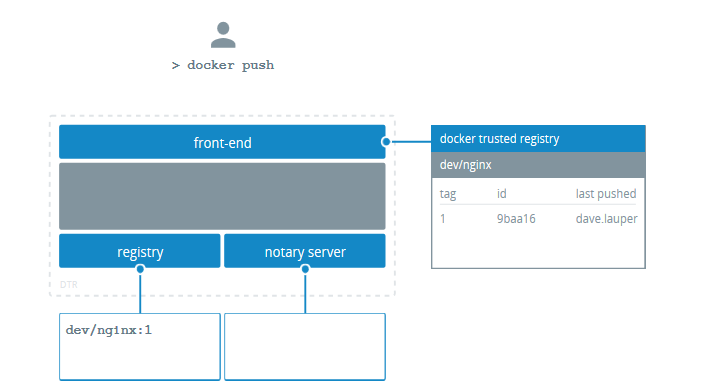
Sign an image

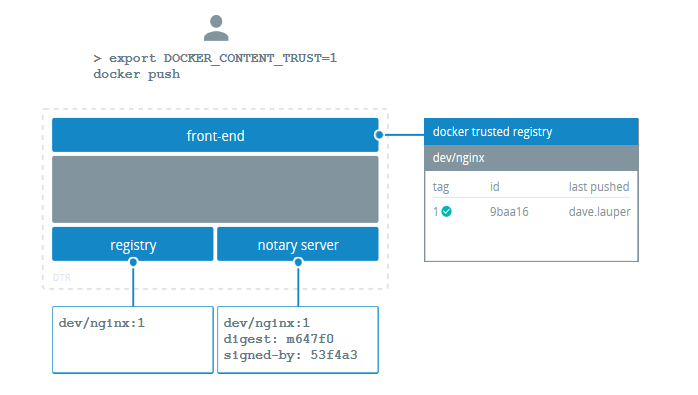
You can configure the Docker CLI client to sign the images you push to DTR. This allows whoever pulls your image to validate if they are getting the image you created, or a forged one.

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 Store, 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 an 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 Store, re-tag it, sign and push it to DTR.

# Pull NGINX from Docker Store

docker pull nginx:latest

# Re-tag NGINX

docker tag nginx:latest dtr.example.org/dev/nginx:1

# Log into DTR

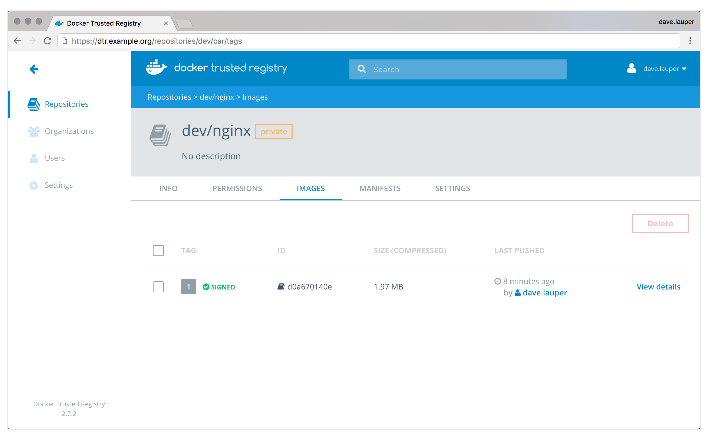
docker login dtr.example.org

# 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

| | `-- 993ad247476da081e45fdb6c28edc4462f0310a55da4acf1e08404c551d94c14.key

| `-- tuf\_keys

| `-- dtr.example.org

| `-- dev

| `-- nginx

| |-- 98a93b2e52c594de4d13d7268a4a5f28ade5fc1cb5f44cc3a4ab118572a86848.key

| `-- f7917aef77d0d4bf8204af78c0716dac6649346ebea1c4cde7a1bfa363c502ce.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** |
| --- | --- |
| root.json | Has data about other keys and their roles. This data is signed by the root key. |
| targets.json | Has data about the digest and size for an image. This data is signed by the target key. |
| snapshot.json | Has data about the version number of the root.json and targets.json files. This data is signed by the snapshot key. |
| timestamp.json | Has data about the digest, size, and version number for the snapshot.json file. This data is signed by the timestamp key. |

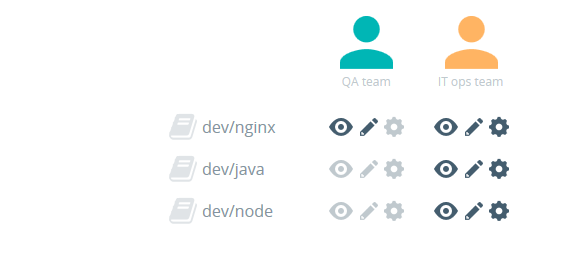
Delegate image signing

Instead of signing all the images yourself, you can delegate that task to other users.

