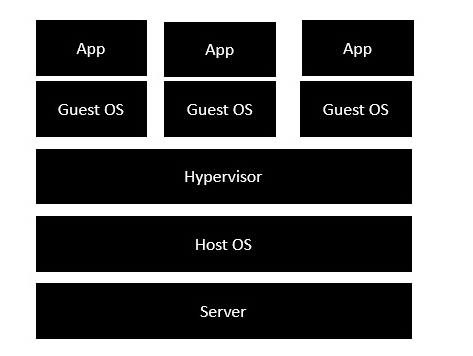
Containers and virtual machines

A **container** runs *natively* on Linux and **shares the kernel of the host machine with other containers.** It runs a discrete process, taking no more memory than any other executable, making it lightweight.

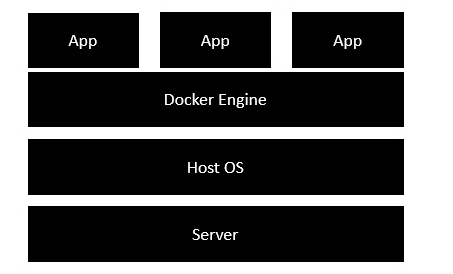
By contrast, a **virtual machine** (VM) runs a full-blown “guest” operating system with *virtual* access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need.

**Virtual Machine:**

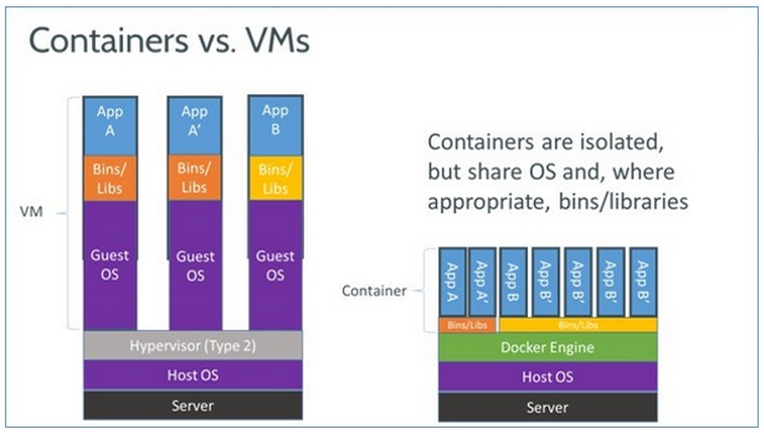


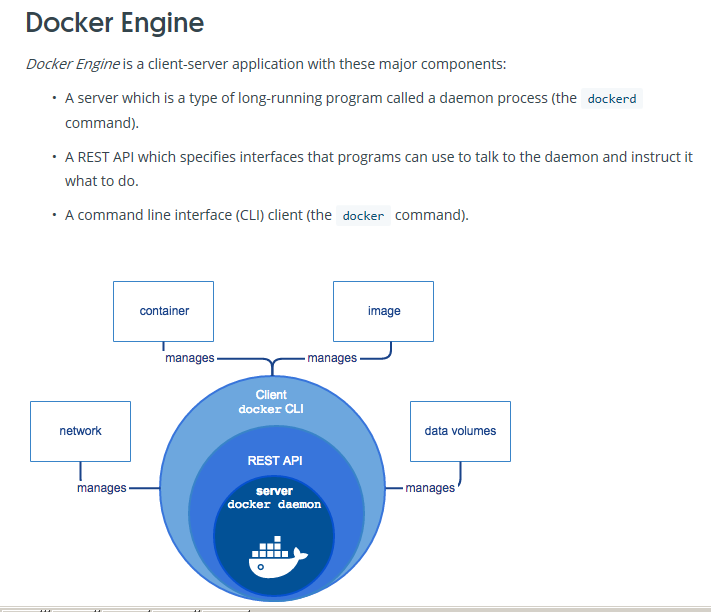
* The server is the physical server that is used to host multiple virtual machines.
* The Host OS is the base machine such as Linux or Windows.
* The Hypervisor is either VMWare or Windows Hyper V that is used to host virtual machines.
* You would then install multiple operating systems as virtual machines on top of the existing hypervisor as Guest OS.
* You would then host your applications on top of each Guest OS.

