Authored by: Kent Lamb (/author/klamb) and Antony Bichon (/author/antonybichon17)

30 1 Share PDF (https://success.docker.com/api/articles/compatibility-matrix/pdf)

Compatibility Matrix

Article ID: KB000204

EE

Updated:12Nov2018

Docker Enterprise Edition (https://www.docker.com/enterprise-edition) is a subscription of software, support, and certification for enterprise dev and IT teams building and managing critical apps in production at scale. Docker EE provides a modern and trusted platform for all apps with integrated management and security across the app lifecycle, and includes three main technology components: the Docker daemon (fka "Engine"), Docker Trusted Registry (DTR), and Docker Universal Control Plane (UCP). Docker EE is validated and supported to work in specific operating environments as outlined in the Docker Compatibility Matrix, adhere to the Docker Maintenance Lifecycle (https://success.docker.com/article/maintenance-lifecycle), and is supported within the defined Docker Scope of Support (https://success.docker.com/article/scope-of-support) and Docker Commercial Support Service Levels (https://success.docker.com/article/commercial-support-service-levels). Refer to the Subscription Services (https://www.docker.com/subscription-services) or the End User Subscription Agreement (https://www.docker.com/docker-software-end-user-subscription-agreement) for more information. To view the latest updates and upgrade instructions, visit the release notes for daemon (https://docs.docker.com/ee/engine/release-notes/), DTR (https://docs.docker.com/ee/dtr/release-notes/), and UCP (https://docs.docker.com/ee/ucp/release-notes/).

See the Maintenance Lifecycle (https://success.docker.com/article/maintenance-lifecycle) page for more information on supported lifecycles.

Docker Enterprise Edition 2.1

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
RHEL 7.4	18.09.x	3.1.x	2.6.x	overlay2, devicemapper	Swarm mode, Kubernetes	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
RHEL 7.5	18.09.x	3.1.x	2.6.x	overlay2	Swarm mode, Kubernetes	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
SLES 12 SP2	18.09.x	3.1.x	2.6.x	overlay2, btrfs	Swarm mode, Kubernetes	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
SLES 12 SP3	18.09.x	3.1.x	2.6.x	overlay2,btrfs	Swarm mode, Kubernetes	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
Ubuntu 16.04	18.09.x	3.1.x	2.6.x	overlay2, aufs	Swarm mode, Kubernetes	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
Ubuntu 18.04	18.09.x	3.1.x	2.6.x	overlay2,aufs	Swarm mode, Kubernetes	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
CentOS 7.5	18.09.x	3.1.x	2.6.x	overlay2, devicemapper	Swarm mode, Kubernetes	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
Windows Server 2016	18.09.x	3.1.x[1]	n/a[2]	windowsfilter	Swarm mode	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
Windows Server, version 1709	18.09.x	3.1.x[1]	n/a[2]	windowsfilter	Swarm mode	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
Windows Server, version 1803	18.09.x	3.1.x[1]	n/a[2]	windowsfilter	Swarm mode	NFSv4, NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

[1] Windows Server 2016, Windows Server version 1709, and Windows Server version 1803 are only supported as worker nodes

[2]DTR does not run on Windows

Kubernetes Volume Drivers

- NFS v4 via Kubernetes e23
- AWS EFS
- AWS EBS
- Azure Disk
- Azure File

UCP and DTR Web Browser Compatibility

- Google Chrome 61 and higher
- Firefox 58 and higher
- IE 11 and higher
- Edge 16 and higher

Docker Enterprise Edition 2.0

Docker Enterprise Edition 2.0 requires Engine 17.06 and will NOT be compatible with the 18.03 Engine version. Any new features in the 18.03 Engine will not work with UCP. Please expect a newer Engine release to add this compatibility.

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
RHEL 7.3	17.06 starting with 17.06.2-ee-8	3.0.x	2.5.x	overlay2, devicemapper	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
RHEL 7.4	17.06 starting with 17.06.2-ee-8	3.0.x	2.5.x	overlay2, devicemapper	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
RHEL 7.5	17.06 starting with 17.06.2-ee-14	3.0.x starting with 3.0.2	2.5.3 and higher	overlay2	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
SLES 12 SP2	17.06 starting with 17.06.2-ee-8	3.0.x	2.5.x	btrfs	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
SLES 12 SP3	17.06 starting with 17.06.2-ee-11	3.0.x starting with 3.0.1	2.5.1 and higher	btrfs, overlay2	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
Ubuntu 14.04	17.06 starting with 17.06.2-ee-8	3.0.x	2.5.x	aufs	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
Ubuntu 16.04	17.06 starting with 17.06.2-ee-8	3.0.x	2.5.x	overlay2, aufs	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
Ubuntu 18.04	17.06 starting with 17.06.2-ee-16	3.0.x starting with 3.0.3	2.5.x starting with 2.5.4	overlay2	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
CentOS 7	17.06 starting with 17.06.2-ee-8	3.0.x	2.5.x	overlay2, devicemapper	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
Oracle Linux 7.3[1]	17.06 starting with 17.06.2-ee-8	3.0.x	2.5.x	overlay2, devicemapper[2]	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
Windows Server 2016	17.06 starting with 17.06.2-ee-8	3.0.x[3]	n/a[5]	windowsfilter	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
Windows Server, version 1709	17.06 starting with 17.06.2-ee-14	3.0.x starting with 3.0.2[3]	n/a[5]	windowsfilter	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
Windows Server, version 1803	17.06 starting with 17.06.2-ee-16	3.0.x starting with 3.0.3[3]	n/a[5]	windowsfilter	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem

IBM Z (s390x) [4]	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
SLES 12 SP3	17.06 starting with 17.06.2-ee-16	3.0.x starting with 3.0.3		overlay2	Swarm mode, Kubernetes	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem

[1] Oracle Linux 7.3 is supported with Red Hat Compatible kernel (RHCK) 3.10.0-514 and higher.

[2] The overlay2 storage driver is supported for all of EE 2.0. The devicemapper storage driver is supported starting with UCP 3.0.2. If you are upgrading from Docker EE 17.06 with devicemapper only support, it is recommended that you upgrade to EE 2.0 with UCP 3.0.2 or higher to maintain devicemapper support.

[3] Windows Server 2016, Windows Server version 1709, and Windows Server version 1803 are only supported as worker nodes.

[4]DTR does not run on IBM Z Linux.

[5]DTR does not run on Windows

Kubernetes Volume Drivers

- NFS v4 via Kubernetes e23
- AWS EFS
- AWS EBS
- Azure Disk
- Azure File

UCP and DTR Web Browser Compatibility

- Google Chrome 61 and higher
- Firefox 58 and higher
- IE 11 and higher
- Edge 16 and higher

Docker 18.03 EE Engine

WARNING: Docker Engine 18.03 is a standalone engine release. UCP and DTR will not run on Docker 18.03 EE Engine. Please see the Docker Enterprise Engine 18.03 FAQs (https://success.docker.com/article/engine-18-03-faqs) for details.

OS Distribution (x86_64)	Enterprise Engine	Storage Driver	Orchestration
RHEL 7.3	18.03.1-ee-1	overlay2, devicemapper	Swarm mode
RHEL 7.4	18.03.1-ee-1	overlay2, devicemapper	Swarm mode
RHEL 7.5	18.03.1-ee-1	overlay2, devicemapper	Swarm mode
SLES 12 SP2	18.03.1-ee-1	overlay2 with XFS, btrfs	Swarm mode
SLES 12 SP3	18.03.1-ee-1	overlay2 with XFS, btrfs	Swarm mode
Ubuntu 14.04	18.03.1-ee-1	aufs	Swarm mode
Ubuntu 16.04	18.03.1-ee-1	overlay2, aufs	Swarm mode
Ubuntu 18.04	18.03.1-ee-1	overlay2	Swarm mode
CentOS 7	18.03.1-ee-1	overlay2, devicemapper	Swarm mode
Oracle Linux 7.3[1]	18.03.1-ee-1	overlay2	Swarm mode
Windows Server 2016	18.03.1-ee-1	windowsfilter	Swarm mode[2]

OS Distribution (x86_64)	Enterprise Engine	Storage Driver	Orchestration
Windows Server, version 1709	18.03.1-ee-1	windowsfilter	Swarm mode[2]
Windows Server, version 1803	18.03.1-ee-1	windowsfilter	Swarm mode[2]

[1] Oracle Linux 7.3 is supported with Red Hat Compatible kernel (RHCK) 3.10.0-514 and higher.

[2] Windows Server 2016, Windows Server version 1709, and Windows Server version 1803 are only supported as worker nodes.

Docker Enterprise Edition 17.06 and earlier

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
RHEL 7.5	17.06 starting with 17.06.2-ee-16	2.2.x starting with 2.2.11	2.4.x starting with 2.4.6	overlay2	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift, Local Filesystem
RHEL 7.1-7.4	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]	devicemapper or overlay2[6]	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
SLES 12 SP2	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]	btrfs	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem
Ubuntu 14.04	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]	aufs3	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem
Ubuntu 16.04	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]	aufs3	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
Ubuntu 18.04	17.06 starting with 17.06.2-ee-16	2.2.x starting with 2.2.11	2.4.x starting with 2.4.6[4]	aufs3	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem
CentOS 7	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]	devicemapper	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem
Oracle Linux 7.3[1]	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]	devicemapper	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem

OS Distribution (x86_64)	Enterprise Engine	UCP	DTR	Storage Driver	Orchestration	DTR Storage Backend
Windows Server 2016[2]	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4,5]	windowsfilter	Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem

IBM Z (s390x) [3]	**Enterprise Engine**	**UCP**	**DTR**	**Storage Driver**	**Orchestration**	**DTR Storage Backend**
RHEL 7.4	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]		Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem
SLES 12R2	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]		Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem
Ubuntu 16.04	17.06.1-ee-1 and higher	2.2.x starting with 2.2.2	2.3.x to 2.4.x[4]		Swarm mode	NFSv3, Amazon S3, S3 Compliant Alternatives, Azure Storage (Blob), Google Cloud Storage, OpenStack Swift. Local Filesystem

[1] Oracle Linux 7.3 is supported with Red Hat Compatible kernel (RHCK) 3.10.0-514 and higher.

[2] Windows Server 2016 is only supported as worker nodes.