A typical workflow looks like this:

1. A repository owner creates a repository in DTR, and initializes the trust metadata for that repository
2. Team members download a UCP client bundle and share their public key certificate with the repository owner
3. The repository owner delegates signing to the team members
4. Team members can sign images using the private keys in their UCP client bundles

In this example, the IT ops team creates and initializes trust for the dev/nginx. Then they allow users in the QA team to push and sign images in that repository.



Create a repository and initialize trust

A member of the IT ops team starts by configuring their [Notary CLI client](https://docs.docker.com/datacenter/dtr/2.3/guides/user/access-dtr/configure-your-notary-client/).

Then they create the dev/nginx repository, [initialize the trust metadata](https://docs.docker.com/datacenter/dtr/2.3/guides/user/manage-images/sign-images/) for that repository, and grant write access to members of the QA team, so that they can push images to that repository.

Ask for the public key certificates

The member of the IT ops team then asks the QA team for their public key certificate files that are part of their UCP client bundle.

If they don’t have a UCP client bundle, [they can download a new one](https://docs.docker.com/datacenter/ucp/2.2/guides/user/access-ucp/cli-based-access/).

Delegate image signing

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/qa:

# 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 <user-1-cert.pem> <user-2-cert.pem>

# Delegate trust, and add that public key with the role targets/admin

notary delegation add --publish \

dtr.example.org/dev/nginx targets/qa \

--all-paths <user-1-cert.pem> <user-2-cert.pem>

Now members from the QA team just need to [configure their Notary CLI client with UCP private keys](https://docs.docker.com/datacenter/dtr/2.3/guides/user/access-dtr/configure-your-notary-client/) before [pushing and signing images](https://docs.docker.com/datacenter/dtr/2.3/guides/user/manage-images/sign-images/) into the dev/nginx repository.

Configure your Notary client

The Docker CLI client makes it easy to sign images but to streamline that process it generates a set of private and public keys that are not tied to your UCP account. This means that you can push and sign images to DTR, but UCP doesn’t trust those images since it doesn’t know anything about the keys you’re using.

So before signing and pushing images to DTR you should:

* Configure the Notary CLI client
* Import your UCP private keys to the Notary client

This allows you to start signing images with the private keys in your UCP client bundle, that UCP can trace back to your user account.

## Download the Notary CLI client

If you’re using Docker for Mac or Docker for Windows, you already have the notary command installed.

If you’re running Docker on a Linux distribution, you can [download the latest version](https://github.com/docker/notary/releases). As an example:

# Get the latest binary

curl -L <download-url> -o notary

# Make it executable

chmod +x notary

# Move it to a location in your path

sudo mv notary /usr/bin/

## Configure the Notary CLI client

Before you use the Notary CLI client, you need to configure it to make it talk with the Notary server that’s part of DTR.

There’s two ways to do this, either by passing flags to the notary command, or using a configuration file.

### With flags

Run the Notary command with:

notary --server https://<dtr-url> --trustDir ~/.docker/trust --tlscacert <dtr-ca.pem> --help

Here’s what the flags mean:

| **Flag** | **Purpose** |
| --- | --- |
| --server | The Notary server to query |
| --trustDir | Path to the local directory where trust metadata will be stored |
| --tlscacert | Path to the DTR CA certificate. If you’ve configured your system to trust the DTR CA certificate, you don’t need to use this flag |

To avoid having to type all the flags when using the command, you can set an alias:

* Bash
* PowerShell

alias notary="notary --server https://<dtr-url> --trustDir ~/.docker/trust --tlscacert <dtr-ca.pem>"

set-alias notary "notary --server https://<dtr-url> --trustDir ~/.docker/trust --tlscacert <dtr-ca.pem>"

### With a configuration file

You can also configure Notary by creating a ~/.notary/config.json file with the following content:

{

"trust\_dir" : "~/.docker/trust",

"remote\_server": {

"url": "<dtr-url>",

"root\_ca": "<dtr-ca.pem>"

}

}

To validate your configuration, try running the notary list command on a DTR repository that already has signed images:

notary list <dtr-url>/<account>/<repository>

The command should print a list of digests for each signed image on the repository.

## Import your UCP key

The last step in configuring the Notary CLI client is to import the private key of your UCP client bundle. [Get a new client bundle if you don’t have one yet](https://docs.docker.com/datacenter/ucp/2.2/guides/user/access-ucp/cli-based-access/).

Import the private key in your UCP bundle into the Notary CLI client:

notary key import <path-to-key.pem>

The private key is copied to ~/.docker/trust, and you are prompted for a password to encrypt it.