**Container**



* The server is the physical server that is used to host multiple virtual machines. So this layer remains the same.
* The Host OS is the base machine such as Linux or Windows. So this layer remains the same.
* Now comes the new generation which is the Docker engine. This is used to run the operating system which earlier used to be virtual machines as Docker containers.
* All of the Apps now run as Docker containers.

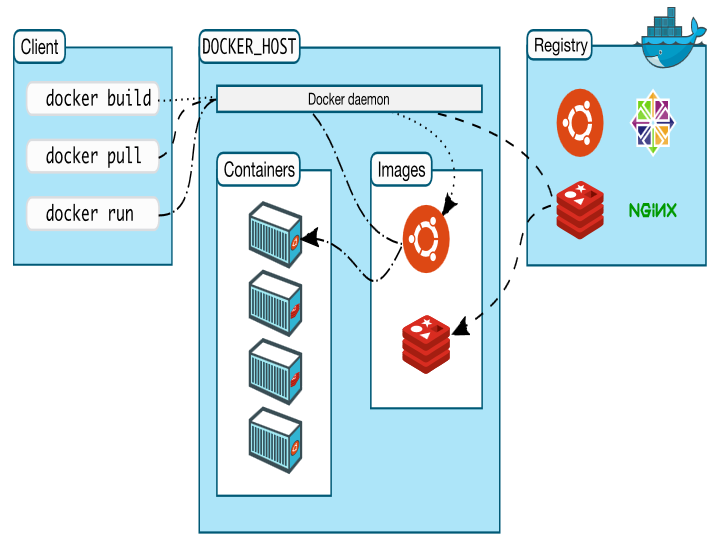




**Advantages on Docker container:**

* They start and stop much faster than virtual machines.
* They are more portable because container host environments are very consistent, no matter which type of operating system is hosting them.
* Containerized applications are easy to scale because containers can be added or subtracted quickly from an environment.
* Containers make it easy to break complex monolithic applications into smaller, modular microservices

**Architecture:**



Name space

Docker Engine uses namespaces such as the following on Linux:

* **The pid namespace:** Process isolation (PID: Process ID).
* **The net namespace:** Managing network interfaces (NET: Networking).
* **The ipc namespace:** Managing access to IPC resources (IPC: InterProcess Communication).
* **The mnt namespace:** Managing filesystem mount points (MNT: Mount).
* **The uts namespace:** Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Control groups

* Docker Engine on Linux also relies on another technology called *control groups* (cgroups). A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available to a specific container.

Union file systems

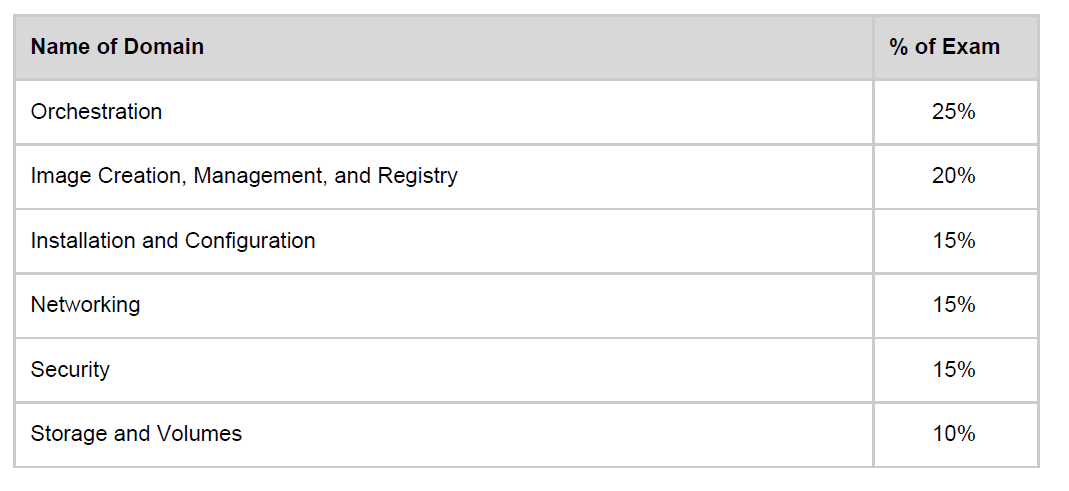
* Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

