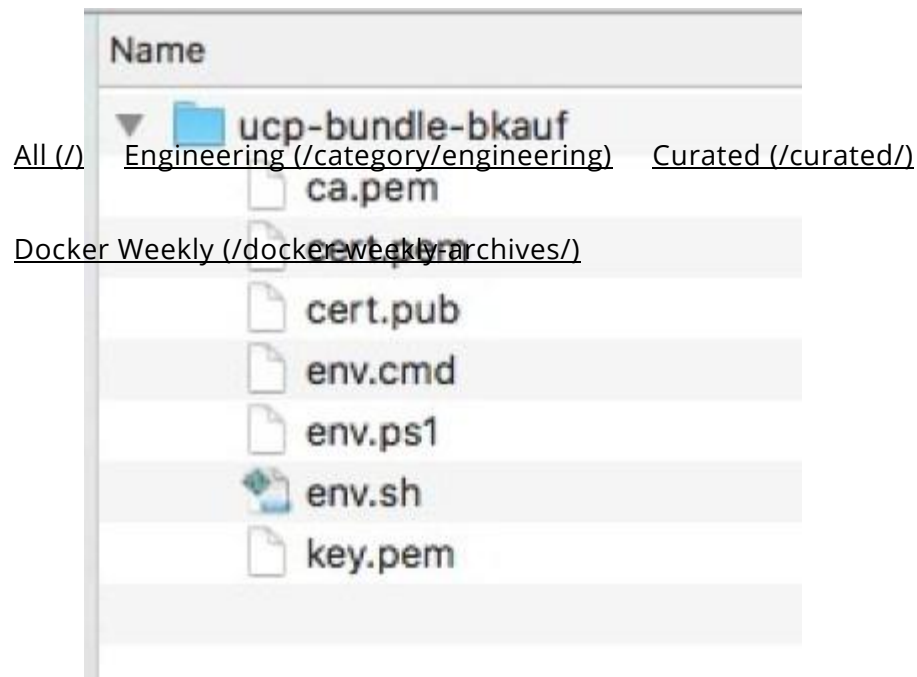
(<https://docs.docker.com/datacenter/ucp/2.2/guides/>) user interface

within the admin section for “My Profile”. This allows you to authorize a remote Docker engine to a specific user account managed in Docker EE, absorbing all associated RBAC controls in the process. You can now execute docker swarm commands from your remote machine that take effect on the remote cluster.

### Example:

I have a user named ‘bkauf’ in my UCP. I download and extract a client bundle for this user.





I open a terminal session with my docker for mac and issue a *docker version* command. You will see the server version matches the client. I can do a *docker ps* and verify nothing is running.

```

Documents $docker version
Client:
  Version: 17.06.2-ce
  API version: 1.30
  Go version: go1.8.3
  Git commit: cec0b72
  Built: Tue Sep  5 20:12:06 2017
  OS/Arch: darwin/amd64

Server:
  Version: 17.06.2-ce
  API version: 1.30 (minimum version 1.12)
  Go version: go1.8.3
  Git commit: cec0b72
  Built: Tue Sep  5 19:59:19 2017
  OS/Arch: linux/amd64
  Experimental: true
Documents $docker ps
CONTAINER ID        IMAGE               COMMAND
Documents $

```

Now, I navigate to the extracted bundle directory and run the env.sh script (env.ps1 for windows)

```

ucp-bundle-bkauf $source env.sh
ucp-bundle-bkauf $docker version
Client:
Version:      17.06.2-ce
API version:  1.30
Go version:   go1.8.3
Git commit:   cec0b72
Built:        Tue Sep  5 20:12:06 2017
OS/Arch:      darwin/amd64

Server:
Version:      ucp/2.2.2
API version:  1.30 (minimum version 1.20)
Go version:   go1.8.3
Git commit:   94d1abdf3
Built:        Wed Aug 30 23:30:05 UTC 2017
OS/Arch:      linux/amd64
Experimental: false
ucp-bundle-bkauf $

```

Notice the server now lists my version as ucp/2.2.2. This is the version of my UCP manager; I'm remotely connected from my laptop to my remote cluster assuming the bkauf user's access levels. I can now do various things such as create a service, view its tasks(containers) and even log into this REMOTE container from my laptop all through the API, no SSH access needed. I need not worry about what host the container is on! This is made possible by the role/permission set up for the use with the granular Role Based Access Control available with Docker EE.

ucp-bundle-bkauf \$docker service create --name node-web-app --p 8080 bkauf/node-web-app

h5rofdm9vc5et09kqb427cawo

Since --detach=false was not specified, tasks will be created in the background.  
In a future release, --detach=false will become the default.

ucp-bundle-bkauf \$docker service ls

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
h5rofdm9vc5e	node-web-app	replicated	1/1	bkauf/node-web-app:latest	*->8080/tcp

ucp-bundle-bkauf \$

ucp-bundle-bkauf \$

ucp-bundle-bkauf \$docker exec -it ecec49857566 sh

/usr/src/app # whoami

root

/usr/src/app # hostname

ecec49857566

/usr/src/app #

[All \(/category/engineering/\)](#) [Curated \(/curated/\)](#)

[Docker Weekly \(/docker-weekly-archives/\)](#)

**My Laptop**

ucp-bundle-bkauf \$docker ps

CONTAINER ID	IMAGE	COMMAND
ecec49857566	bkauf/node-web-app:latest	"npm start"

app.1.lzivde1ln3wzcx6m4p41jqv

ucp-bundle-bkauf \$

ucp-bundle-bkauf \$

ucp-bundle-bkauf \$docker exec -it ecec49857566 sh

/usr/src/app # whoami

root

/usr/src/app # hostname

ecec49857566

/usr/src/app #

**Remote Cluster (UCP Container View)**

1 Containers

State Status ID Node Name

☒ Up 2 minutes (healthy) ecec49857566 node-3 node-web-app.1.lzivde1ln3w

What about a Windows container on a Windows node in a UCP cluster you ask? Linux OR Windows nodes, remote access through your client bundle all works the same!

2. docker

ucp-bundle-docker \$

ucp-bundle-docker \$docker exec -it 1adcdc76fc5c cmd

Microsoft Windows [Version 10.0.14393]  
(c) 2016 Microsoft Corporation. All rights reserved.

C:\>hostname

1adcdc76fc5c

C:\>

State	Status	ID	Node	Name	Image
<input checked="" type="checkbox"/>	Up 33 minutes	1adcdc76fc5c	hq-win-25-b	iis-web-app.1.p2siwcrwhwl	bkauf/iis-web-app:latest

Docker Enterprise Edition (EE) is the only Containers as a Service (CaaS) Platform for IT that manages and secures diverse applications across disparate infrastructure, both on-premises and in the cloud. Docker EE embraces both traditional applications and microservices,

built on Linux and Windows, and intended for x86 servers, mainframes, and public clouds. Docker EE unites all of these applications into a single platform, complete with customizable and flexible access control, support for a broad range of applications and infrastructure, and a highly automated software supply chain.

Learn More

- Visit [IT Starts with Docker \(https://www.docker.com/itpro\)](https://www.docker.com/itpro) and [learn more about MTA \(http://www.docker.com/mta\)](http://www.docker.com/mta)
- Learn more about [Docker Enterprise Edition \(https://www.docker.com/enterprise\)](https://www.docker.com/enterprise)
- Start a [hosted trial \(https://www.docker.com/trial\)](https://www.docker.com/trial)
- Sign up for [upcoming webinars \(https://www.docker.com/webinars\)](https://www.docker.com/webinars)

Get Familiar with #Docker Enterprise Edition Client Bundles  
(<https://twitter.com/share?text=Get+Familiar+with+%23Docker+Enterprise+Edition+Client+Bundles&via=docker&related=docker&url=https://dockr.ly/2xQQLvw>)

CLICK TO TWEET ([HTTPS://TWITTER.COM/SHARE?](https://twitter.com/share?text=Get+Familiar+with+%23Docker+Enterprise+Edition+Client+Bundles&via=docker&related=docker&url=https://dockr.ly/2xQQLvw)

TEXT=GET+FAMILIAR+WITH+%23DOCKER+ENTERPRISE+EDITION+CLIENT+BUNDLES&VIA=DOCKER&RELAT

23DOCKER+ENTERPRISE+EDITION+CLIENT+BUNDLES&VIA=DOCKER&RELAT

[Client Bundles \(https://blog.docker.com/tag/client-bundles/\)](https://blog.docker.com/tag/client-bundles/), [Docker EE \(https://blog.docker.com/tag/docker-ee/\)](https://blog.docker.com/tag/docker-ee/), [Docker Enterprise Edition. \(https://blog.docker.com/tag/docker-enterprise-edition/\)](https://blog.docker.com/tag/docker-enterprise-edition/), [ucp \(https://blog.docker.com/tag/ucp/\)](https://blog.docker.com/tag/ucp/), [universal control plane \(https://blog.docker.com/tag/universal-control-plane/\)](https://blog.docker.com/tag/universal-control-plane/)

[All \(/\)](#) [Engineering \(/category/engineering\)](#) [Curated \(/curated/\)](#)

[Docker Weekly \(/docker-weekly-archives/\)](#)



## GET FAMILIAR WITH DOCKER ENTERPRISE EDITION CLIENT BUNDLES

By [Brian Kaufman \(https://blog.docker.com/author/brian-kaufman/\)](https://blog.docker.com/author/brian-kaufman/)

Solutions Engineer at Docker, Inc

---

## Leave a Reply

Comment

[All \(/\)](#) [Engineering \(/category/engineering\)](/category/engineering/) [Curated \(/curated/\)](/curated/)

[Docker Weekly \(/docker-weekly-archives/\)](/docker-weekly-archives/)

Submit Comment

Notify me of follow-up comments by email.