[3] Starting with UCP 2.2.0, you can join worker nodes running on IBM Z(s390x) architecture. Starting with UCP 2.2.4, you can join manager nodes running on IBM Z(s390x) architecture. SELinux policies for Docker containers are not supported for Docker EE running on IBM Z.

[4] DTR 2.3.z and 2.4.z are only compatible with UCP 2.2.2 and greater.

[5] Windows Server 2016, Windows Server version 1709, and Windows Server version 1803 are only supported as worker nodes.

[6] Overlay2 is recommended with Docker EE engine 17.06.02-ee-5 and higher. RHEL 7.1 and 7.2 do not support overlay2.

Docker Store (https://store.docker.com/)

Get Docker (https://www.docker.com/get-docker)

Docker for Mac (https://www.docker.com/docker-mac)

Docker for Windows(PC) (https://www.docker.com/docker-windows)

Docker for AWS (https://www.docker.com/docker-aws)

Docker for Azure (https://www.docker.com/docker-azure)

Docker for Windows Server (https://www.docker.com/docker-windows-server)

Docker for Debian (https://www.docker.com/docker-debian)

Docker for Fedora® (https://www.docker.com/docker-fedora)

Docker for Oracle Linux (https://www.docker.com/docker-oracle-linux)

Docker for RHEL (https://www.docker.com/docker-red-hat-enterprise-linux-rhel)

Docker for SLES (https://www.docker.com/docker-suse-linux-enterprise-server-sles)

Docker for Ubuntu (https://www.docker.com/docker-ubuntu)

Documentation (https://docs.docker.com/)

Blog (https://blog.docker.com/)

RSS Feed (https://blog.docker.com/feed/)

Training (https://training.docker.com/)

Knowledge Base (https://success.docker.com/kbase)

Resources (https://www.docker.com/products/resources)

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

Open Source (https://www.docker.com/technologies/overview)

Events (https://www.docker.com/community/events)

Forums (https://forums.docker.com/)

Docker Captains (https://www.docker.com/community/docker-captains)

Scholarships (https://www.docker.com/community-partnerships)

Community News (https://blog.docker.com/curated/)

Status (http://status.docker.com/) Security (https://www.docker.com/docker-security)

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

(http: (http: (https (https (https (https (http: (http:

Configure logging drivers

Estimated reading time: 6 minutes

Docker includes multiple logging mechanisms to help you get information from running containers and services (https://docs.docker.com/engine/admin/logging/view_container_logs/). These mechanisms are called logging drivers.

Each Docker daemon has a default logging driver, which each container uses unless you configure it to use a different logging driver.

In addition to using the logging drivers included with Docker, you can also implement and use logging driver plugins (https://docs.docker.com/engine/admin/logging/plugins/).

Configure the default logging driver

To configure the Docker daemon to default to a specific logging driver, set the value of `log-driver` to the name of the logging driver in the `daemon.json` file, which is located in `/etc/docker/` on Linux hosts or `C:\ProgramData\docker\config\` on Windows server hosts. The default logging driver is `json-file`. The following example explicitly sets the default logging driver to `syslog`:

```
{
  "log-driver": "syslog"
}
```

If the logging driver has configurable options, you can set them in the `daemon.json` file as a JSON array with the key `log-opts`. The following example sets two configurable options on the `json-file` logging driver:

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3",
    "labels": "production_status",
    "env": "os,customer"
  }
}
```

> Note: `log-opt` configuration options in the `daemon.json` configuration file must be provided as strings. Boolean and numeric values (such as the value for `max-file` in the example above) must therefore be enclosed in quotes (`"`).

If you do not specify a logging driver, the default is `json-file` . Thus, the default output for commands such as `docker inspect <CONTAINER>` is JSON.

To find the current default logging driver for the Docker daemon, run `docker info` and search for `Logging Driver` . You can use the following command:

```
$ docker info --format '{{.LoggingDriver}}'

json-file
```

Configure the logging driver for a container

When you start a container, you can configure it to use a different logging driver than the Docker daemon's default, using the `--log-driver` flag. If the logging driver has configurable options, you can set them using one or more instances of the `--log-opt <NAME>=<VALUE>` flag. Even if the container uses the default logging driver, it can use different configurable options.

The following example starts an Alpine container with the `none` logging driver.

```
$ docker run -it --log-driver none alpine ash
```

To find the current logging driver for a running container, if the daemon is using the `json-file` logging driver, run the following `docker inspect` command, substituting the container name or ID for `<CONTAINER>` :

```
$ docker inspect -f '{{.HostConfig.LogConfig.Type}}' <CONTAINER>

json-file
```

Configure the delivery mode of log messages from container to log driver

Docker provides two modes for delivering messages from the container to the log driver:

- (default) direct, blocking delivery from container to driver
- non-blocking delivery that stores log messages in an intermediate per-container ring buffer for consumption by driver

The `non-blocking` message delivery mode prevents applications from blocking due to logging back pressure. Applications are likely to fail in unexpected ways when STDERR or STDOUT streams block.

> ⊗ WARNING: When the buffer is full and a new message is enqueued, the oldest message in memory is dropped. Dropping messages is often preferred to blocking the log-writing process of an application.

The `mode` log option controls whether to use the `blocking` (default) or `non-blocking` message delivery.

The `max-buffer-size` log option controls the size of the ring buffer used for intermediate message storage when `mode` is set to `non-blocking` . `max-buffer-size` defaults to 1 megabyte.

The following example starts an Alpine container with log output in non-blocking mode and a 4 megabyte buffer:

```
$ docker run -it --log-opt mode=non-blocking --log-opt max-buffer-si
ze=4m alpine ping 127.0.0.1
```

Use environment variables or labels with logging drivers

Some logging drivers add the value of a container's `--env|-e` or `--label` flags to the container's logs. This example starts a container using the Docker daemon's default logging driver (let's assume `json-file`) but sets the environment variable `os=ubuntu` .

```
$ docker run -dit --label production_status=testing -e os=ubuntu alp
ine sh
```

If the logging driver supports it, this adds additional fields to the logging output. The following output is generated by the `json-file` logging driver:

```
"attrs":{"production_status":"testing","os":"ubuntu"}
```

Supported logging drivers

The following logging drivers are supported. See the link to each driver's documentation for its configurable options, if applicable. If you are using logging driver plugins (https://docs.docker.com/engine/admin/logging/plugins/), you may see more options.

Driver	Description
`none`	No logs are available for the container and `docker logs` does not return any output.

Driver	Description
`json-file` (https://docs.docker.com/config/containers/logging/json-file/)	The logs are formatted as JSON. The default logging driver for Docker.
`syslog` (https://docs.docker.com/config/containers/logging/syslog/)	Writes logging messages to the `syslog` facility. The `syslog` daemon must be running on the host machine.
`journald` (https://docs.docker.com/config/containers/logging/journald/)	Writes log messages to `journald` . The `journald` daemon must be running on the host machine.
`gelf` (https://docs.docker.com/config/containers/logging/gelf/)	Writes log messages to a Graylog Extended Log Format (GELF) endpoint such as Graylog or Logstash.

Driver	Description
`fluentd` (https://docs.docker.com/config/containers/logging/fluentd/)	Writes log messages to `fluentd` (forward input). The `fluentd` daemon must be running on the host machine.
`awslogs` (https://docs.docker.com/config/containers/logging/awslogs/)	Writes log messages to Amazon CloudWatch Logs.
`splunk` (https://docs.docker.com/config/containers/logging/splunk/)	Writes log messages to `splunk` using the HTTP Event Collector.
`etwlogs` (https://docs.docker.com/config/containers/logging/etwlogs/)	Writes log messages as Event Tracing for Windows (ETW) events. Only available on Windows platforms.
`gcplogs` (https://docs.docker.com/config/containers/logging/gcplogs/)	Writes log messages to Google Cloud Platform (GCP) Logging.

Driver	Description
`logentries` (https://docs.docker.com/config/containers/logging/logentries/)	Writes log messages to Rapid7 Logentries.

Limitations of logging drivers

The `docker logs` command is not available for drivers other than `json-file` and `journald` .

docker (https://docs.docker.com/glossary/?term=docker), logging (https://docs.docker.com/glossary/?term=logging), driver (https://docs.docker.com/glossary/?term=driver)

Administer and maintain a swarm of Docker Engines

Estimated reading time: 16 minutes

When you run a swarm of Docker Engines, manager nodes are the key components for managing the swarm and storing the swarm state. It is important to understand some key features of manager nodes to properly deploy and maintain the swarm.

Refer to How nodes work (https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/) for a brief overview of Docker Swarm mode and the difference between manager and worker nodes.

Operate manager nodes in a swarm

Swarm manager nodes use the Raft Consensus Algorithm (https://docs.docker.com/engine/swarm/raft/) to manage the swarm state. You only need to understand some general concepts of Raft in order to manage a swarm.

There is no limit on the number of manager nodes. The decision about how many manager nodes to implement is a trade-off between performance and fault-tolerance. Adding manager nodes to a swarm makes the swarm more fault-tolerant. However, additional manager nodes reduce write performance because more nodes must acknowledge proposals to update the swarm state. This means more network round-trip traffic.

Raft requires a majority of managers, also called the quorum, to agree on proposed updates to the swarm, such as node additions or removals. Membership operations are subject to the same constraints as state replication.

Maintain the quorum of managers

If the swarm loses the quorum of managers, the swarm cannot perform management tasks. If your swarm has multiple managers, always have more than two. To maintain quorum, a majority of managers must be available. An odd number of managers is recommended, because the next even number does not make the quorum easier to keep. For instance, whether you have 3 or 4 managers, you can still only lose 1 manager and maintain the quorum. If you have 5 or 6 managers, you can still only lose two.

Even if a swarm loses the quorum of managers, swarm tasks on existing worker nodes continue to run. However, swarm nodes cannot be added, updated, or removed, and new or existing tasks cannot be started, stopped, moved, or updated.

See Recovering from losing the quorum (/engine/swarm/admin_guide/#recovering-from-losing-the-quorum) for troubleshooting steps if you do lose the quorum of managers.

Configure the manager to advertise on a static IP address

When initiating a swarm, you must specify the `--advertise-addr` flag to advertise your address to other manager nodes in the swarm. For more information, see Run Docker Engine in swarm mode (https://docs.docker.com/engine/swarm/swarm-mode/#configure-the-advertise-address). Because manager nodes are meant to be a stable component of the infrastructure, you should use a *fixed IP address* for the advertise address to prevent the swarm from becoming unstable on machine reboot.

If the whole swarm restarts and every manager node subsequently gets a new IP address, there is no way for any node to contact an existing manager. Therefore the swarm is hung while nodes try to contact one another at their old IP addresses.

Dynamic IP addresses are OK for worker nodes.

Add manager nodes for fault tolerance

You should maintain an odd number of managers in the swarm to support manager node failures. Having an odd number of managers ensures that during a network partition, there is a higher chance that the quorum remains available to process requests if the network is partitioned into two sets. Keeping the quorum is not guaranteed if you encounter more than two network partitions.

Swarm Size	Majority	Fault Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3
9	5	4

For example, in a swarm with *5 nodes*, if you lose *3 nodes*, you don't have a quorum. Therefore you can't add or remove nodes until you recover one of the unavailable manager nodes or recover the swarm with disaster recovery commands. See Recover from disaster (/engine/swarm/admin_guide/#recover-from-disaster).

While it is possible to scale a swarm down to a single manager node, it is impossible to demote the last manager node. This ensures you maintain access to the swarm and that the swarm can still process requests. Scaling down to a single manager is an unsafe operation and is not recommended. If the last node leaves the swarm unexpectedly during the demote operation, the swarm becomes unavailable until you reboot the node or restart with `--force-new-cluster` .

You manage swarm membership with the `docker swarm` and `docker node` subsystems. Refer to Add nodes to a swarm (https://docs.docker.com/engine/swarm/join-nodes/) for more information on how to add worker nodes and promote a worker node to be a manager.

Distribute manager nodes

In addition to maintaining an odd number of manager nodes, pay attention to datacenter topology when placing managers. For optimal fault-tolerance, distribute manager nodes across a minimum of 3 availability-zones to support failures of an entire set of machines or common maintenance scenarios. If you suffer a failure in any of those zones, the swarm should maintain the quorum of manager nodes available to process requests and rebalance workloads.

Swarm manager nodes	Repartition (on 3 Availability zones)
3	1-1-1
5	2-2-1
7	3-2-2
9	3-3-3

Run manager-only nodes

By default manager nodes also act as a worker nodes. This means the scheduler can assign tasks to a manager node. For small and non-critical swarms assigning tasks to managers is relatively low-risk as long as you schedule services using resource constraints for *cpu* and *memory*.

However, because manager nodes use the Raft consensus algorithm to replicate data in a consistent way, they are sensitive to resource starvation. You should isolate managers in your swarm from processes that might block swarm operations like swarm heartbeat or leader elections.

To avoid interference with manager node operation, you can drain manager nodes to make them unavailable as worker nodes:

```
docker node update --availability drain <NODE>
```

When you drain a node, the scheduler reassigns any tasks running on the node to other available worker nodes in the swarm. It also prevents the scheduler from assigning tasks to the node.

Add worker nodes for load balancing

Add nodes to the swarm (https://docs.docker.com/engine/swarm/join-nodes/) to balance your swarm's load. Replicated service tasks are distributed across the swarm as evenly as possible over time, as long as the worker nodes are matched to the requirements of the services. When limiting a service to run on only specific types of nodes, such as nodes with a specific number of CPUs or amount of memory, remember that worker nodes that do not meet these requirements cannot run these tasks.

Monitor swarm health

You can monitor the health of manager nodes by querying the docker `nodes` API in JSON format through the `/nodes` HTTP endpoint. Refer to the nodes API documentation (https://docs.docker.com/engine/api/v1.25/#tag/Node) for more information.

From the command line, run `docker node inspect <id-node>` to query the nodes. For instance, to query the reachability of the node as a manager:

```
docker node inspect manager1 --format "{{ .ManagerStatus.Reachabilit
y }}"
reachable
```

To query the status of the node as a worker that accept tasks:

```
docker node inspect manager1 --format "{{ .Status.State }}"
ready
```

From those commands, we can see that `manager1` is both at the status `reachable` as a manager and `ready` as a worker.

An `unreachable` health status means that this particular manager node is unreachable from other manager nodes. In this case you need to take action to restore the unreachable manager:

- Restart the daemon and see if the manager comes back as reachable.
- Reboot the machine.
- If neither restarting or rebooting work, you should add another manager node or promote a worker to be a manager node. You also need to cleanly remove the failed node entry from the manager set with `docker node demote <NODE>` and `docker node rm <id-node>` .

Alternatively you can also get an overview of the swarm health from a manager node with `docker node ls` :

```
docker node ls
ID                           HOSTNAME  MEMBERSHIP  STATUS  AVAILABIL
ITY  MANAGER STATUS
1mhtdwhvsgr3c26xxbnzdc3yp    node05    Accepted    Ready   Active
516pacagkqp2xc3fk9t1dhjor    node02    Accepted    Ready   Active
     Reachable
9ifojw8of78kkusuc4a6c23fx *  node01    Accepted    Ready   Active
     Leader
ax11wdpwrrb6db3mfjydscgk7    node04    Accepted    Ready   Active
bb1nrq2cswhtbg4mrsqnlx1ck    node03    Accepted    Ready   Active
     Reachable
di9wxgz8dtuh9d2hn089ecqkf    node06    Accepted    Ready   Active
```

Troubleshoot a manager node

You should never restart a manager node by copying the `raft` directory from another node. The data directory is unique to a node ID. A node can only use a node ID once to join the swarm. The node ID space should be globally unique.

To cleanly re-join a manager node to a cluster:

1. To demote the node to a worker, run `docker node demote <NODE>` .
2. To remove the node from the swarm, run `docker node rm <NODE>` .
3. Re-join the node to the swarm with a fresh state using `docker swarm join` .

For more information on joining a manager node to a swarm, refer to Join nodes to a swarm (https://docs.docker.com/engine/swarm/join-nodes/).

Forcibly remove a node

In most cases, you should shut down a node before removing it from a swarm with the `docker node rm` command. If a node becomes unreachable, unresponsive, or compromised you can forcefully remove the node without shutting it down by passing the `--force` flag. For instance, if `node9` becomes compromised:

```
$ docker node rm node9

Error response from daemon: rpc error: code = 9 desc = node node9 is
 not down and can't be removed

$ docker node rm --force node9

Node node9 removed from swarm
```

Before you forcefully remove a manager node, you must first demote it to the worker role. Make sure that you always have an odd number of manager nodes if you demote or remove a manager.

Back up the swarm

Docker manager nodes store the swarm state and manager logs in the `/var/lib/docker/swarm/` directory. In 1.13 and higher, this data includes the keys used to encrypt the Raft logs. Without these keys, you cannot restore the swarm.

You can back up the swarm using any manager. Use the following procedure.

1. If the swarm has auto-lock enabled, you need the unlock key to restore the swarm from backup. Retrieve the unlock key if necessary and store it in a safe location. If you are unsure, read Lock your swarm to protect its encryption key (https://docs.docker.com/engine/swarm/swarm_manager_locking/).
2. Stop Docker on the manager before backing up the data, so that no data is being changed during the backup. It is possible to take a backup while the manager is running (a “hot” backup), but this is not recommended and your

results are less predictable when restoring. While the manager is down, other nodes continue generating swarm data that is not part of this backup.

> Note: Be sure to maintain the quorum of swarm managers. During the time that a manager is shut down, your swarm is more vulnerable to losing the quorum if further nodes are lost. The number of managers you run is a trade-off. If you regularly take down managers to do backups, consider running a 5-manager swarm, so that you can lose an additional manager while the backup is running, without disrupting your services.

3. Back up the entire `/var/lib/docker/swarm` directory.
4. Restart the manager.

To restore, see Restore from a backup (/engine/swarm/admin_guide/#restore-from-a-backup).

Recover from disaster

Restore from a backup

After backing up the swarm as described in Back up the swarm (/engine/swarm/admin_guide/#back-up-the-swarm), use the following procedure to restore the data to a new swarm.

1. Shut down Docker on the target host machine for the restored swarm.
2. Remove the contents of the `/var/lib/docker/swarm` directory on the new swarm.
3. Restore the `/var/lib/docker/swarm` directory with the contents of the backup.

> ✓ Note: The new node uses the same encryption key for on-disk storage as the old one. It is not possible to change the on-disk storage encryption keys at this time.
>
> In the case of a swarm with auto-lock enabled, the unlock key is also the same as on the old swarm, and the unlock key is needed to restore the swarm.

4. Start Docker on the new node. Unlock the swarm if necessary. Re-initialize the swarm using the following command, so that this node does not attempt to connect to nodes that were part of the old swarm, and presumably no longer exist.

   ```
   $ docker swarm init --force-new-cluster
   ```

5. Verify that the state of the swarm is as expected. This may include application-specific tests or simply checking the output of `docker service ls` to be sure that all expected services are present.
6. If you use auto-lock, rotate the unlock key (https://docs.docker.com/engine/swarm/swarm_manager_locking/#rotate-the-unlock-key).
7. Add manager and worker nodes to bring your new swarm up to operating capacity.
8. Reinstate your previous backup regimen on the new swarm.

Recover from losing the quorum

Swarm is resilient to failures and the swarm can recover from any number of temporary node failures (machine reboots or crash with restart) or other transient errors. However, a swarm cannot automatically recover if it loses a quorum. Tasks on existing worker nodes continue to run, but administrative tasks are not possible, including scaling or updating services and joining or removing nodes from the swarm. The best way to recover is to bring the missing manager nodes back online. If that is not possible, continue reading for some options for recovering your swarm.

In a swarm of `N` managers, a quorum (a majority) of manager nodes must always be available. For example, in a swarm with 5 managers, a minimum of 3 must be operational and in communication with each other. In other words, the swarm can tolerate up to `(N-1)/2` permanent failures beyond which requests involving swarm management cannot be processed. These types of failures include data corruption or hardware failures.

If you lose the quorum of managers, you cannot administer the swarm. If you have lost the quorum and you attempt to perform any management operation on the swarm, an error occurs:

```
Error response from daemon: rpc error: code = 4 desc = context deadl
ine exceeded
```

The best way to recover from losing the quorum is to bring the failed nodes back online. If you can't do that, the only way to recover from this state is to use the `--force-new-cluster` action from a manager node. This removes all managers except the manager the command was run from. The quorum is achieved because there is now only one manager. Promote nodes to be managers until you have the desired number of managers.