Container format

* Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a container format. The default container format is libcontainer

**Docker certification:DCA**





**Installation and Configuration:**

1. CE – COMMUNITY EDITION
2. EE – ENTERPRISE EDITION

# **Get Docker CE for CentOS**

There are two ways to install and upgrade [Docker Enterprise Edition (Docker EE)](https://www.docker.com/enterprise-edition/) on Centos:

* [YUM repository](https://docs.docker.com/install/linux/docker-ee/centos/#repo-install-and-upgrade): Set up a Docker repository and install Docker EE from it. This is the recommended approach because installation and upgrades are managed with YUM and easier to do.
* [RPM package](https://docs.docker.com/install/linux/docker-ee/centos/#package-install-and-upgrade): Download the RPM package, install it manually, and manage upgrades manually. This is useful when installing Docker EE on air-gapped systems with no access to the internet.

## **Prerequisites**

This section lists what you need to consider before installing Docker EE. Items that require action are explained below.

* Use CentOS 64-bit 7.1 and higher on x86\_64.
* Use storage driver overlay2 or devicemapper (direct-lvm mode in production).
* Find the URL for your Docker EE repo at [Docker Store](https://store.docker.com/my-content).
* Uninstall old versions of Docker.
* Remove old Docker repos from /etc/yum.repos.d/.

### Uninstall old versions

Older versions of Docker were called docker or docker-engine. If these are installed, uninstall them, along with associated dependencies.

$ sudo yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-selinux \

docker-engine-selinux \

docker-engine

It’s OK if yum reports that none of these packages are installed.

The contents of /var/lib/docker/, including images, containers, volumes, and networks, are preserved. The Docker CE package is now called docker-ce.

## **Install Docker CE**

You can install Docker CE in different ways, depending on your needs:

* Most users [set up Docker’s repositories](https://docs.docker.com/install/linux/docker-ce/centos/#install-using-the-repository) and install from them, for ease of installation and upgrade tasks. This is the recommended approach.
* Some users download the RPM package and [install it manually](https://docs.docker.com/install/linux/docker-ce/centos/#install-from-a-package) and manage upgrades completely manually. This is useful in situations such as installing Docker on air-gapped systems with no access to the internet.
* In testing and development environments, some users choose to use automated [convenience scripts](https://docs.docker.com/install/linux/docker-ce/centos/#install-using-the-convenience-script) to install Docker.

### Install using the repository

Before you install Docker CE for the first time on a new host machine, you need to set up the Docker repository. Afterward, you can install and update Docker from the repository.

#### Set up the repository

Install required packages. yum-utils provides the yum-config-manager utility, and device-mapper-persistent-data and lvm2 are required by the devicemapper storage driver.

$ sudo yum install -y yum-utils \

device-mapper-persistent-data \

lvm2

1. Use the following command to set up the **stable** repository. You always need the **stable** repository, even if you want to install builds from the **edge** or **test** repositories as well.

$ sudo yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

1. **Optional**: Enable the **edge** and **test** repositories. These repositories are included in the docker.repo file above but are disabled by default. You can enable them alongside the stable repository.