Notify me of new posts by email.

## Related Posts



[\(https://blog.docker.com/2015/01/docker-project-2014-a-whirlwind-year-in-review/\)](https://blog.docker.com/2015/01/docker-project-2014-a-whirlwind-year-in-review/)

[Docker Project 2014: A Whirlwind Year in Review](https://blog.docker.com/2015/01/docker-project-2014-a-whirlwind-year-in-review/)

[\(https://blog.docker.com/2015/01/docker-project-2014-a-whirlwind-year-in-review/\)](https://blog.docker.com/2015/01/docker-project-2014-a-whirlwind-year-in-review/)

By [David Messina \(https://blog.docker.com/author/david-messina/\)](https://blog.docker.com/author/david-messina/)

| January 15, 2015

[container downloads \(https://blog.docker.com/tag/container-downloads/\)](https://blog.docker.com/tag/container-downloads/), [Dockerfiles \(https://blog.docker.com/tag/dockerfiles/\)](https://blog.docker.com/tag/dockerfiles/), [Dockerized apps \(https://blog.docker.com/tag/dockerized-apps/\)](https://blog.docker.com/tag/dockerized-apps/), [GitHub metrics \(https://blog.docker.com/tag/github-metrics/\)](https://blog.docker.com/tag/github-metrics/), [milestones \(https://blog.docker.com/tag/milestones/\)](https://blog.docker.com/tag/milestones/)



[All \(/\)](#)[Engineering \(/category/engineering/\)](/category/engineering/)[Curated \(/curated/\)](/curated/)[Docker Weekly \(/docker-weekly-archives/\)](/docker-weekly-archives/)

(<https://blog.docker.com/2015/03/videos-from-the-biggest-docker-meetup-ever-clocker-storage-drivers-docker-on-the-desktop-powerstrip/>)

[Videos from the biggest docker meetup ever: Clocker, Storage Drivers, Docker on the Desktop, Powerstrip](https://blog.docker.com/2015/03/videos-from-the-biggest-docker-meetup-ever-clocker-storage-drivers-docker-on-the-desktop-powerstrip/)  
(<https://blog.docker.com/2015/03/videos-from-the-biggest-docker-meetup-ever-clocker-storage-drivers-docker-on-the-desktop-powerstrip/>)

By [Victor Coisne \(https://blog.docker.com/author/victor-coisne/\)](https://blog.docker.com/author/victor-coisne/)

| March 31, 2015

[Clocker \(https://blog.docker.com/tag/clocker/\)](https://blog.docker.com/tag/clocker/), [desktop \(https://blog.docker.com/tag/desktop/\)](https://blog.docker.com/tag/desktop/), [Docker London \(https://blog.docker.com/tag/docker-london/\)](https://blog.docker.com/tag/docker-london/), [Docker Meetup \(https://blog.docker.com/tag/docker-meetup/\)](https://blog.docker.com/tag/docker-meetup/), [NoSQL \(https://blog.docker.com/tag/nosql/\)](https://blog.docker.com/tag/nosql/), [Powerstrip \(https://blog.docker.com/tag/powerstrip/\)](https://blog.docker.com/tag/powerstrip/), [Storage Drivers \(https://blog.docker.com/tag/storage-drivers/\)](https://blog.docker.com/tag/storage-drivers/)

(<https://blog.docker.com/2015/11/aws-docker-eu/>)

## Docker Subscription on AWS Now Available in Europe

(<https://blog.docker.com/2015/11/aws-docker-eu/>)

By Trisha McCanna (<https://blog.docker.com/author/trisha-mccanna/>)

All (/) Engineering (/category/engineering) Curated (/curated/)  
November 17, 2015

aws (<https://blog.docker.com/tag/aws-2/>), docker

Docker Weekly (/docker-weekly-archives/)  
(<https://blog.docker.com/tag/docker/>), docker subscription  
(<https://blog.docker.com/tag/docker-subscription/>)

## Get the Latest Docker News by Email

Docker Weekly is a newsletter with the latest content on Docker and the agenda for the upcoming weeks.

Subscribe to our newsletter

<input type="text" value="Company Email"/>	<input type="text" value="Select Country..."/>
<input type="submit" value="Submit"/>	

Sign up for our newsletter.

<input type="text" value="Company Email"/>	<input type="text" value="Select Country..."/>	<input type="submit" value="Submit"/>
--	--	---------------------------------------

[All \(/\)](#) [Engineering \(/category/engineering\)](#) [Curated \(/curated/\)](#)  
**[What is Docker \(https://www.docker.com/what-docker\)](https://www.docker.com/what-docker)**

[Docker Weekly \(/docker-weekly-archives/\)](#)  
**[Product \(https://www.docker.com/get-docker\)](https://www.docker.com/get-docker)**

**[Documentation \(https://docs.docker.com/\)](https://docs.docker.com/)**

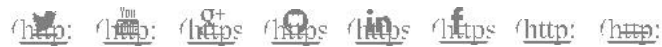
**[Community \(https://www.docker.com/docker-community\)](https://www.docker.com/docker-community)**

[Status \(http://status.docker.com/\)](http://status.docker.com/) [Security \(https://docker.com/docker-security\)](https://docker.com/docker-security)

[Legal \(https://www.docker.com/legal\)](https://www.docker.com/legal) [Contact \(https://www.docker.com/company/contact\)](https://www.docker.com/company/contact)

---

Copyright © 2018 Docker Inc. All rights reserved.



# Secure Engine

*Estimated reading time: 1 minute*

This section discusses the security features you can configure and use within your Docker Engine installation.

- You can configure Docker's trust features so that your users can push and pull trusted images. To learn how to do this, see [Use trusted images \(https://docs.docker.com/engine/security/trust/\)](https://docs.docker.com/engine/security/trust/) in this section.
- You can protect the Docker daemon socket and ensure only trusted Docker client connections. For more information, see [Protect the Docker daemon socket \(https://docs.docker.com/engine/security/https/\)](https://docs.docker.com/engine/security/https/)
- You can use certificate-based client-server authentication to verify a Docker daemon has the rights to access images on a registry. For more information, see [Using certificates for repository client verification \(https://docs.docker.com/engine/security/certificates/\)](https://docs.docker.com/engine/security/certificates/).
- You can configure secure computing mode (Seccomp) policies to secure system calls in a container. For more information, see [Seccomp security profiles for Docker \(https://docs.docker.com/engine/security/seccomp/\)](https://docs.docker.com/engine/security/seccomp/).
- An AppArmor profile for Docker is installed with the official `.deb` packages. For information about this profile and overriding it, see [AppArmor security profiles for Docker \(https://docs.docker.com/engine/security/apparmor/\)](https://docs.docker.com/engine/security/apparmor/).

[seccomp \(https://docs.docker.com/glossary/?term=seccomp\)](https://docs.docker.com/glossary/?term=seccomp), [security \(https://docs.docker.com/glossary/?term=security\)](https://docs.docker.com/glossary/?term=security), [docker \(https://docs.docker.com/glossary/?term=docker\)](https://docs.docker.com/glossary/?term=docker), [documentation \(https://docs.docker.com/glossary/?term=documentation\)](https://docs.docker.com/glossary/?term=documentation)

# Docker security

*Estimated reading time: 11 minutes*

There are four major areas to consider when reviewing Docker security:

- the intrinsic security of the kernel and its support for namespaces and cgroups;
- the attack surface of the Docker daemon itself;
- loopholes in the container configuration profile, either by default, or when customized by users.
- the “hardening” security features of the kernel and how they interact with containers.

## Kernel namespaces

Docker containers are very similar to LXC containers, and they have similar security features. When you start a container with `docker run`, behind the scenes Docker creates a set of namespaces and control groups for the container.

Namespaces provide the first and most straightforward form of isolation: processes running within a container cannot see, and even less affect, processes running in another container, or in the host system.