```
# From the node to recover
docker swarm init --force-new-cluster --advertise-addr node01:2377
```

When you run the `docker swarm init` command with the `--force-new-cluster` flag, the Docker Engine where you run the command becomes the manager node of a single-node swarm which is capable of managing and running services. The manager has all the previous information about services and tasks, worker nodes are still part of the swarm, and services are still running. You need to add or re-add manager nodes to achieve your previous task distribution and ensure that you have enough managers to maintain high availability and prevent losing the quorum.

Force the swarm to rebalance

Generally, you do not need to force the swarm to rebalance its tasks. When you add a new node to a swarm, or a node reconnects to the swarm after a period of unavailability, the swarm does not automatically give a workload to the idle node. This is a design decision. If the swarm periodically shifted tasks to different nodes for the sake of balance, the clients using those tasks would be disrupted. The goal is to avoid disrupting running services for the sake of balance across the swarm. When new tasks start, or when a node with running tasks becomes unavailable, those tasks are given to less busy nodes. The goal is eventual balance, with minimal disruption to the end user.

In Docker 1.13 and higher, you can use the `--force` or `-f` flag with the `docker service update` command to force the service to redistribute its tasks across the available worker nodes. This causes the service tasks to restart. Client applications may be disrupted. If you have configured it, your service uses a rolling update (https://docs.docker.com/engine/swarm/swarm-tutorial/rolling-update/).

If you use an earlier version and you want to achieve an even balance of load across workers and don't mind disrupting running tasks, you can force your swarm to re-balance by temporarily scaling the service upward. Use `docker service inspect --pretty <servicename>` to see the configured scale of a service. When you use `docker service scale` , the nodes with the lowest number of tasks are targeted to receive the new workloads. There may be multiple under-loaded nodes in your swarm. You may need to scale the service up by modest increments a few times to achieve the balance you want across all the nodes.

When the load is balanced to your satisfaction, you can scale the service back down to the original scale. You can use `docker service ps` to assess the current balance of your service across nodes.

See also `docker service scale` (https://docs.docker.com/engine/reference/commandline/service_scale/) and `docker service ps` (https://docs.docker.com/engine/reference/commandline/service_ps/).

docker (https://docs.docker.com/glossary/?term=docker), container (https://docs.docker.com/glossary/?term=container), swarm (https://docs.docker.com/glossary/?term=swarm), manager

(https://docs.docker.com/glossary/?term=manager), raft
(https://docs.docker.com/glossary/?term=raft)

Create and manage users

Estimated reading time: 1 minute

✓ These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

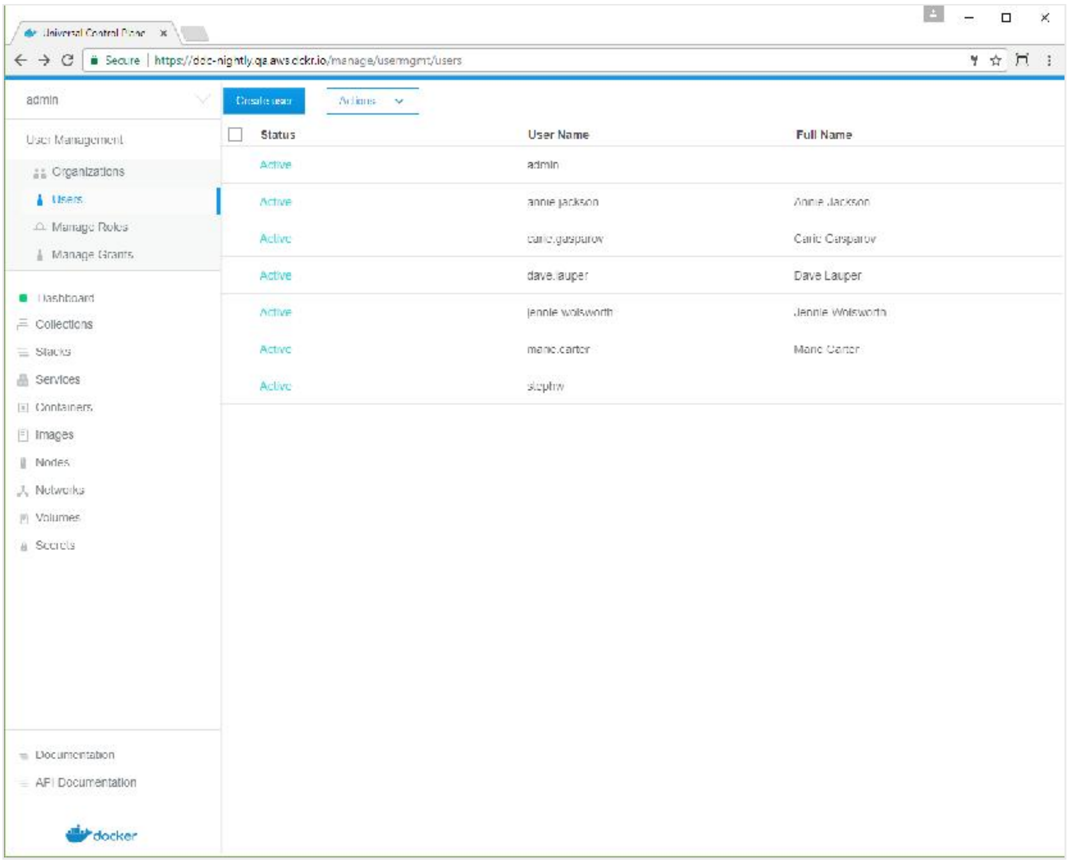
2.2.14 ▾

Docker Universal Control Plane provides built-in authentication and also integrates with LDAP directory services. If you want to manage users and groups from your organization's directory, choose LDAP. Learn to integrate with an LDAP directory (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/external-auth/).

When using the UCP built-in authentication, you need to create users and optionally grant them UCP administrator permissions.

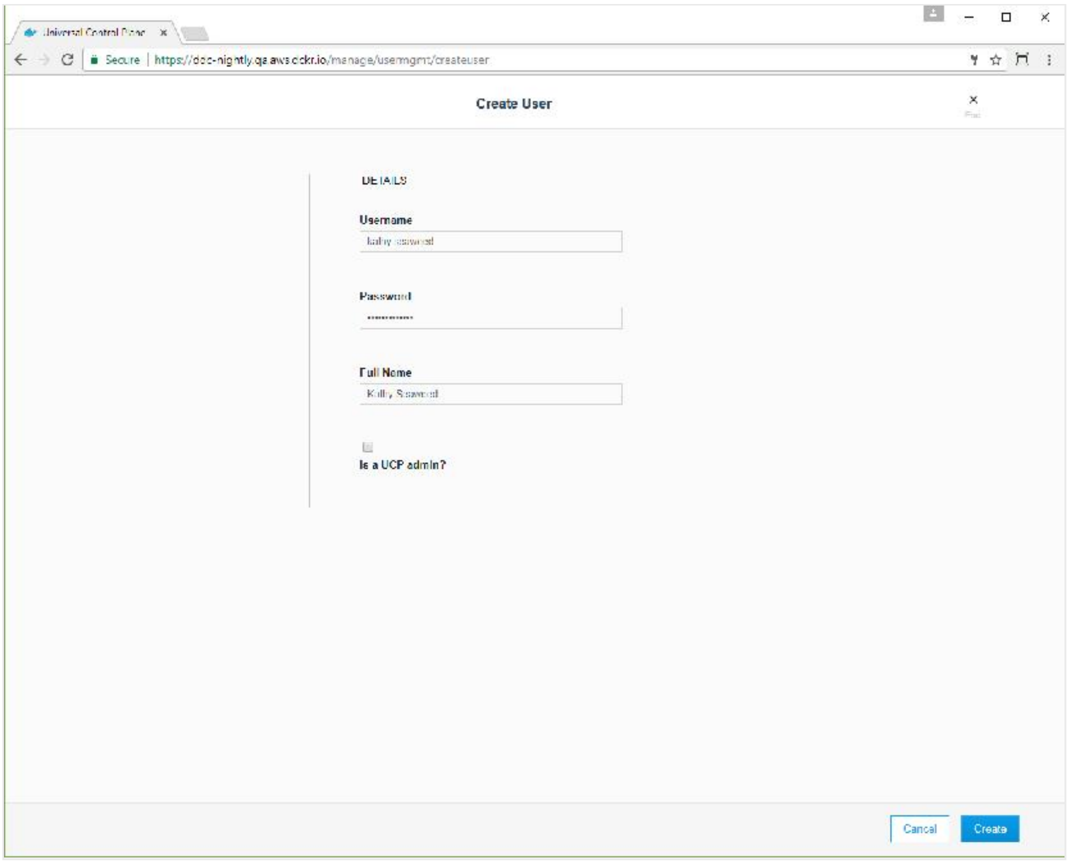
Each new user gets a default permission level so that they can access the swarm.

To create a new user, go to the UCP web UI, and navigate to the Users page.



Status	User Name	Full Name
Active	admin	
Active	annie.jackson	Annie Jackson
Active	carlo.gasparov	Carlo Gasparov
Active	dave.lauper	Dave Lauper
Active	jennie.wolsworth	Jennie Wolsworth
Active	marc.carter	Marc Carter
Active	stephw	