$ sudo yum-config-manager --enable docker-ce-edge

$ sudo yum-config-manager --enable docker-ce-test

Install a specific version by its fully qualified package name, which is the package name (docker-ce) plus the version string (2nd column) up to the first hyphen, separated by a hyphen (-), for example, docker-ce-18.03.0.ce.

$ sudo yum install docker-ce-<VERSION STRING>

1. Start Docker.

$ sudo systemctl start docker

1. Verify that docker is installed correctly by running the hello-world image.

$ sudo docker run hello-world

### Test Docker version

1. Run docker --version and ensure that you have a supported version of Docker:

docker --version

Docker version 17.12.0-ce, build c97c6d6

1. Run docker info or (docker version without --) to view even more details about your docker installation:

docker info

Containers: 0

Running: 0

Paused: 0

Stopped: 0

Images: 0

Server Version: 17.12.0-ce

Storage Driver: overlay2

...

### Test Docker installation

1. **Test that your installation works by running the simple Docker image,** [**hello-world**](https://hub.docker.com/_/hello-world/)**:**

docker run hello-world

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

ca4f61b1923c: Pull complete

Digest: sha256:ca0eeb6fb05351dfc8759c20733c91def84cb8007aa89a5bf606bc8b315b9fc7

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

1. ...
2. **List the** hello-world **image that was downloaded to your machine:**

docker image ls





1. **List the hello-world container (spawned by the image) which exits after displaying its message. If it were still running, you would not need the --all option**

docker container ls --all

CONTAINER ID IMAGE COMMAND CREATED STATUS

54f4984ed6a8 hello-world "/hello" 20 seconds ago Exited (0) 19 seconds ago

1. **Recap and cheat sheet**

## List Docker CLI commands

docker

docker container --help

## Display Docker version and info

docker --version

docker version

docker info

## Execute Docker image

docker run hello-world

## List Docker images

docker image ls

## List Docker containers (running, all, all in quiet mode)

docker container ls

docker container ls --all

docker container ls -aq

# **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/).

# 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.

## Hardware and 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.

### Minimum requirements

* 8GB of RAM for nodes running DTR
* 2 vCPUs for nodes running DTR
* 10GB of free disk space

### Recommended production requirements

* 16GB of RAM for nodes running DTR
* 4 vCPUs for nodes running DTR
* 25-100GB of free disk space

Note that Windows container images are typically larger than Linux ones and for this reason, you should consider provisioning more local storage for Windows nodes and for DTR setups that will store Windows container images.

## Ports used

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

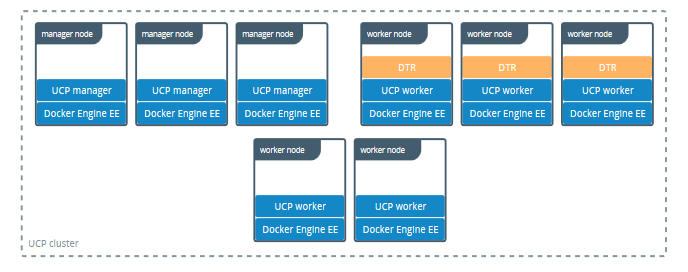
| **Direction** | **Port** | **Purpose** |
| --- | --- | --- |
| in | 80/tcp | Web app and API client access to DTR. |
| in | 443/tcp | Web app and API client access to DTR. |

These ports are configurable when installing DTR.