Each container also gets its own network stack, meaning that a container doesn't get privileged access to the sockets or interfaces of another container. Of course, if the host system is setup accordingly, containers can interact with each other through their respective network interfaces — just like they can interact with external hosts. When you specify public ports for your containers or use *links*

([https://docs.docker.com/engine/userguide/networking/default\\_network/dockerlinks/](https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/)) then IP traffic is allowed between containers. They can ping each other, send/receive UDP packets, and establish TCP connections, but that can be restricted if necessary. From a network architecture point of view, all containers on a given Docker host are sitting on bridge interfaces. This means that they are just like physical machines connected through a common Ethernet switch; no more, no less.

How mature is the code providing kernel namespaces and private networking? Kernel namespaces were introduced between kernel version 2.6.15 and 2.6.26 (<http://man7.org/linux/man-pages/man7/namespaces.7.html>). This means that since July 2008 (date of the 2.6.26 release ), namespace code has been exercised and scrutinized on a large number of production systems. And there is more: the design and inspiration for the namespaces code are even older. Namespaces are actually an effort to reimplement the features of OpenVZ (<http://en.wikipedia.org/wiki/OpenVZ>) in such a way that they could be merged within the mainstream kernel. And OpenVZ was initially released in 2005, so both the design and the implementation are pretty mature.

## Control groups

Control Groups are another key component of Linux Containers. They implement resource accounting and limiting. They provide many useful metrics, but they also help ensure that each container gets its fair share of memory, CPU, disk I/O; and, more importantly, that a single container cannot bring the system down by exhausting one of those resources.

So while they do not play a role in preventing one container from accessing or affecting the data and processes of another container, they are essential to fend off some denial-of-service attacks. They are particularly important on multi-tenant platforms, like public and private PaaS, to guarantee a consistent uptime (and performance) even when some applications start to misbehave.

Control Groups have been around for a while as well: the code was started in 2006, and initially merged in kernel 2.6.24.

## Docker daemon attack surface

Running containers (and applications) with Docker implies running the Docker daemon. This daemon currently requires `root` privileges, and you should therefore be aware of some important details.

First of all, only trusted users should be allowed to control your Docker daemon. This is a direct consequence of some powerful Docker features. Specifically, Docker allows you to share a directory between the Docker host and a guest container; and it allows you to do so without limiting the access rights of the container. This means that you can start a container where the `/host`

directory is the `/` directory on your host; and the container can alter your host filesystem without any restriction. This is similar to how virtualization systems allow filesystem resource sharing. Nothing prevents you from sharing your root filesystem (or even your root block device) with a virtual machine.

This has a strong security implication: for example, if you instrument Docker from a web server to provision containers through an API, you should be even more careful than usual with parameter checking, to make sure that a malicious user cannot pass crafted parameters causing Docker to create arbitrary containers.

For this reason, the REST API endpoint (used by the Docker CLI to communicate with the Docker daemon) changed in Docker 0.5.2, and now uses a UNIX socket instead of a TCP socket bound on 127.0.0.1 (the latter being prone to cross-site request forgery attacks if you happen to run Docker directly on your local machine, outside of a VM). You can then use traditional UNIX permission checks to limit access to the control socket.

You can also expose the REST API over HTTP if you explicitly decide to do so. However, if you do that, be aware of the above mentioned security implications. Ensure that it is reachable only from a trusted network or VPN or protected with a mechanism such as `stunnel` and client SSL certificates. You can also secure API endpoints with HTTPS and certificates (<https://docs.docker.com/engine/security/https/>).

The daemon is also potentially vulnerable to other inputs, such as image loading from either disk with `docker load`, or from the network with `docker pull`. As of Docker 1.3.2, images are now extracted in a chrooted subprocess on Linux/Unix platforms, being the first-step in a wider effort toward privilege separation. As of Docker 1.10.0, all images are stored and accessed by the cryptographic checksums of their contents, limiting the possibility of an attacker causing a collision with an existing image.

Finally, if you run Docker on a server, it is recommended to run exclusively Docker on the server, and move all other services within containers controlled by Docker. Of course, it is fine to keep your favorite admin tools (probably at least an SSH server), as well as existing monitoring/supervision processes, such as NRPE and collectd.

## Linux kernel capabilities

By default, Docker starts containers with a restricted set of capabilities. What does that mean?

Capabilities turn the binary “root/non-root” dichotomy into a fine-grained access control system. Processes (like web servers) that just need to bind on a port below 1024 do not need to run as root: they can just be granted the `net_bind_service` capability instead. And there are many other capabilities, for almost all the specific areas where root privileges are usually needed.

This means a lot for container security; let’s see why!

Typical servers run several processes as `root`, including the SSH daemon, `cron` daemon, logging daemons, kernel modules, network configuration tools, and more. A container is different, because almost all of those tasks are handled by the infrastructure around the container:

- SSH access are typically managed by a single server running on the Docker host;
- `cron`, when necessary, should run as a user process, dedicated and tailored for the app that needs its scheduling service, rather than as a platform-wide facility;
- log management is also typically handed to Docker, or to third-party services like Loggly or Splunk;
- hardware management is irrelevant, meaning that you never need to run `udev` or equivalent daemons within containers;
- network management happens outside of the containers, enforcing separation of concerns as much as possible, meaning that a container should never need to perform `ifconfig`, `route`, or `ip` commands (except when a container is specifically engineered to behave like a router or firewall, of course).

This means that in most cases, containers do not need “real” root privileges *at all*. And therefore, containers can run with a reduced capability set; meaning that “root” within a container has much less privileges than the real “root”. For instance, it is possible to:

- deny all “mount” operations;
- deny access to raw sockets (to prevent packet spoofing);
- deny access to some filesystem operations, like creating new device nodes, changing the owner of files, or altering attributes (including the immutable flag);
- deny module loading;
- and many others.

This means that even if an intruder manages to escalate to root within a



container, it is much harder to do serious damage, or to escalate to the host.

This doesn't affect regular web apps, but reduces the vectors of attack by malicious users considerably. By default Docker drops all capabilities except those needed (<https://github.com/moby/moby/blob/master/oci/defaults.go#L14-L30>), a whitelist instead of a blacklist approach. You can see a full list of available capabilities in Linux manpages (<http://man7.org/linux/man-pages/man7/capabilities.7.html>).

One primary risk with running Docker containers is that the default set of capabilities and mounts given to a container may provide incomplete isolation, either independently, or when used in combination with kernel vulnerabilities.

Docker supports the addition and removal of capabilities, allowing use of a non-default profile. This may make Docker more secure through capability removal, or less secure through the addition of capabilities. The best practice for users would be to remove all capabilities except those explicitly required for their processes.

## Docker Content Trust Signature Verification

The Docker Engine can be configured to only run signed images. The Docker Content Trust signature verification feature is built directly into the `dockerd` binary.

This is configured in the Dockerd configuration file.

To enable this feature, trustpinning can be configured in `daemon.json`, whereby only repositories signed with a user-specified root key can be pulled and run.

This feature provides more insight to administrators than previously available with the CLI for enforcing and performing image signature verification.

For more information on configuring Docker Content Trust Signature Verification, go to (Content trust in Docker)[engine/security/trust/content\_trust].

## Other kernel security features

Capabilities are just one of the many security features provided by modern Linux kernels. It is also possible to leverage existing, well-known systems like TOMOYO, AppArmor, SELinux, GRSEC, etc. with Docker.

While Docker currently only enables capabilities, it doesn't interfere with the other systems. This means that there are many different ways to harden a Docker host. Here are a few examples.

- You can run a kernel with GRSEC and PAX. This adds many safety checks, both at compile-time and run-time; it also defeats many exploits, thanks to techniques like address randomization. It doesn't require Docker-specific configuration, since those security features apply system-wide, independent of containers.
- If your distribution comes with security model templates for Docker containers, you can use them out of the box. For instance, we ship a template that works with AppArmor and Red Hat comes with SELinux policies for Docker. These templates provide an extra safety net (even though it overlaps greatly with capabilities).
- You can define your own policies using your favorite access control mechanism.

Just as you can use third-party tools to augment Docker containers, including special network topologies or shared filesystems, tools exist to harden Docker containers without the need to modify Docker itself.

As of Docker 1.10 User Namespaces are supported directly by the docker daemon. This feature allows for the root user in a container to be mapped to a non uid-0 user outside the container, which can help to mitigate the risks of container breakout. This facility is available but not enabled by default.

Refer to the daemon command

(<https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-user-namespace-options>) in the command line reference for more information on this feature. Additional information on the implementation of User Namespaces in Docker can be found in this blog post (<https://integratedcode.us/2015/10/13/user-namespaces-have-arrived-in-docker/>).

## Conclusions

Docker containers are, by default, quite secure; especially if you run your processes as non-privileged users inside the container.

You can add an extra layer of safety by enabling AppArmor, SELinux, GRSEC, or another appropriate hardening system.

If you think of ways to make docker more secure, we welcome feature requests,

pull requests, or comments on the Docker community forums.