Click the Create user button, and fill-in the user information.



https://doc-nightly.qa.aws.dckr.io/manage/usermgmt/createuser
Create User
DETAILS
Cancel
Create

Username: kathy.seaword

Password: ••••••••••

Full Name: Kathy Seaword

Is a UCP admin? [ ]

Check the `Is a UCP admin?` option, if you want to grant permissions for the user to change the swarm configuration and manage grants, roles, and collections.

Finally, click the Create button to create the user.

Where to go next

- 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/)

authorize (https://docs.docker.com/glossary/?term=authorize), 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), Docker (https://docs.docker.com/glossary/?term=Docker)

Create and manage teams

Estimated reading time: 4 minutes

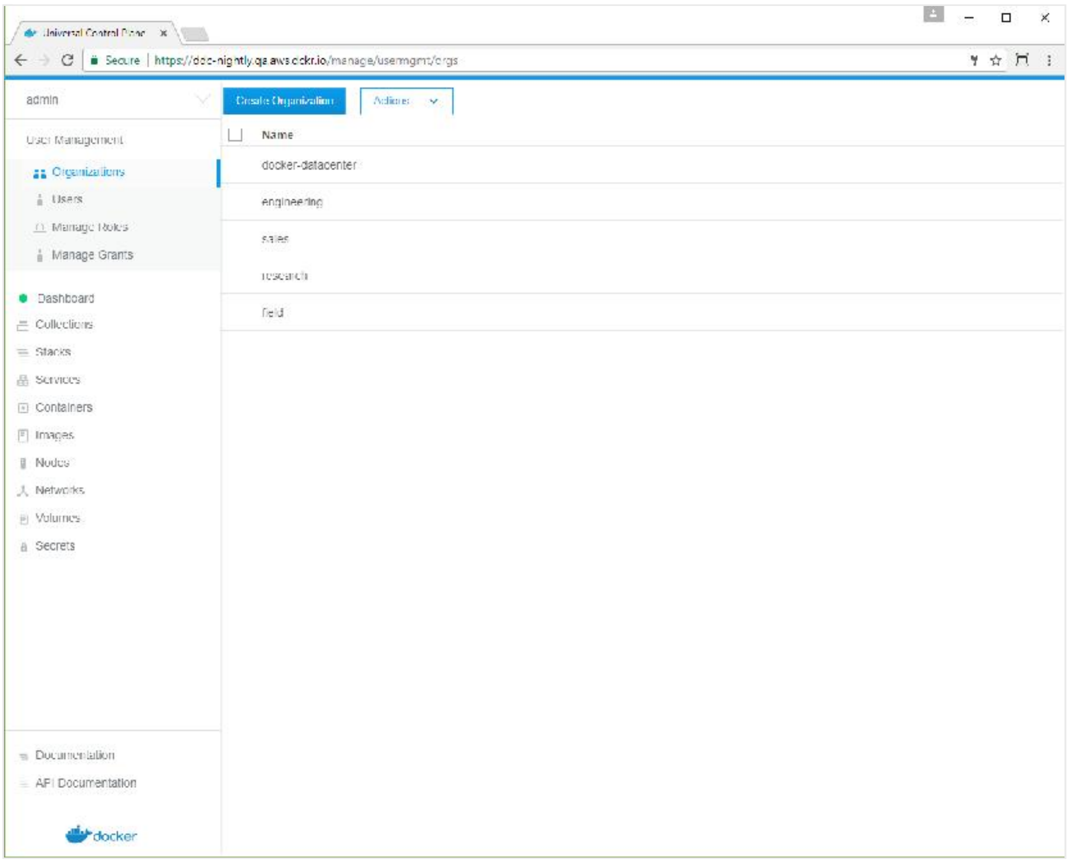
These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

2.2.14

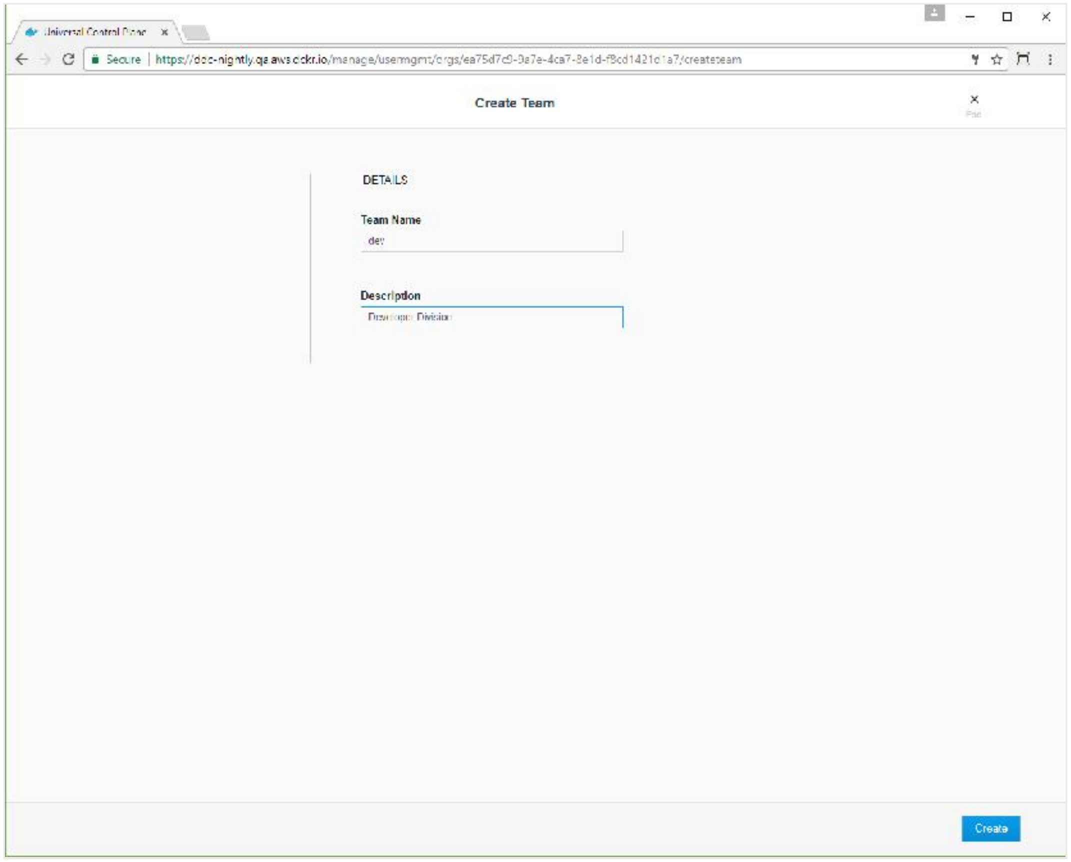
You can extend the user's default permissions by granting them fine-grained permissions over resources. You do this by adding the user to a team.

To create a new team, go to the UCP web UI, and navigate to the Organizations page.



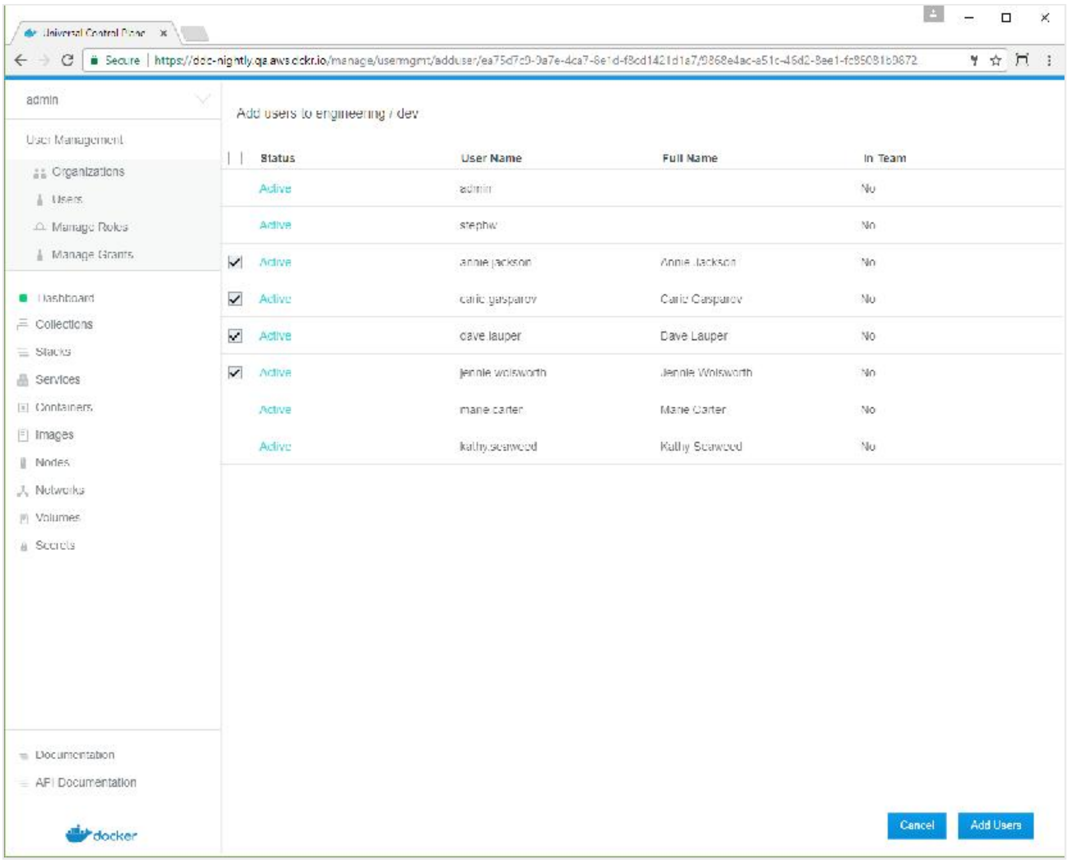
If you want to put the team in a new organization, click Create Organization and give the new organization a name, like “engineering”. Click Create to create it.

In the list, click the organization where you want to create the new team. Name the team, give it an optional description, and click Create to create a new team.


Universal Control Plane
Secure | https://doc-nightly.qa.aws.dckr.io/manage/usermgmt/orgs/ea75d7c9-0a7e-4ca7-8e1d-f8cd1421c1a7/createteam
Create Team
DETAILS
Team Name
dev
Description
Developer Division
Create

Add users to a team

You can now add and remove users from the team. In the current organization’s teams list, click the new team, and in the details pane, click Add Users. Choose the users that you want to add to the team, and when you’re done, click Add Users.



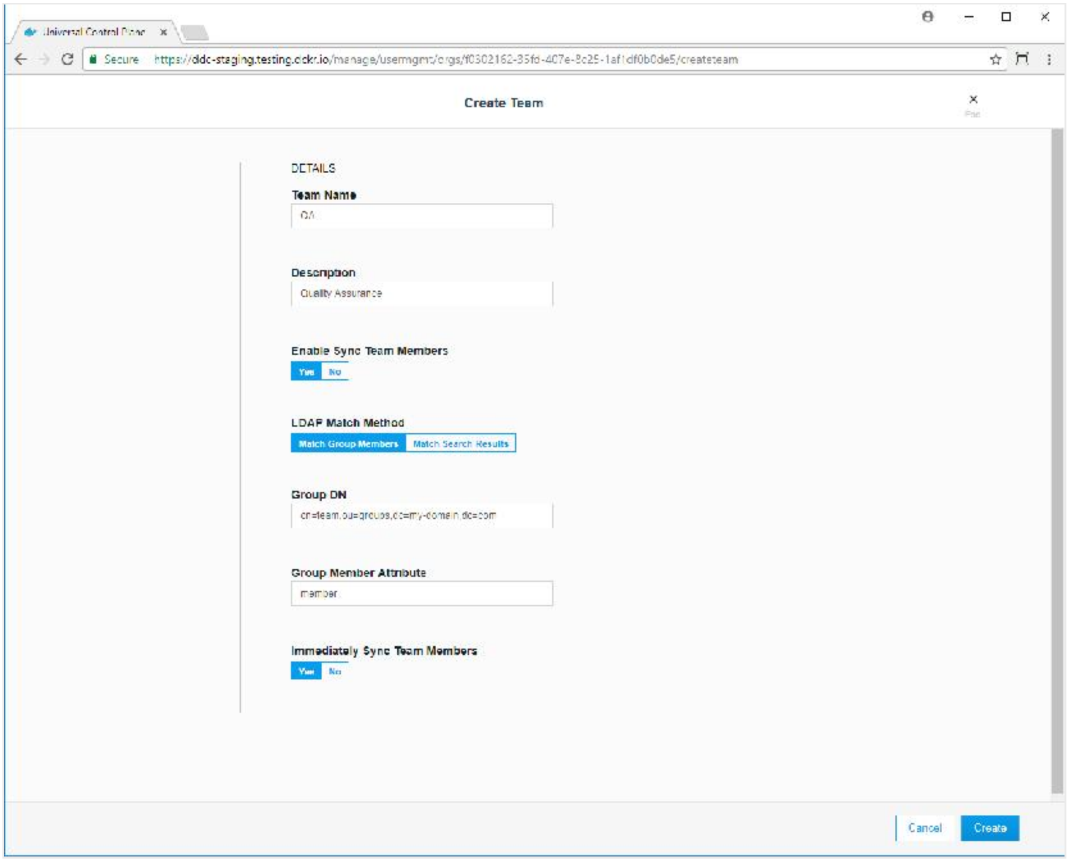
Add users to engineering / dev

	Status	User Name	Full Name	In Team
	Active	admin		No
	Active	stephw		No
✓	Active	annie jackson	Annie Jackson	No
✓	Active	carlo gasparov	Carlo Gasparov	No
✓	Active	dave lauper	Dave Lauper	No
✓	Active	jennie wolsworth	Jennie Wolsworth	No
	Active	marie carter	Marie Carter	No
	Active	kathy.seaweed	Kathy Seaweed	No

Enable Sync Team Members

To sync the team with your organization’s LDAP directory, click Yes.

If UCP is configured to sync users with your organization’s LDAP directory server, you have the option to enable syncing the new team’s members when creating a new team or when modifying settings of an existing team. Learn how to configure integration with an LDAP directory (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/external-auth/). Enabling this option expands the form with additional fields for configuring the sync of team members.


https://ddc-staging.testing.dckr.io/manage/usermgmt/orgs/f0302162-35fd-407e-8c25-1af1df0b0de5/createteam

Create Team

DETAILS

Team Name: QA

Description: Quality Assurance

Enable Sync Team Members: [✓ Yes] No

LDAP Match Method: [✓ Match Group Members] Match Search Results

Group DN: cn=team,ou=groups,dc=my-domain,dc=com

Group Member Attribute: member

Immediately Sync Team Members: [✓ Yes] No

There are two methods for matching group members from an LDAP directory:

Match Group Members

This option specifies that team members should be synced directly with members of a group in your organization's LDAP directory. The team's membership will by synced to match the membership of the group.

Field	Description
Group DN	This specifies the distinguished name of the group from which to select users.
Group Member Attribute	The value of this group attribute corresponds to the distinguished names of the members of the group.

Match Search Results

This option specifies that team members should be synced using a search query against your organization's LDAP directory. The team's membership will be synced to match the users in the search results.