## 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.



UCP System requirements

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

## Hardware and software requirements

You can install UCP on-premises or on a cloud provider. Common requirements:

* [Docker Enterprise Edition](https://docs.docker.com/install/) version 17.06 or higher
* Linux kernel version 3.10 or higher
* A static IP address

### Minimum requirements

* 8GB of RAM for manager nodes or nodes running DTR
* 4GB of RAM for worker nodes
* 3GB of free disk space

### Recommended production requirements

* 16GB of RAM for manager nodes or nodes running DTR
* 4 vCPUs for manager nodes or nodes running DTR
* 25-100GB of free disk space

### RUN THE BELOW COMMAND

# Pull the latest version of UCP

$ docker image pull docker/ucp:2.2.10

# Install UCP

$ docker container run --rm -it --name ucp \

-v /var/run/docker.sock:/var/run/docker.sock \

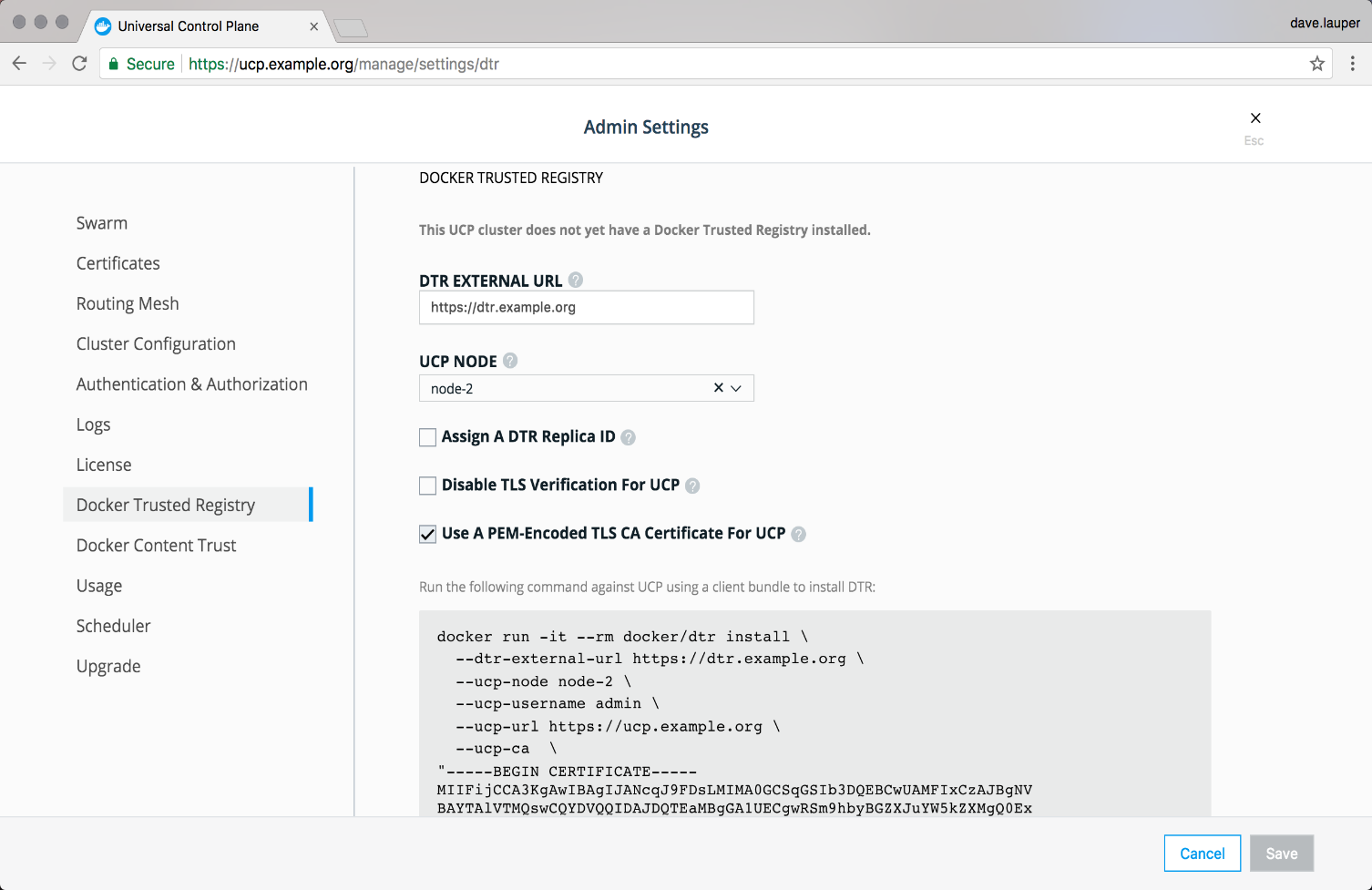
docker/ucp:2.2.10 install \