You can validate what keys Notary knows about by running:

notary key list

The key you’ve imported should be listed with the role delegation.

Docker Trusted Registry system requirements

Docker Trusted Registry can be installed on-premises or on the cloud. Before installing, be sure your infrastructure has these requirements.

Software requirements

You can install DTR on-premises or on a cloud provider. To install DTR, all nodes must:

* Be a worker node managed by Universal Control Plane.
* Have a fixed hostname.

Ports used

When installing DTR on a node, make sure the following ports are open on that node:

| **Direction** | **Port** | **Purpose** |
| --- | --- | --- |
| in | 80/tcp | Web app and API client access to DTR. |
| in | 443/tcp | Web app and API client access to DTR. |

These ports are configurable when installing DTR.

Compatibility and maintenance lifecycle

Docker Enterprise Edition is a software subscription that includes three products:

* Docker Engine EE,
* Docker Trusted Registry,
* Docker Universal Control Plane.

[Learn more about the maintenance lifecycle for these products](https://success.docker.com/article/Compatibility_Matrix).

Install Docker Trusted Registry

Docker Trusted Registry (DTR) is a containerized application that runs on a swarm managed by Docker Universal Control Plane (UCP). It can be installed on-premises or on a cloud infrastructure.

Use these instructions to install DTR.

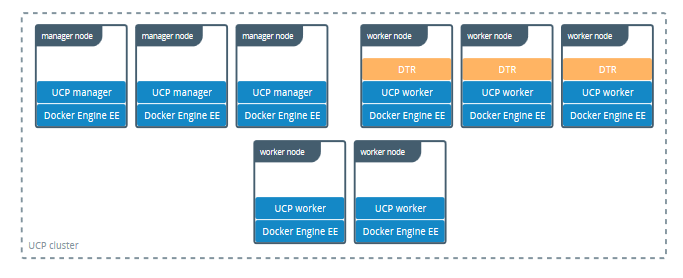
## Step 1. Validate the system requirements

The first step in installing DTR, is ensuring your infrastructure has all the [requirements DTR needs to run](https://docs.docker.com/ee/dtr/admin/install/system-requirements/).

## Step 2. Install UCP

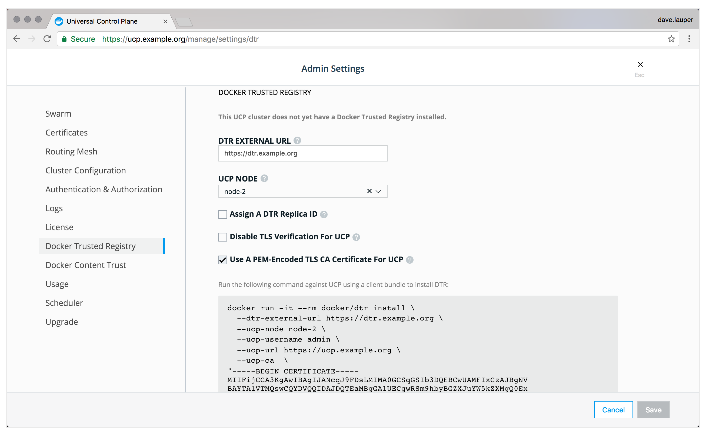
Since DTR requires Docker Universal Control Plane (UCP) to run, you need to install UCP on all the nodes where you plan to install DTR. [Learn how to install UCP](https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/).

DTR needs to be installed on a worker node that is being managed by UCP. You can’t install DTR on a standalone Docker Engine.



## Step 3. Install DTR

Once UCP is installed, navigate to the **UCP web UI**. In the **Admin Settings**, choose **Docker Trusted Registry**.



After you configure all the options, you’ll have a snippet that you can use to deploy DTR. It should look like this:

# Pull the latest version of DTR

$ docker pull docker/dtr:2.5.3

# Install DTR

$ docker run -it --rm \

docker/dtr:2.5.3 install \

--ucp-node <ucp-node-name> \

--ucp-insecure-tls

You can run that snippet on any node where Docker is installed. As an example you can SSH into a UCP node and run the DTR installer from there. By default the installer runs in interactive mode and prompts you for any additional information that is necessary. [Learn more about the installer](https://docs.docker.com/reference/dtr/2.5/cli/install/).

By default DTR is deployed with self-signed certificates, so your UCP deployment might not be able to pull images from DTR. Use the --dtr-external-url <dtr-domain>:<port> optional flag while deploying DTR, so that UCP is automatically reconfigured to trust DTR.

## Step 4. Check that DTR is running

In your browser, navigate to the Docker **Universal Control Plane** web UI, and navigate to the **Applications** screen. DTR should be listed as an application.