## Related information

- Use trusted images (<https://docs.docker.com/engine/security/trust/>)
- Seccomp security profiles for Docker (<https://docs.docker.com/engine/security/seccomp/>)
- AppArmor security profiles for Docker (<https://docs.docker.com/engine/security/apparmor/>)
- On the Security of Containers (2014) (<https://medium.com/@ewindisch/on-the-security-of-containers-2c60ffe25a9e>)
- Docker swarm mode overlay network security model (<https://docs.docker.com/engine/userguide/networking/overlay-security-model/>)

Docker (<https://docs.docker.com/glossary/?term=Docker>), Docker documentation (<https://docs.docker.com/glossary/?term=Docker documentation>), security (<https://docs.docker.com/glossary/?term=security>)

# Swarm mode overview

*Estimated reading time: 3 minutes*

To use Docker in swarm mode, install Docker. See installation instructions (<https://docs.docker.com/install/>) for all operating systems and platforms.

Current versions of Docker include *swarm mode* for natively managing a cluster of Docker Engines called a *swarm*. Use the Docker CLI to create a swarm, deploy application services to a swarm, and manage swarm behavior.

If you are using a Docker version prior to [1.12.0](#), you can use standalone swarm (<https://docs.docker.com/swarm/>), but we recommend updating.

## Feature highlights

- Cluster management integrated with Docker Engine: Use the Docker Engine CLI to create a swarm of Docker Engines where you can deploy application services. You don't need additional orchestration software to create or manage a swarm.
- Decentralized design: Instead of handling differentiation between node roles at deployment time, the Docker Engine handles any specialization at runtime. You can deploy both kinds of nodes, managers and workers, using the Docker Engine. This means you can build an entire swarm from a single disk image.
- Declarative service model: Docker Engine uses a declarative approach to let you define the desired state of the various services in your application stack. For example, you might describe an application comprised of a web front end service with message queueing services and a database backend.
- Scaling: For each service, you can declare the number of tasks you want to run. When you scale up or down, the swarm manager automatically adapts by adding or removing tasks to maintain the desired state.

- **Desired state reconciliation:** The swarm manager node constantly monitors the cluster state and reconciles any differences between the actual state and your expressed desired state. For example, if you set up a service to run 10 replicas of a container, and a worker machine hosting two of those replicas crashes, the manager creates two new replicas to replace the replicas that crashed. The swarm manager assigns the new replicas to workers that are running and available.
- **Multi-host networking:** You can specify an overlay network for your services. The swarm manager automatically assigns addresses to the containers on the overlay network when it initializes or updates the application.
- **Service discovery:** Swarm manager nodes assign each service in the swarm a unique DNS name and load balances running containers. You can query every container running in the swarm through a DNS server embedded in the swarm.
- **Load balancing:** You can expose the ports for services to an external load balancer. Internally, the swarm lets you specify how to distribute service containers between nodes.
- **Secure by default:** Each node in the swarm enforces TLS mutual authentication and encryption to secure communications between itself and all other nodes. You have the option to use self-signed root certificates or certificates from a custom root CA.
- **Rolling updates:** At rollout time you can apply service updates to nodes incrementally. The swarm manager lets you control the delay between service deployment to different sets of nodes. If anything goes wrong, you can roll-back a task to a previous version of the service.

## What's next?

### Swarm mode key concepts and tutorial

- Learn swarm mode key concepts  
(<https://docs.docker.com/engine/swarm/key-concepts/>).
- Get started with the Swarm mode tutorial  
(<https://docs.docker.com/engine/swarm/swarm-tutorial/>).

## Swarm mode CLI commands

Explore swarm mode CLI commands

- `swarm init`  
([https://docs.docker.com/engine/reference/commandline/swarm\\_init/](https://docs.docker.com/engine/reference/commandline/swarm_init/))
- `swarm join`  
([https://docs.docker.com/engine/reference/commandline/swarm\\_join/](https://docs.docker.com/engine/reference/commandline/swarm_join/))
- `service create`  
([https://docs.docker.com/engine/reference/commandline/service\\_create/](https://docs.docker.com/engine/reference/commandline/service_create/))
- `service inspect`  
([https://docs.docker.com/engine/reference/commandline/service\\_inspect/](https://docs.docker.com/engine/reference/commandline/service_inspect/))
- `service ls`  
([https://docs.docker.com/engine/reference/commandline/service\\_ls/](https://docs.docker.com/engine/reference/commandline/service_ls/))
- `service rm`  
([https://docs.docker.com/engine/reference/commandline/service\\_rm/](https://docs.docker.com/engine/reference/commandline/service_rm/))
- `service scale`  
([https://docs.docker.com/engine/reference/commandline/service\\_scale/](https://docs.docker.com/engine/reference/commandline/service_scale/))
- `service ps`  
([https://docs.docker.com/engine/reference/commandline/service\\_ps/](https://docs.docker.com/engine/reference/commandline/service_ps/))
- `service update`  
([https://docs.docker.com/engine/reference/commandline/service\\_update/](https://docs.docker.com/engine/reference/commandline/service_update/))

`docker` (<https://docs.docker.com/glossary/?term=docker>), `container`

(<https://docs.docker.com/glossary/?term=container>), `cluster`

(<https://docs.docker.com/glossary/?term=cluster>), `swarm`

(<https://docs.docker.com/glossary/?term=swarm>)

# Manage swarm security with public key infrastructure (PKI)

*Estimated reading time: 4 minutes*

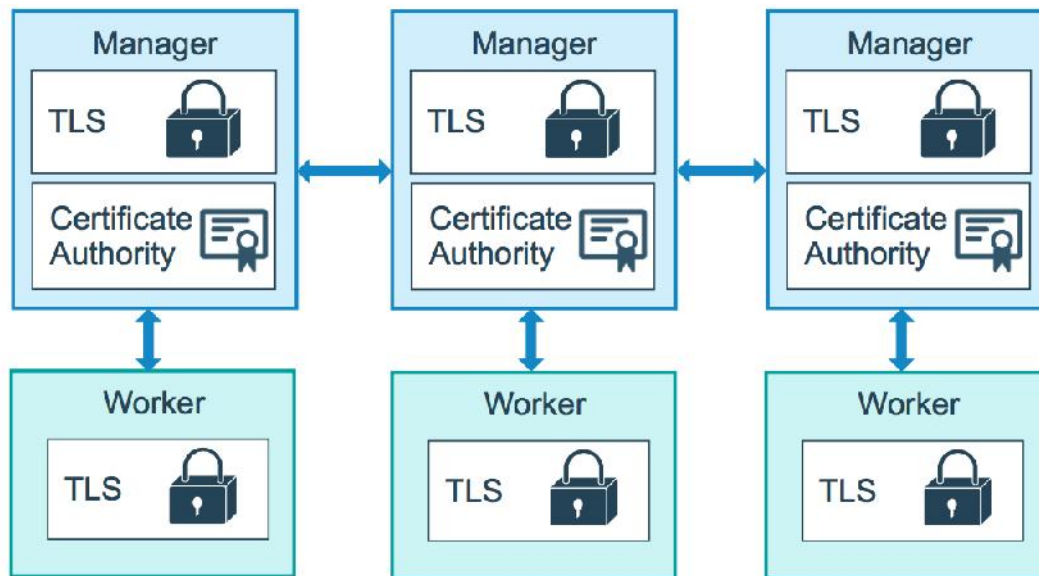
The swarm mode public key infrastructure (PKI) system built into Docker makes it simple to securely deploy a container orchestration system. The nodes in a swarm use mutual Transport Layer Security (TLS) to authenticate, authorize, and encrypt the communications with other nodes in the swarm.

When you create a swarm by running `docker swarm init`, Docker designates itself as a manager node. By default, the manager node generates a new root Certificate Authority (CA) along with a key pair, which are used to secure communications with other nodes that join the swarm. If you prefer, you can specify your own externally-generated root CA, using the `--external-ca` flag of the `docker swarm init` ([https://docs.docker.com/engine/reference/commandline/swarm\\_init/](https://docs.docker.com/engine/reference/commandline/swarm_init/)) command.

The manager node also generates two tokens to use when you join additional nodes to the swarm: one worker token and one manager token. Each token includes the digest of the root CA's certificate and a randomly generated secret. When a node joins the swarm, the joining node uses the digest to validate the root CA certificate from the remote manager. The remote manager uses the secret to ensure the joining node is an approved node.

Each time a new node joins the swarm, the manager issues a certificate to the node. The certificate contains a randomly generated node ID to identify the node under the certificate common name (CN) and the role under the organizational unit (OU). The node ID serves as the cryptographically secure node identity for the lifetime of the node in the current swarm.

The diagram below illustrates how manager nodes and worker nodes encrypt communications using a minimum of TLS 1.2.