--host-address <node-ip-address> \

--interactive

## 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 \

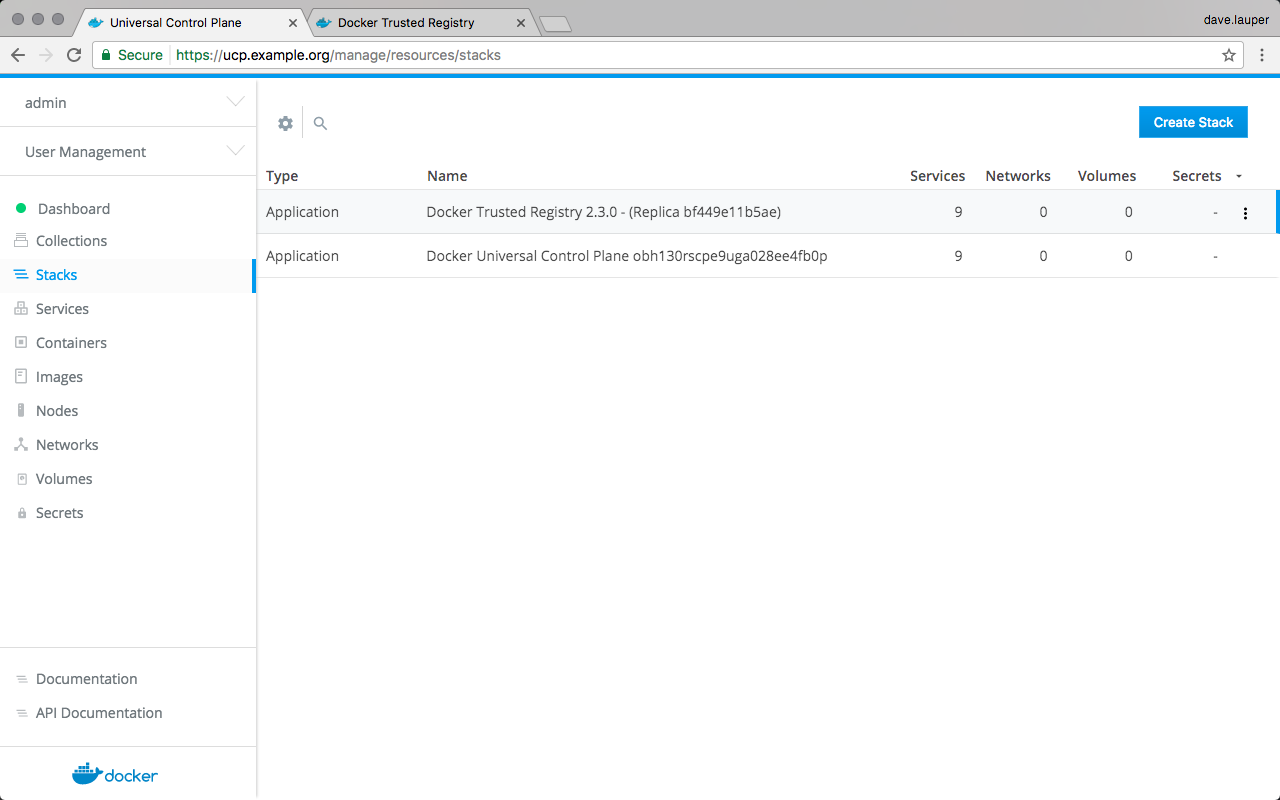
docker/dtr:2.5.3 install \

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

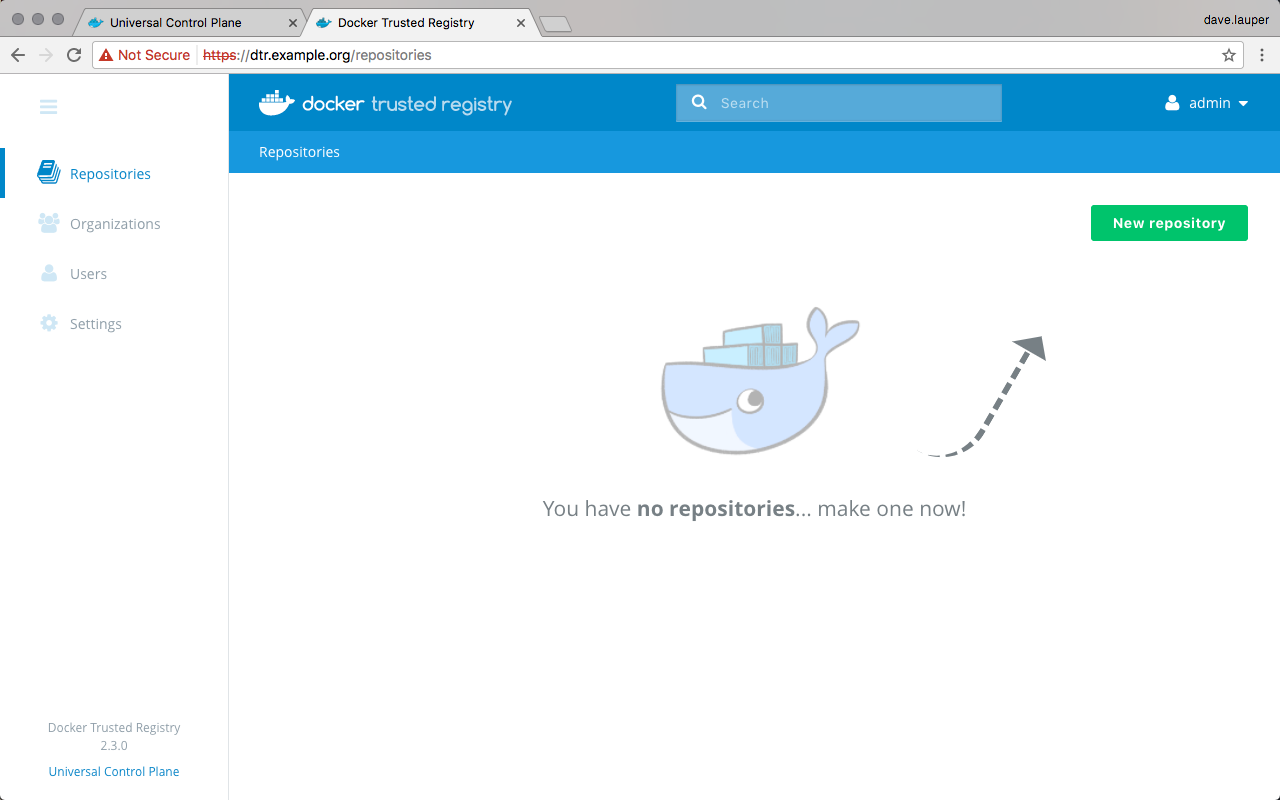
--ucp-insecure-tls

## 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.



You can also access the **DTR web UI**, to make sure it is working. In your browser, navigate to the address where you installed DTR.