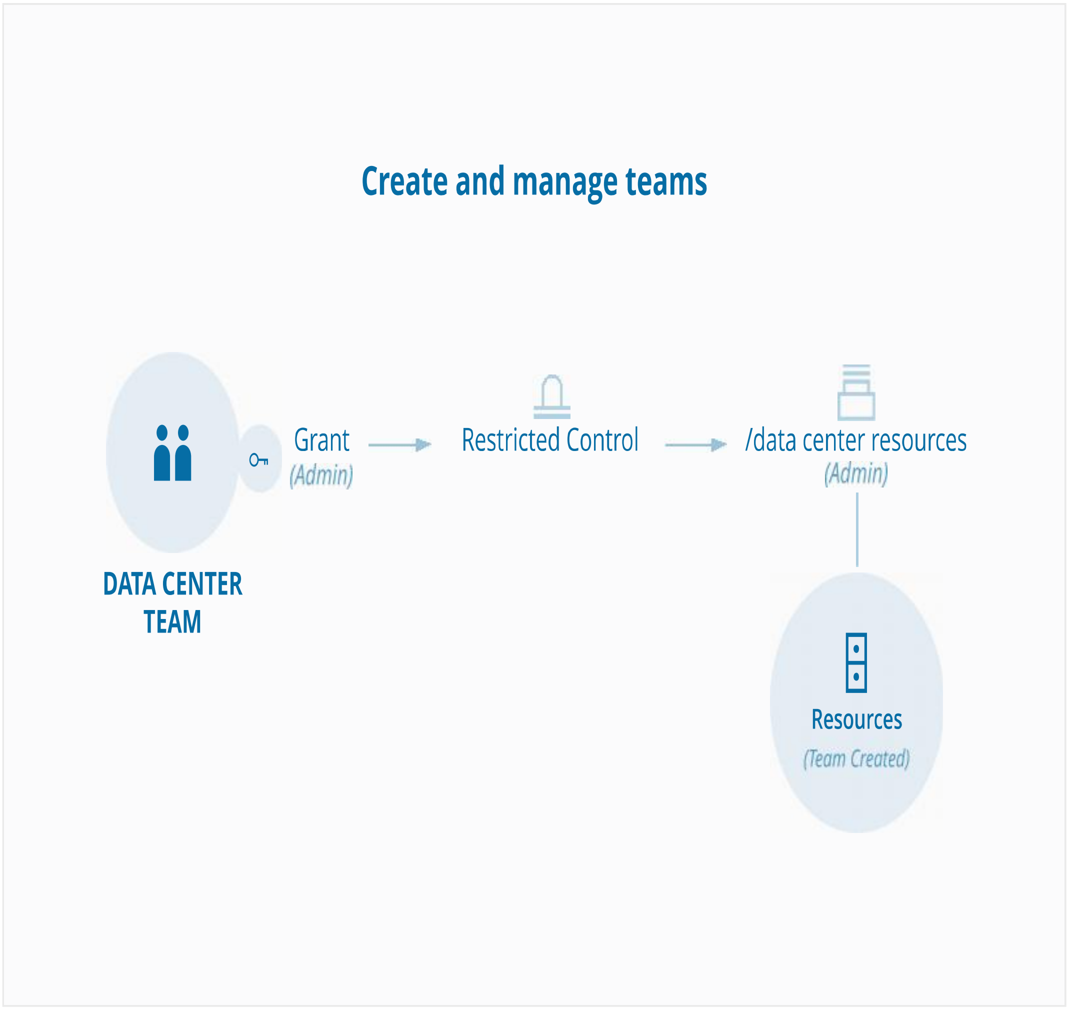
Field	Description
Search Base DN	The distinguished name of the node in the directory tree where the search should start looking for users.
Search Filter	The LDAP search filter used to find users. If you leave this field empty, all existing users in the search scope will be added as members of the team.
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.

Immediately Sync Team Members

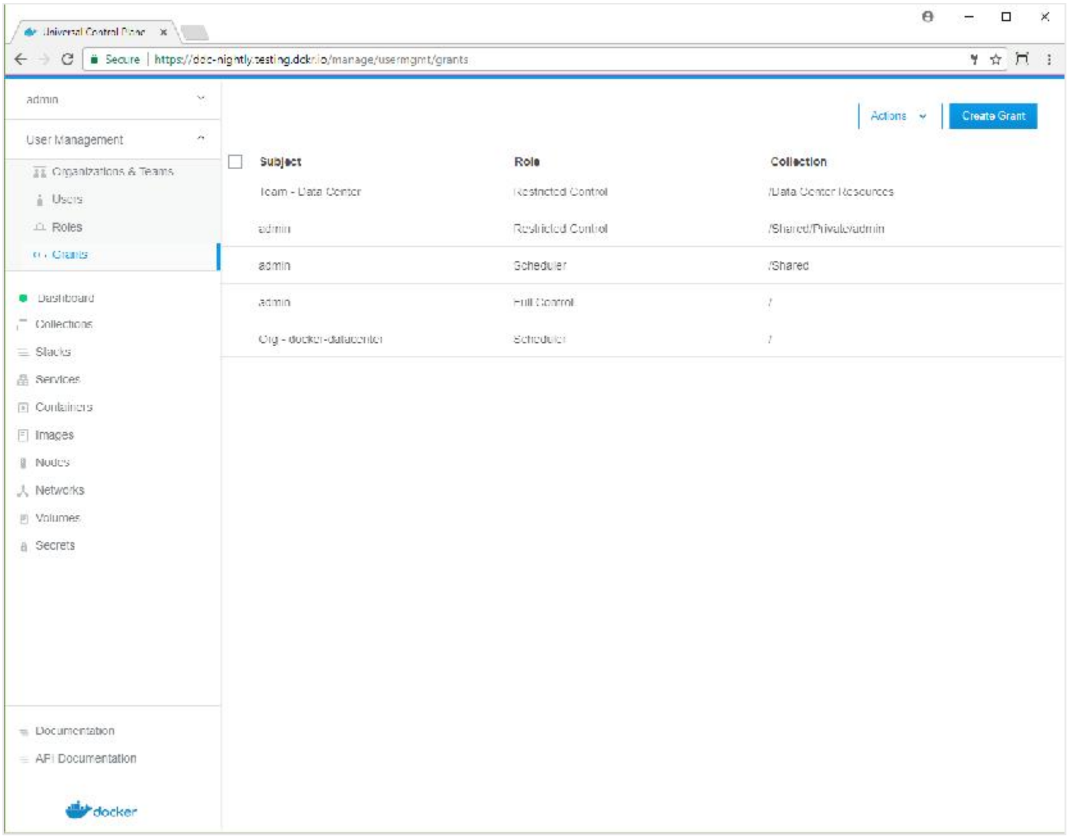
Select this option to run an LDAP sync operation immediately after saving the configuration for the team. It may take a moment before the members of the team are fully synced.

Manage team permissions

Create a grant to manage the team's permissions. Learn how to grant permissions to users based on roles (https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/grant-permissions/). In this example, you create a collection for the "Data Center" team, where they can deploy services and resources, and you create a grant that gives the team permission to access the collection.



1. Navigate to the Organizations & Teams page.
2. Select docker-datacenter, and click Create Team. Name the team “Data Center”, and click Create.
3. Navigate to the Collections page.
4. Click Create Collection, name the collection “Data Center Resources”, and click Create.
5. Navigate to the Grants page, and click Create Grant.
6. Find Swarm in the collections list, and click View Children.
7. Find Data Center Resources, and click Select Collection.
8. In the left pane, click Roles and in the Role dropdown, select Restricted Control.
9. In the left pane, click Subjects and select the Organizations subject type.
10. In the Organization dropdown, select docker-datacenter, and in the Teams dropdown, select Data Center.
11. Click Create to create the grant.



Subject	Role	Collection
Team - Data Center	Restricted Control	/Data Center Resources
admin	Restricted Control	/Shared/Private/admin
admin	Scheduler	/Shared
admin	Full Control	/
Org - docker-datacenter	Scheduler	/

In this example, you gave members of the `Data Center` team `Restricted Control` permissions to create and edit resources in the `Data Center Resources` collection.

Where to go next

- UCP permission levels (https://docs.docker.com/datacenter/ucp/2.2/guides/access-control/permission-levels/)
- 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/)

authorize (https://docs.docker.com/glossary/?term=authorize), authentication (https://docs.docker.com/glossary/?term=authentication), users (https://docs.docker.com/glossary/?term=users), teams (https://docs.docker.com/glossary/?term=teams), groups (https://docs.docker.com/glossary/?term=groups), sync (https://docs.docker.com/glossary/?term=sync), UCP

(https://docs.docker.com/glossary/?term=UCP), Docker
(https://docs.docker.com/glossary/?term=Docker)

Configure and troubleshoot the Docker daemon

Estimated reading time: 11 minutes

After successfully installing and starting Docker, the `dockerd` daemon runs with its default configuration. This topic shows how to customize the configuration, start the daemon manually, and troubleshoot and debug the daemon if you run into issues.

Start the daemon using operating system utilities

On a typical installation the Docker daemon is started by a system utility, not manually by a user. This makes it easier to automatically start Docker when the machine reboots.

The command to start Docker depends on your operating system. Check the correct page under Install Docker (https://docs.docker.com/install/). To configure Docker to start automatically at system boot, see Configure Docker to start on boot (https://docs.docker.com/install/linux/linux-postinstall/#configure-docker-to-start-on-boot).

Start the daemon manually

If you don't want to use a system utility to manage the Docker daemon, or just want to test things out, you can manually run it using the `dockerd` command. You may need to use `sudo` , depending on your operating system configuration.

When you start Docker this way, it runs in the foreground and sends its logs directly to your terminal.