The example below shows the information from a certificate from a worker node:

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      3b:1c:06:91:73:fb:16:ff:69:c3:f7:a2:fe:96:c1:73:e2:80:97
:3b
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: CN=swarm-ca
    Validity
      Not Before: Aug 30 02:39:00 2016 GMT
      Not After : Nov 28 03:39:00 2016 GMT
    Subject: O=ec2adilxf4ngv7ev8fws161i7, OU=swarm-worker, CN=dw
02poa4vqvzxi5c10gm4pq2g
...snip...
```

By default, each node in the swarm renews its certificate every three months. You can configure this interval by running the

`docker swarm update --cert-expiry <TIME PERIOD>` command. The minimum rotation value is 1 hour. Refer to the `docker swarm update` ([https://docs.docker.com/engine/reference/commandline/swarm\\_update/](https://docs.docker.com/engine/reference/commandline/swarm_update/)) CLI reference for details.

## Rotating the CA certificate



In the event that a cluster CA key or a manager node is compromised, you can rotate the swarm root CA so that none of the nodes trust certificates signed by the old root CA anymore.

Run `docker swarm ca --rotate` to generate a new CA certificate and key. If you prefer, you can pass the `--ca-cert` and `--external-ca` flags to specify the root certificate and to use a root CA external to the swarm. Alternately, you can pass the `--ca-cert` and `--ca-key` flags to specify the exact certificate and key you would like the swarm to use.

When you issue the `docker swarm ca --rotate` command, the following things happen in sequence:

1. Docker generates a cross-signed certificate. This means that a version of the new root CA certificate is signed with the old root CA certificate. This cross-signed certificate is used as an intermediate certificate for all new node certificates. This ensures that nodes that still trust the old root CA can still validate a certificate signed by the new CA.
2. In Docker 17.06 and higher, Docker also tells all nodes to immediately renew their TLS certificates. This process may take several minutes, depending on the number of nodes in the swarm.

✔ Note: If your swarm has nodes with different Docker versions, the following two things are true:

Only a manager that is running as the leader and running Docker 17.06 or higher tells nodes to renew their TLS certificates.

Only nodes running Docker 17.06 or higher obey this directive.

For the most predictable behavior, ensure that all swarm nodes are running Docker 17.06 or higher.

3. After every node in the swarm has a new TLS certificate signed by the new CA, Docker forgets about the old CA certificate and key material, and tells all the nodes to trust the new CA certificate only.

This also causes a change in the swarm's join tokens. The previous join tokens are no longer valid.

From this point on, all new node certificates issued are signed with the new root

CA, and do not contain any intermediates.

## Learn More

- Read about how nodes (<https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/>) work.
- Learn how swarm mode services (<https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/>) work.

swarm (<https://docs.docker.com/glossary/?term=swarm>), security (<https://docs.docker.com/glossary/?term=security>), tls (<https://docs.docker.com/glossary/?term=tls>), pki (<https://docs.docker.com/glossary/?term=pki>)

# Roles and permission levels

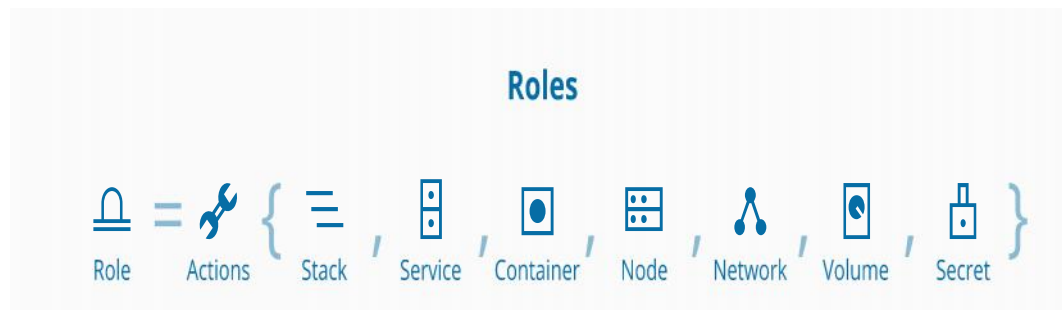
*Estimated reading time: 3 minutes*

✔ These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

2.2.14 ▼

Docker Universal Control Plane has two types of users: administrators and regular users. Administrators can make changes to the UCP swarm, while regular users have permissions that range from no access to full control over resources like volumes, networks, images, and containers. Users are grouped into teams and organizations.



Administrators create *grants* to users, teams, and organizations to give permissions to swarm resources.

## Administrator users

In Docker UCP, only users with administrator privileges can make changes to swarm settings. This includes:

- Managing user permissions by creating grants.
- Managing swarm configurations, like adding and removing nodes.

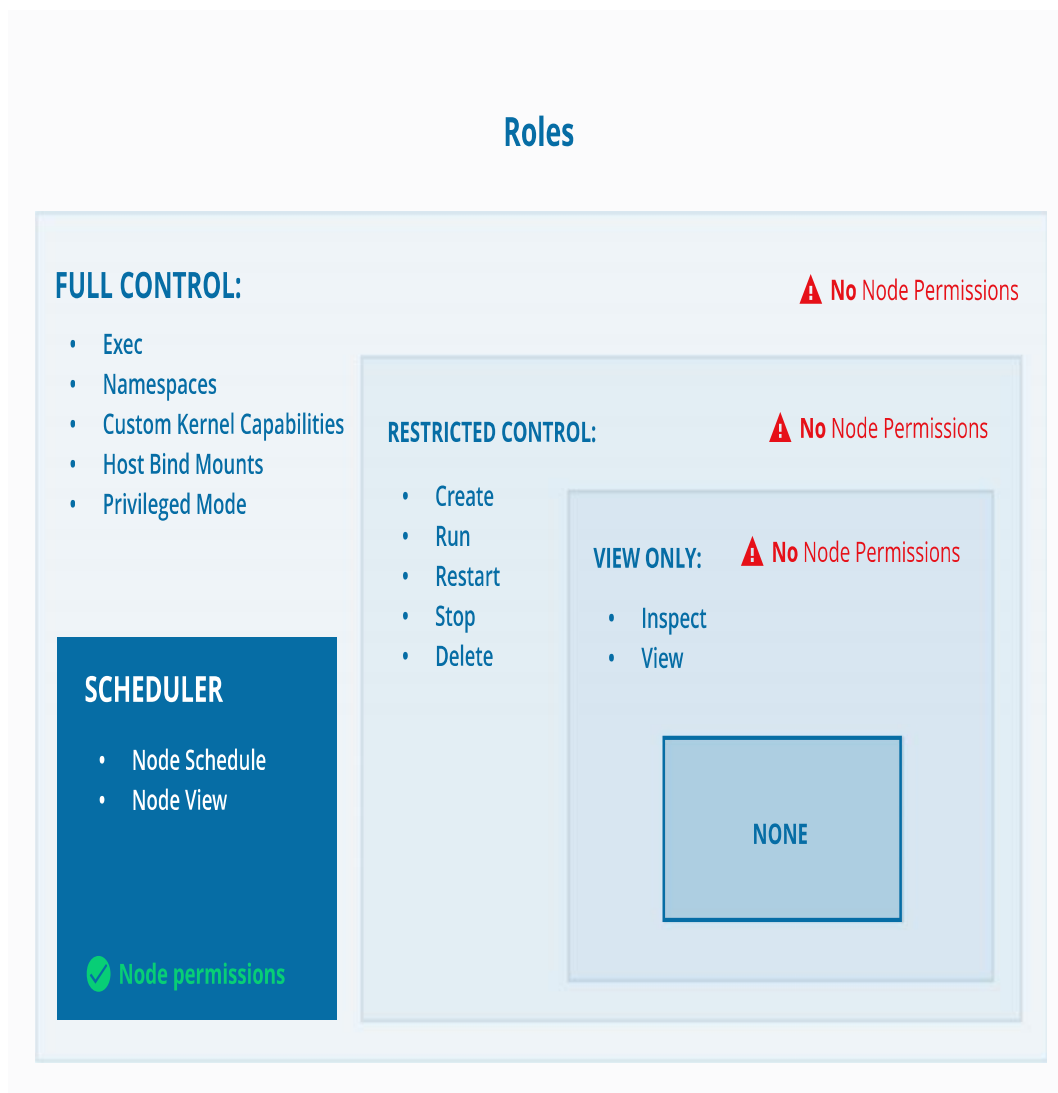
## Roles

A role is a set of permitted API operations on a collection that you can assign to a specific user, team, or organization by using a grant.

UCP administrators view and manage roles by navigating to the Roles page.

The system provides the following default roles:

Built-in role	Description
<a href="#">None</a>	The user has no access to swarm resources. This maps to the <a href="#">No Access</a> role in UCP 2.1.x.
<a href="#">View Only</a>	The user can view resources like services, volumes, and networks but can't create them.
<a href="#">Restricted Control</a>	The user can view and edit volumes, networks, and images but can't run a service or container in a way that might affect the node where it's running. The user can't mount a node directory and can't <a href="#">exec</a> into containers. Also, The user can't run containers in privileged mode or with additional kernel capabilities.
<a href="#">Scheduler</a>	The user can view nodes and schedule workloads on them. Worker nodes and manager nodes are affected by <a href="#">Scheduler</a> grants. Having <a href="#">Scheduler</a> access doesn't allow the user to view workloads on these nodes. They need the appropriate resource permissions, like <a href="#">Container View</a> . By default, all users get a grant with the <a href="#">Scheduler</a> role against the <a href="#">/Shared</a> collection.