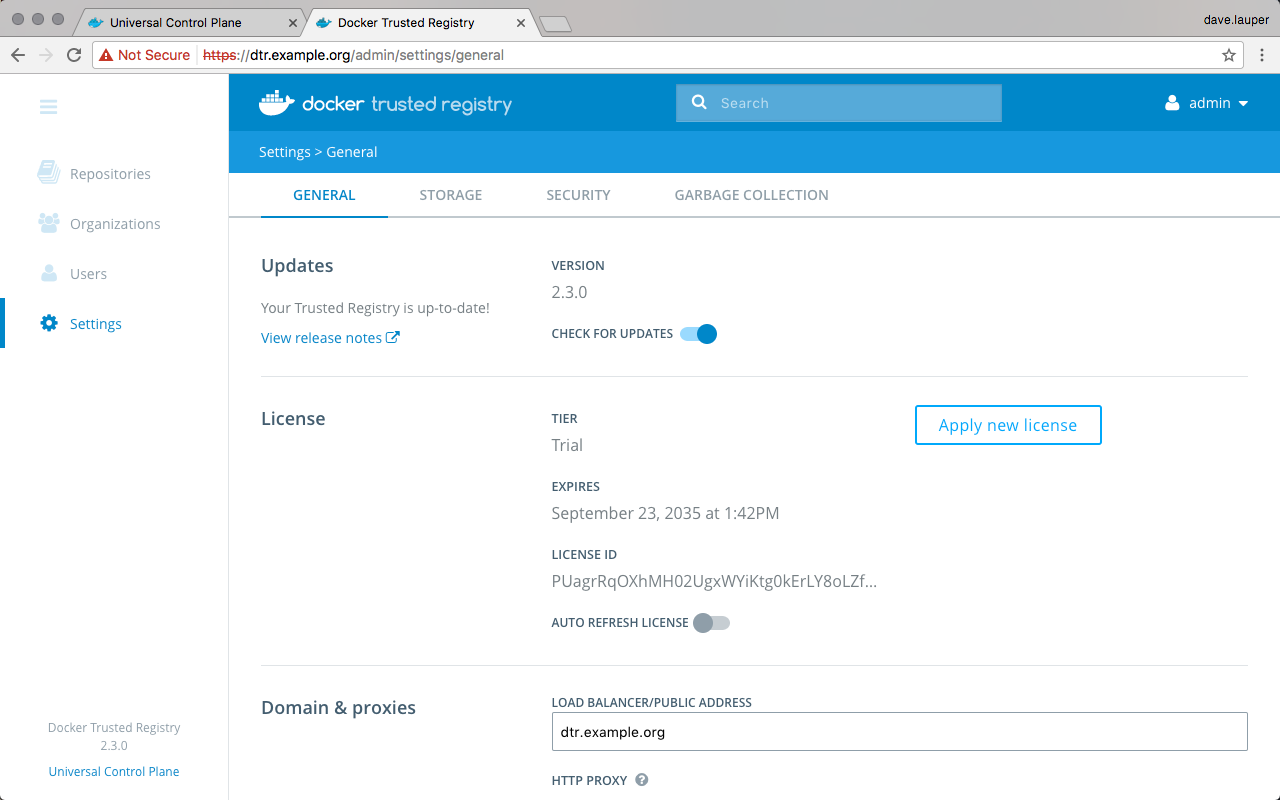


## Step 5. Configure DTR

After installing DTR, you should configure:

* The certificates used for TLS communication. [Learn more](https://docs.docker.com/ee/dtr/admin/configure/use-your-own-tls-certificates/).
* The storage backend to store the Docker images. [Lean more](https://docs.docker.com/ee/dtr/admin/configure/external-storage/).

To perform these configurations, navigate to the **Settings** page of DTR.



## Step 6. Test pushing and pulling

Now that you have a working installation of DTR, you should test that you can push and pull images to it:

* [Configure your local Docker Engine](https://docs.docker.com/ee/dtr/user/access-dtr/)
* [Create a repository](https://docs.docker.com/ee/dtr/user/manage-images/)
* [Push and pull images](https://docs.docker.com/ee/dtr/user/manage-images/pull-and-push-images/)