```
$ dockerd

INFO[0000] +job init_networkdriver()
INFO[0000] +job serveapi(unix:///var/run/docker.sock)
INFO[0000] Listening for HTTP on unix (/var/run/docker.sock)
```

To stop Docker when you have started it manually, issue a `Ctrl+C` in your terminal.

Configure the Docker daemon

There are two ways to configure the Docker daemon:

- Use a JSON configuration file. This is the preferred option, since it keeps all configurations in a single place.
- Use flags when starting `dockerd` .

You can use both of these options together as long as you don't specify the same option both as a flag and in the JSON file. If that happens, the Docker daemon won't start and prints an error message.

To configure the Docker daemon using a JSON file, create a file at `/etc/docker/daemon.json` on Linux systems, or `C:\ProgramData\docker\config\daemon.json` on Windows.

Here's what the configuration file looks like:

```
{
  "debug": true,
  "tls": true,
  "tlscert": "/var/docker/server.pem",
  "tlskey": "/var/docker/serverkey.pem",
  "hosts": ["tcp://192.168.59.3:2376"]
}
```

With this configuration the Docker daemon runs in debug mode, uses TLS, and listens for traffic routed to `192.168.59.3` on port `2376` . You can learn what configuration options are available in the dockerd reference docs (https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-configuration-file)

You can also start the Docker daemon manually and configure it using flags. This can be useful for troubleshooting problems.

Here's an example of how to manually start the Docker daemon, using the same configurations as above:

```
dockerd --debug \
  --tls=true \
  --tlscert=/var/docker/server.pem \
  --tlskey=/var/docker/serverkey.pem \
  --host tcp://192.168.59.3:2376
```

You can learn what configuration options are available in the dockerd reference docs (https://docs.docker.com/engine/reference/commandline/dockerd/), or by running:

```
dockerd --help
```

Many specific configuration options are discussed throughout the Docker documentation. Some places to go next include:

- Automatically start containers (https://docs.docker.com/engine/admin/host_integration/)
- Limit a container's resources (https://docs.docker.com/engine/admin/resource_constraints/)
- Configure storage drivers (https://docs.docker.com/engine/userguide/storagedriver/)
- Container security (https://docs.docker.com/engine/security/)

Docker daemon directory

The Docker daemon persists all data in a single directory. This tracks everything related to Docker, including containers, images, volumes, service definition, and secrets.

By default this directory is:

- `/var/lib/docker` on Linux.
- `C:\ProgramData\docker` on Windows.

You can configure the Docker daemon to use a different directory, using the `data-root` configuration option.

Since the state of a Docker daemon is kept on this directory, make sure you use a dedicated directory for each daemon. If two daemons share the same directory, for example, an NFS share, you are going to experience errors that are difficult to troubleshoot.

Troubleshoot the daemon

You can enable debugging on the daemon to learn about the runtime activity of the daemon and to aid in troubleshooting. If the daemon is completely non-responsive, you can also force a full stack trace (/config/daemon/#force-a-full-stack-trace-to-be-logged) of all threads to be added to the daemon log by sending the `SIGUSR` signal to the Docker daemon.

Troubleshoot conflicts between the `daemon.json` and startup scripts

If you use a `daemon.json` file and also pass options to the `dockerd` command manually or using start-up scripts, and these options conflict, Docker fails to start with an error such as:

```
unable to configure the Docker daemon with file /etc/docker/daemon.json:
the following directives are specified both as a flag and in the configuration
file: hosts: (from flag: [unix:///var/run/docker.sock], from file: [tcp://127.0.0.1:2376])
```

If you see an error similar to this one and you are starting the daemon manually with flags, you may need to adjust your flags or the `daemon.json` to remove the conflict.

> Note: If you see this specific error, continue to the next section (/config/daemon/#use-the-hosts-key-in-daemon-json-with-systemd) for a workaround.

If you are starting Docker using your operating system's init scripts, you may need to override the defaults in these scripts in ways that are specific to the operating system.

USE THE HOSTS KEY IN DAEMON.JSON WITH SYSTEMD

One notable example of a configuration conflict that is difficult to troubleshoot is when you want to specify a different daemon address from the default. Docker listens on a socket by default. On Debian and Ubuntu systems using `systemd`, this means that a host flag `-H` is always used when starting `dockerd`. If you specify a `hosts` entry in the `daemon.json`, this causes a configuration conflict (as in the above message) and Docker fails to start.

To work around this problem, create a new file `/etc/systemd/system/docker.service.d/docker.conf` with the following contents, to remove the `-H` argument that is used when starting the daemon by default.

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd
```

There are other times when you might need to configure `systemd` with Docker, such as configuring a HTTP or HTTPS proxy (https://docs.docker.com/engine/admin/systemd/#httphttps-proxy).

> Note: If you override this option and then do not specify a `hosts` entry in the `daemon.json` or a `-H` flag when starting Docker manually, Docker fails to start.

Run `sudo systemctl daemon-reload` before attempting to start Docker. If Docker starts successfully, it is now listening on the IP address specified in the `hosts` key of the `daemon.json` instead of a socket.

> ❗ Important: Setting `hosts` in the `daemon.json` is not supported on Docker for Windows or Docker for Mac.

Out Of Memory Exceptions (OOME)

If your containers attempt to use more memory than the system has available, you may experience an Out Of Memory Exception (OOME) and a container, or the Docker daemon, might be killed by the kernel OOM killer. To prevent this from happening, ensure that your application runs on hosts with adequate memory and see Understand the risks of running out of memory (https://docs.docker.com/engine/admin/resource_constraints/#understand-the-risks-of-running-out-of-memory).

Read the logs

The daemon logs may help you diagnose problems. The logs may be saved in one of a few locations, depending on the operating system configuration and the logging subsystem used:

Operating system	Location
RHEL, Oracle Linux	`/var/log/messages`
Debian	`/var/log/daemon.log`
Ubuntu 16.04+, CentOS	Use the command `journalctl -u docker.service`
Ubuntu 14.10-	`/var/log/upstart/docker.log`
macOS (Docker 18.01+)	`~/Library/Containers/com.docker.docker/Data/vms/0/console-ring`
macOS (Docker <18.01)	`~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/console-ring`
Windows	`AppData\Local`

Enable debugging

There are two ways to enable debugging. The recommended approach is to set the `debug` key to `true` in the `daemon.json` file. This method works for every Docker platform.

1. Edit the `daemon.json` file, which is usually located in `/etc/docker/`. You may need to create this file, if it does not yet exist. On macOS or Windows, do not edit the file directly. Instead, go to Preferences / Daemon / Advanced.
2. If the file is empty, add the following:

   ```
   {
     "debug": true
   }
   ```

   If the file already contains JSON, just add the key `"debug": true`, being careful to add a comma to the end of the line if it is not the last line before the closing bracket. Also verify that if the `log-level` key is set, it is set to either `info` or `debug`. `info` is the default, and possible values are `debug`, `info`, `warn`, `error`, `fatal`.

3. Send a `HUP` signal to the daemon to cause it to reload its configuration. On Linux hosts, use the following command.

   ```
   $ sudo kill -SIGHUP $(pidof dockerd)
   ```

   On Windows hosts, restart Docker.

Instead of following this procedure, you can also stop the Docker daemon and restart it manually with the debug flag `-D`. However, this may result in Docker restarting with a different environment than the one the hosts' startup scripts create, and this may make debugging more difficult.