<a href="#">Full Control</a>	The user can view and edit volumes, networks, and images. They can create containers without any restriction, but can't see other users' containers.

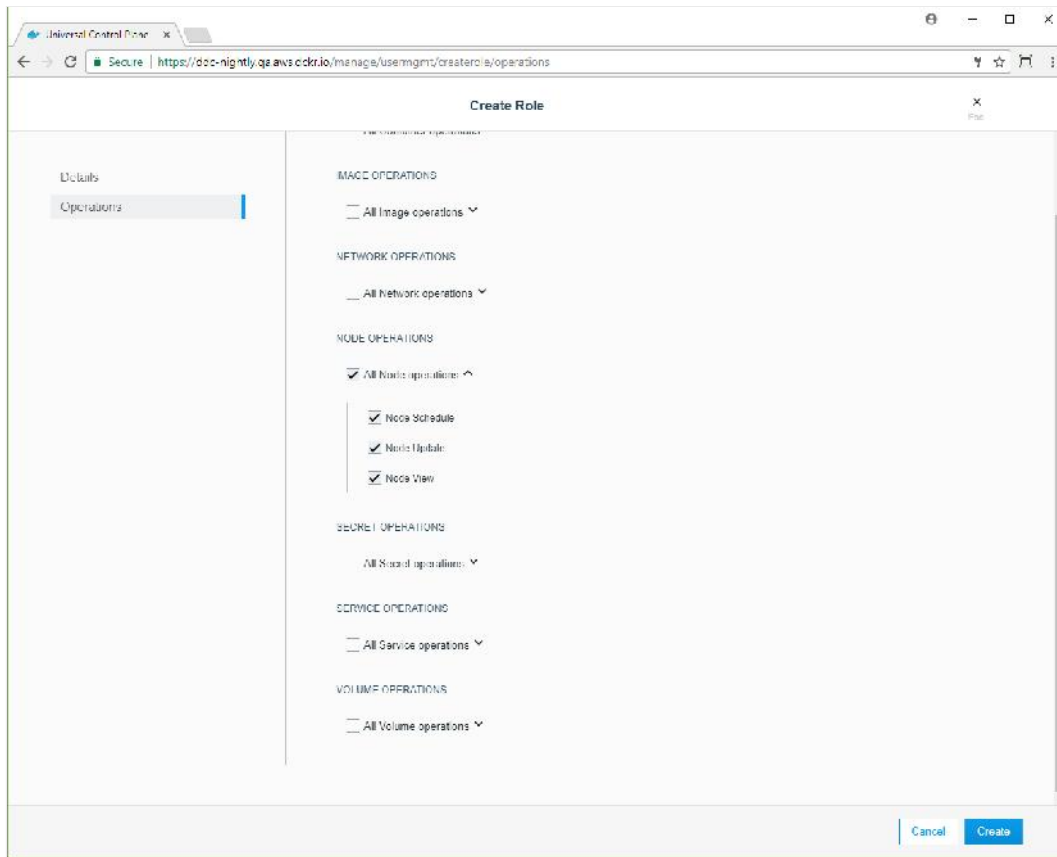


Administrators can create a custom role that has Docker API permissions that specify the API actions that a subject may perform.

The Roles page lists the available roles, including the default roles and any custom roles that administrators have created. In the Roles list, click a role to see the API operations that it uses. For example, the `Scheduler` role has two of the node operations, `Schedule` and `View`.

## Create a custom role

Click Create role to create a custom role and define the API operations that it uses. When you create a custom role, all of the APIs that you can use are listed on the Create Role page. For example, you can create a custom role that uses the node operations, `Schedule`, `Update`, and `View`, and you might give it a name like "Node Operator".



You can give a role a global name, like “Remove Images”, which might enable the Remove and Force Remove operations for images. You can apply a role with the same name to different collections.

Only an administrator can create and remove roles. Roles are always enabled. Roles can’t be edited, so to change a role’s API operations, you must delete it and create it again.

You can’t delete a custom role if it’s used in a grant. You must first delete the grants that use the role.

## Where to go next

- Create and manage users  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/create-and-manage-users/>)
- Create and manage teams  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/create-and-manage-teams/>)
- Docker Reference Architecture: Securing Docker EE and Security Best Practices

([https://success.docker.com/Architecture/Docker\\_Reference\\_Architecture%3A\\_Securing\\_Docker\\_EE\\_and\\_Security\\_Best\\_Practices](https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Securing_Docker_EE_and_Security_Best_Practices))

authorization (<https://docs.docker.com/glossary/?term=authorization>),  
authentication (<https://docs.docker.com/glossary/?term=authentication>), users  
(<https://docs.docker.com/glossary/?term=users>), teams  
(<https://docs.docker.com/glossary/?term=teams>), UCP  
(<https://docs.docker.com/glossary/?term=UCP>)

# UCP architecture

*Estimated reading time: 7 minutes*

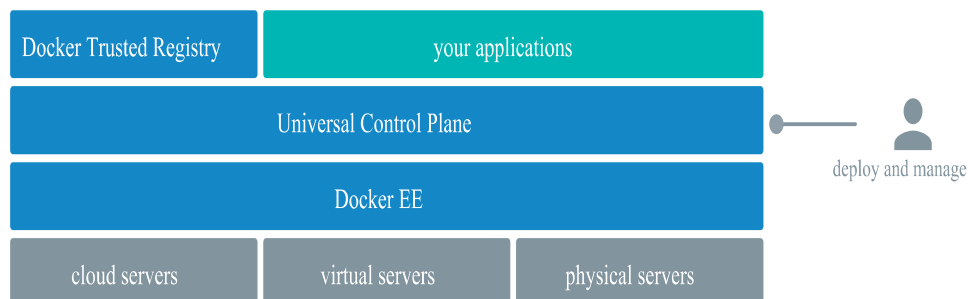
✔ These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

2.2.14 ▼

Universal Control Plane is a containerized application that runs on Docker Enterprise Edition (<https://docs.docker.com/ee/>) and extends its functionality to make it easier to deploy, configure, and monitor your applications at scale.

UCP also secures Docker with role-based access control so that only authorized users can make changes and deploy applications to your Docker cluster.

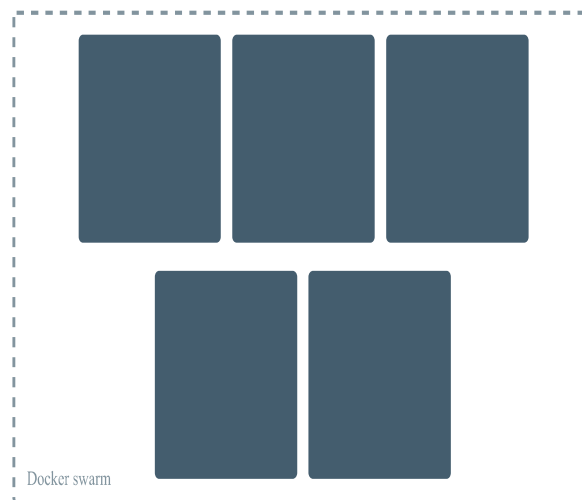


Once Universal Control Plane (UCP) instance is deployed, developers and IT operations no longer interact with Docker Engine directly, but interact with UCP instead. Since UCP exposes the standard Docker API, this is all done transparently, so that you can use the tools you already know and love, like the Docker CLI client and Docker Compose.



## Under the hood

Docker UCP leverages the clustering and orchestration functionality provided by Docker.



A swarm is a collection of nodes that are in the same Docker cluster. Nodes (<https://docs.docker.com/engine/swarm/key-concepts/>) in a Docker swarm operate in one of two modes: Manager or Worker. If nodes are not already running in a swarm when installing UCP, nodes will be configured to run in swarm mode.

When you deploy UCP, it starts running a globally scheduled service called `ucp-agent`. This service monitors the node where it's running and starts and stops UCP services, based on whether the node is a manager or a worker node (<https://docs.docker.com/engine/swarm/key-concepts/>).

If the node is a:

- **Manager:** the `ucp-agent` service automatically starts serving all UCP components, including the UCP web UI and data stores used by UCP. The `ucp-agent` accomplishes this by deploying several containers (</datacenter/ucp/2.2/guides/architecture/#ucp-components-in-manager-nodes>) on the node. By promoting a node to manager, UCP automatically becomes highly available and fault tolerant.
- **Worker:** on worker nodes, the `ucp-agent` service starts serving a proxy service that ensures only authorized users and other UCP services can run Docker commands in that node. The `ucp-agent` deploys a subset of containers (</datacenter/ucp/2.2/guides/architecture/#ucp-components-in-worker-nodes>) on worker nodes.

# UCP internal components

The core component of UCP is a globally-scheduled service called `ucp-agent`. When you install UCP on a node, or join a node to a swarm that's being managed by UCP, the `ucp-agent` service starts running on that node.

Once this service is running, it deploys containers with other UCP components, and it ensures they keep running. The UCP components that are deployed on a node depend on whether the node is a manager or a worker.

## ✔ OS-specific component names

Some UCP component names depend on the node's operating system. For example, on Windows, the `ucp-agent` component is named `ucp-agent-win`. Learn about architecture-specific images (<https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/architecture-specific-images/>).

## UCP components in manager nodes

Manager nodes run all UCP services, including the web UI and data stores that persist the state of UCP. These are the UCP services running on manager nodes:

UCP component	Description
<code>ucp-agent</code>	Monitors the node and ensures the right UCP services are running
<code>ucp-reconcile</code>	When <code>ucp-agent</code> detects that the node is not running the right UCP components, it starts the <code>ucp-reconcile</code> container to converge the node to its desired state. It is expected for the <code>ucp-reconcile</code> container to remain in an exited state when the node is healthy.
<code>ucp-auth-api</code>	The centralized service for identity and authentication used by UCP and DTR
<code>ucp-auth-store</code>	Stores authentication configurations and data for users, organizations, and teams

UCP component	Description
ucp-auth-worker	Performs scheduled LDAP synchronizations and cleans authentication and authorization data
ucp-client-root-ca	A certificate authority to sign client bundles
ucp-cluster-root-ca	A certificate authority used for TLS communication between UCP components
ucp-controller	The UCP web server
ucp-dsinfo	Docker system information collection script to assist with troubleshooting
ucp-kv	Used to store the UCP configurations. Don't use it in your applications, since it's for internal use only
ucp-metrics	Used to collect and process metrics for a node, like the disk space available
ucp-proxy	A TLS proxy. It allows secure access to the local Docker Engine to UCP components
ucp-swarm-manager	Used to provide backwards-compatibility with Docker Swarm

## UCP components in worker nodes

Worker nodes are the ones where you run your applications. These are the UCP services running on worker nodes:

UCP component	Description
ucp-agent	Monitors the node and ensures the right UCP services are running
ucp-dsinfo	Docker system information collection script to assist with troubleshooting

UCP component	Description
ucp-reconcile	When ucp-agent detects that the node is not running the right UCP components, it starts the ucp-reconcile container to converge the node to its desired state. It is expected for the ucp-reconcile container to remain in an exited state when the node is healthy.
ucp-proxy	A TLS proxy. It allows secure access to the local Docker Engine to UCP components

## Volumes used by UCP

Docker UCP uses these named volumes to persist data in all nodes where it runs:

Volume name	Description
ucp-auth-api-certs	Certificate and keys for the authentication and authorization service
ucp-auth-store-certs	Certificate and keys for the authentication and authorization store
ucp-auth-store-data	Data of the authentication and authorization store, replicated across managers
ucp-auth-worker-certs	Certificate and keys for authentication worker
ucp-auth-worker-data	Data of the authentication worker
ucp-client-root-ca	Root key material for the UCP root CA that issues client certificates
ucp-cluster-root-ca	Root key material for the UCP root CA that issues certificates for swarm members
ucp-controller-client-certs	Certificate and keys used by the UCP web server to communicate with other UCP components

Volume name	Description
ucp-controller-server-certs	Certificate and keys for the UCP web server running in the node
ucp-kv	UCP configuration data, replicated across managers
ucp-kv-certs	Certificates and keys for the key-value store
ucp-metrics-data	Monitoring data gathered by UCP
ucp-metrics-inventory	Configuration file used by the ucp-metrics service
ucp-node-certs	Certificate and keys for node communication

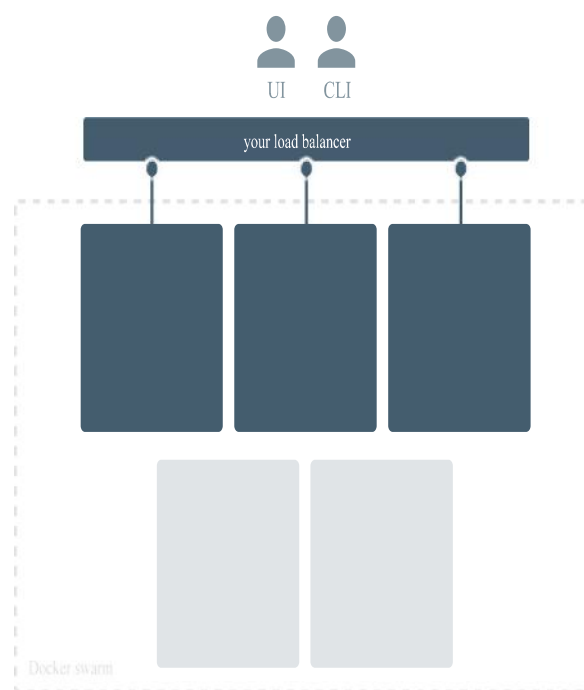
You can customize the volume driver used for these volumes, by creating the volumes before installing UCP. During the installation, UCP checks which volumes don't exist in the node, and creates them using the default volume driver.

By default, the data for these volumes can be found at `/var/lib/docker/volumes/<volume-name>/_data` .

## How you interact with UCP

There are two ways to interact with UCP: the web UI or the CLI.

You can use the UCP web UI to manage your swarm, grant and revoke user permissions, deploy, configure, manage, and monitor your applications.



UCP also exposes the standard Docker API, so you can continue using existing tools like the Docker CLI client. Since UCP secures your cluster with role-based access control, you need to configure your Docker CLI client and other client tools to authenticate your requests using client certificates (<https://docs.docker.com/datacenter/ucp/2.2/guides/user/access-ucp/>) that you can download from your UCP profile page.

## Where to go next

- System requirements (<https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/system-requirements/>)
- Plan your installation (<https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/system-requirements/>)

ucp (<https://docs.docker.com/glossary/?term=ucp>), architecture (<https://docs.docker.com/glossary/?term=architecture>)

# Use your own TLS certificates

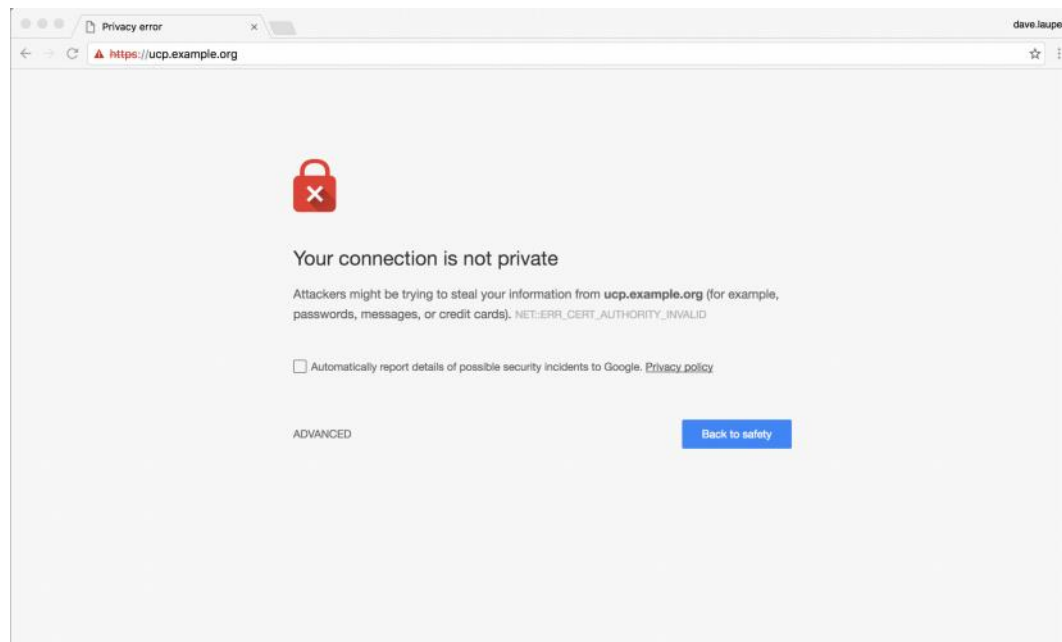
*Estimated reading time: 1 minute*

✔ These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

2.2.14 ▼

All UCP services are exposed using HTTPS, to ensure all communications between clients and UCP are encrypted. By default, this is done using self-signed TLS certificates that are not trusted by client tools like web browsers. So when you try to access UCP, your browser warns that it doesn't trust UCP or that UCP has an invalid certificate.



The same happens with other client tools.

```
$ curl https://ucp.example.org
```

```
SSL certificate problem: Invalid certificate chain
```

You can configure UCP to use your own TLS certificates, so that it is automatically trusted by your browser and client tools.

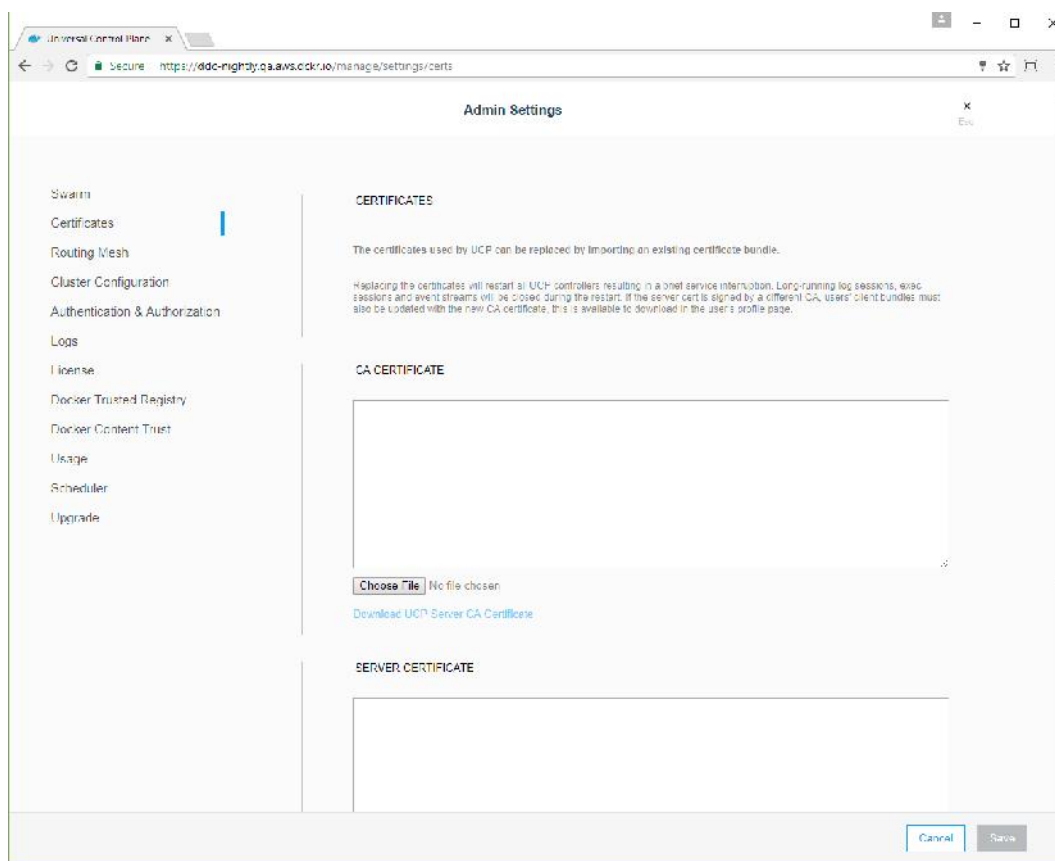
To ensure minimal impact to your business, you should plan for this change to happen outside business peak hours. Your applications will continue running normally, but existing UCP client certificates will become invalid, so users need to download new ones to access UCP from the CLI (<https://docs.docker.com/datacenter/ucp/2.2/guides/user/access-ucp/cli-based-access/>).

## Configure UCP to use your own TLS certificates and keys

In the UCP web UI, log in with administrator credentials and navigate to the Admin Settings page.

In the left pane, click Certificates.





Upload your certificates and keys:

- A `ca.pem` file with the root CA public certificate.
- A `cert.pem` file with the TLS certificate for your domain and any intermediate public certificates, in this order.
- A `key.pem` file with TLS private key. Make sure it is not encrypted with a password. Encrypted keys should have `ENCRYPTED` in the first line.

Finally, click Save for the changes to take effect.

After replacing the TLS certificates, your users can't authenticate with their old client certificate bundles. Ask your users to go to the UCP web UI and get new client certificate bundles

(<https://docs.docker.com/datacenter/ucp/2.2/guides/user/access-ucp/cli-based-access/>).

If you deployed Docker Trusted Registry, you also need to reconfigure it to trust the new UCP TLS certificates. Learn how to configure DTR

(<https://docs.docker.com/datacenter/dtr/2.3/reference/cli/reconfigure/>).

## Where to go next

- Access UCP from the CLI  
(<https://docs.docker.com/datacenter/ucp/2.2/guides/user/access-ucp/cli-based-access/>)

Universal Control Plane ([https://docs.docker.com/glossary/?term=Universal Control Plane](https://docs.docker.com/glossary/?term=Universal%20Control%20Plane)), UCP (<https://docs.docker.com/glossary/?term=UCP>), certificate (<https://docs.docker.com/glossary/?term=certificate>), authentication (<https://docs.docker.com/glossary/?term=authentication>), tls (<https://docs.docker.com/glossary/?term=tls>)

# Use your own TLS certificates

*Estimated reading time: 1 minute*

✔ These are the docs for DTR version 2.3.9

To select a different version, use the selector below.

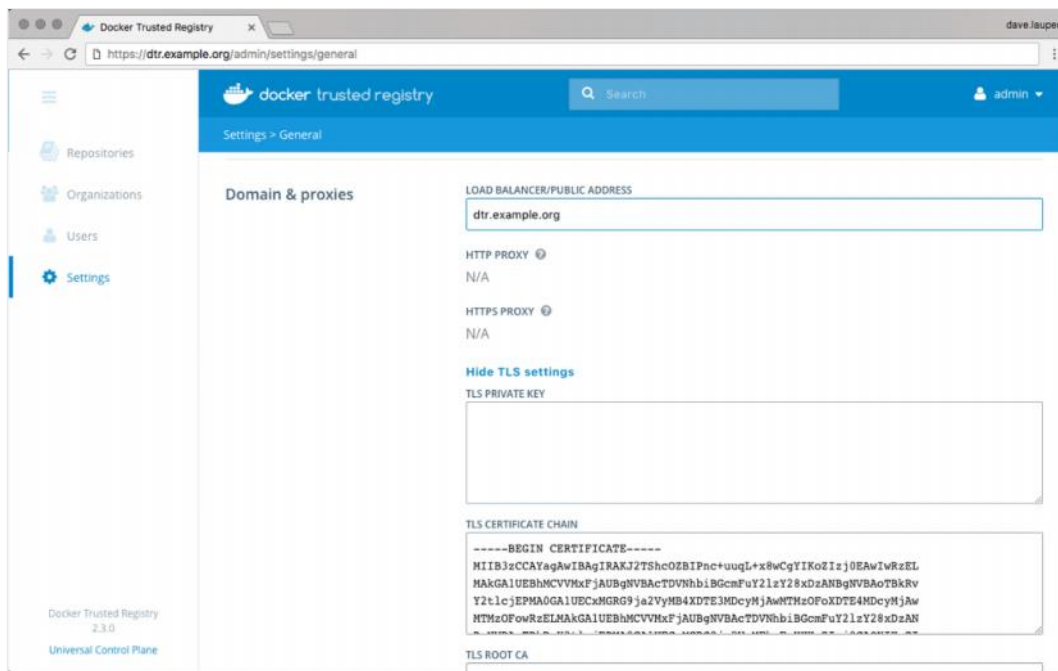
2.3.9 ▼

By default the DTR services are exposed using HTTPS, to ensure all communications between clients and DTR is encrypted. Since DTR replicas use self-signed certificates for this, when a client accesses DTR, their browsers don't trust this certificate, so the browser displays a warning message.

You can configure DTR to use your own certificates, so that it is automatically trusted by your users' browser and client tools.

## Replace the server certificates

To configure DTR to use your own certificates and keys, go to the DTR web UI, navigate to the Settings page, and scroll down to the Domain section.



Set the DTR domain name and upload the certificates and key:

- Load balancer/public address, is the domain name clients will use to access DTR.
- TLS certificate, is the server certificate and any intermediate CA public certificates. This certificate needs to be valid for the DTR public address, and have SANs for all addresses used to reach the DTR replicas, including load balancers.
- TLS private key is the server private key.
- TLS CA is the root CA public certificate.

Finally, click Save for the changes to take effect.

If you're using certificates issued by a globally trusted certificate authority, any web browser or client tool should now trust DTR. If you're using an internal certificate authority, you need to configure your system to trust that certificate authority.

## Where to go next

- Set up external storage  
(<https://docs.docker.com/datacenter/dtr/2.3/guides/admin/configure/external-storage/>)

dtr (<https://docs.docker.com/glossary/?term=dtr>), tls

(<https://docs.docker.com/glossary/?term=tls>)