Force a stack trace to be logged

If the daemon is unresponsive, you can force a full stack trace to be logged by sending a `SIGUSR1` signal to the daemon.

- Linux:

  ```
  $ sudo kill -SIGUSR1 $(pidof dockerd)
  ```

- Windows Server:

  Download docker-signal (https://github.com/jhowardmsft/docker-signal).

  Run the executable with the flag `--pid=<PID of daemon>`.

This forces a stack trace to be logged but does not stop the daemon. Daemon logs show the stack trace or the path to a file containing the stack trace if it was logged to a file.

The daemon continues operating after handling the `SIGUSR1` signal and dumping the stack traces to the log. The stack traces can be used to determine the state of all goroutines and threads within the daemon.

View stack traces

The Docker daemon log can be viewed by using one of the following methods:

- By running `journalctl -u docker.service` on Linux systems using `systemctl`
- `/var/log/messages`, `/var/log/daemon.log`, or `/var/log/docker.log` on older Linux systems
- By running `Get-EventLog -LogName Application -Source Docker -After (Get-Date).AddMinutes(-5) | Sort-Object Time | Export-CSV ~/last5minutes.CSV` on Docker EE for Windows Server

> Note: It is not possible to manually generate a stack trace on Docker for Mac or Docker for Windows. However, you can click the Docker taskbar icon and choose Diagnose and feedback to send information to Docker if you run into issues.

Look in the Docker logs for a message like the following:

```
...goroutine stacks written to /var/run/docker/goroutine-stacks-2017-06-02T193336z.log
...daemon datastructure dump written to /var/run/docker/daemon-data-2017-06-02T193336z.log
```

The locations where Docker saves these stack traces and dumps depends on your operating system and configuration. You can sometimes get useful diagnostic information straight from the stack traces and dumps. Otherwise, you can provide this information to Docker for help diagnosing the problem.

Check whether Docker is running

The operating-system independent way to check whether Docker is running is to ask Docker, using the `docker info` command.

You can also use operating system utilities, such as `sudo systemctl is-active docker` or `sudo status docker` or `sudo service docker status`, or checking the service status using Windows utilities.

Finally, you can check in the process list for the `dockerd` process, using commands like `ps` or `top`.

docker (https://docs.docker.com/glossary/?term=docker), daemon (https://docs.docker.com/glossary/?term=daemon), configuration (https://docs.docker.com/glossary/?term=configuration), troubleshooting (https://docs.docker.com/glossary/?term=troubleshooting)

UCP System requirements

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 ▾

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

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

When planning for host storage, workflows based around `docker pull` through UCP will result in higher storage requirements on manager nodes, since `docker pull` through UCP results in the image being pulled on all nodes.

Also, make sure the nodes are running an operating system support by Docker EE (https://success.docker.com/Policies/Compatibility_Matrix).

For highly-available installations, you also need a way to transfer files between hosts.

> Workloads on manager nodes
>
> These requirements assume that manager nodes don't run regular workloads. If you plan to run additional workloads on manager nodes, you may need to provision more powerful nodes. If manager nodes become overloaded, the swarm may experience issues.

Ports used

When installing UCP on a host, make sure the following ports are open:

Hosts	Direction	Port	Purpose
managers, workers	in	TCP 443 (configurable)	Port for the UCP web UI and API
managers	in	TCP 2376 (configurable)	Port for the Docker Swarm manager. Used for backwards compatibility
managers, workers	in	TCP 2377 (configurable)	Port for communication between swarm nodes
workers	out	TCP 2377 (configurable)	Port for communication between swarm nodes
managers, workers	in, out	UDP 4789	Port for overlay networking
managers, workers	in, out	TCP, UDP 7946	Port for gossip-based clustering

Hosts	Direction	Port	Purpose
managers, workers	in	TCP 12376	Port for a TLS proxy that provides access to UCP, Docker Engine, and Docker Swarm
managers	in	TCP 12379	Port for internal node configuration, cluster configuration, and HA
managers	in	TCP 12380	Port for internal node configuration, cluster configuration, and HA
managers	in	TCP 12381	Port for the certificate authority
managers	in	TCP 12382	Port for the UCP certificate authority
managers	in	TCP 12383	Port for the authentication storage backend
managers	in	TCP 12384	Port for the authentication storage backend for replication across managers
managers	in	TCP 12385	Port for the authentication service API
managers	in	TCP 12386	Port for the authentication worker
managers	in	TCP 12387	Port for the metrics service

For overlay networks with encryption to work, you need to ensure that IP protocol 50 (ESP) traffic is allowed.

Also, make sure the networks you’re using allow the UCP components enough time to communicate before they time out.

Component	Timeout (ms)	Configurable

Component	Timeout (ms)	Configurable
Raft consensus between manager nodes	3000	no
Gossip protocol for overlay networking	5000	no
etcd	500	yes
RethinkDB	10000	no
Stand-alone swarm	90000	no

Time Synchronization

In distributed systems like Docker UCP, time synchronization is critical to ensure proper operation. As a best practice to ensure consistency between the engines in a UCP swarm, all engines should regularly synchronize time with a Network Time Protocol (NTP) server. If a server's clock is skewed, unexpected behavior may cause poor performance or even failures.

Compatibility and maintenance lifecycle

Docker EE is a software subscription that includes three products:

- Docker Engine with enterprise-grade support,
- Docker Trusted Registry,
- Docker Universal Control Plane.

Learn more about compatibility and the maintenance lifecycle for these products:

- Compatibility Matrix (https://success.docker.com/Policies/Compatibility_Matrix)
- Maintenance Lifecycle (https://success.docker.com/Policies/Maintenance_Lifecycle)

Version compatibility

UCP 2.2 requires minimum versions of the following Docker components:

- Docker Engine 17.06 or higher
- DTR 2.3 or higher

Where to go next

- UCP architecture (https://docs.docker.com/datacenter/ucp/2.2/guides/architecture/)
- Plan your installation (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/plan-installation/)

UCP (https://docs.docker.com/glossary/?term=UCP), architecture (https://docs.docker.com/glossary/?term=architecture), requirements (https://docs.docker.com/glossary/?term=requirements), Docker EE (https://docs.docker.com/glossary/?term=Docker EE)

Authored by: Justin Nevill (/author/justinnevill)

1 0 Share PDF (https://success.docker.com/api/articles/what-are-the-minimum-requirements-for-docker-trusted-registry-dtr/pdf)

What are the minimum requirements for Docker Trusted Registry (DTR)?

Article ID: KB000471

EE

Questions

- What are the minimum requirements for Docker Trusted Registry (DTR)?
- Why aren't the minimum requirements for DTR published?

Answer

The minimum requirements for DTR aren't documented anywhere because those requirements are highly dependent on how much the installation is being used. DTR is made up of small binaries that don't do a whole lot other than serve up large files. The biggest bottlenecks will be introduced by how well your hardware can handle that task.

RAM: Generally 4GB is considered sufficient for smaller production installations.

DISK (DTR itself, not image storage): The current version of the DTR software itself takes up about 1.8GB. Generally it's preferable to have no less than 20-30GB of space available. This allows plenty of breathing room for upgrades, or running other software in Docker alongside DTR.

DISK (image storage): The requirements for the storage of Docker container images used in conjunction with DTR will scale linearly with the size and number of images, as well as how much I/O will be conducted. There are several available storage backends to choose from: The local storage backend is the default, and stores docker images on your DTR server's local filesystem. You could introduce high performance drives or a SAN to meet the I/O demand of your installation with this driver. With the local filesystem driver, a significant amount of RAM is recommended in order to maximize the benefits of Linux' native storage caching. Other available storage drivers include S3, Azure, and Swift (among others). Using an external storage mechanism will lessen the I/O requirements for the DTR server itself, but that external service will of course then need to be able to meet the I/O demands of the DTR installation.

CPU: DTR is not very CPU intensive. Most modern dual core processors are likely to be sufficient.

What is Docker (https://www.docker.com/what-docker)

What is a Container (https://www.docker.com/what-container)

Use Cases (https://www.docker.com/use-cases)

Customers (https://www.docker.com/customers)

For Government (https://www.docker.com/industry-government)

For IT Pros (https://www.docker.com/itpro)

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)

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

Pricing (https://www.docker.com/pricing)

Community Edition (https://www.docker.com/community-edition)

Enterprise Edition (https://www.docker.com/enterprise-edition)

Docker Datacenter (https://www.docker.com/enterprise-edition#container_management)

Docker Cloud (https://cloud.docker.com/)

Docker Store (https://store.docker.com/)

Get Docker (https://www.docker.com/get-docker)

Docker for Mac (https://www.docker.com/docker-mac)

Docker for Windows(PC) (https://www.docker.com/docker-windows)

Docker for AWS (https://www.docker.com/docker-aws)

Docker for Azure (https://www.docker.com/docker-azure)

Docker for Windows Server (https://www.docker.com/docker-windows-server)

Docker for Debian (https://www.docker.com/docker-debian)

Docker for Fedora® (https://www.docker.com/docker-fedora)

Docker for Oracle Linux (https://www.docker.com/docker-oracle-linux)

Docker for RHEL (https://www.docker.com/docker-red-hat-enterprise-linux-rhel)

Docker for SLES (https://www.docker.com/docker-suse-linux-enterprise-server-sles)

Docker for Ubuntu (https://www.docker.com/docker-ubuntu)

Documentation (https://docs.docker.com/)

Blog (https://blog.docker.com/)

RSS Feed (https://blog.docker.com/feed/)

Training (https://training.docker.com/)

Knowledge Base (https://success.docker.com/kbase)

Resources (https://www.docker.com/products/resources)

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

Open Source (https://www.docker.com/technologies/overview)

Events (https://www.docker.com/community/events)

Forums (https://forums.docker.com/)

Docker Captains (https://www.docker.com/community/docker-captains)

Scholarships (https://www.docker.com/community-partnerships)

Community News (https://blog.docker.com/curated/)

Status (http://status.docker.com/) Security (https://www.docker.com/docker-security)

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

(http: (http: (https (https (https (https (http: (http:

Authored by: Anoop Kumar (/author/anoop.kumar) and Alex Drahon (/author/alex.drahon)

32 0 Share PDF (https://success.docker.com/api/articles/docker-ee-best-practices-17-03/pdf)

Docker Reference Architecture: Docker EE Best Practices and Design Considerations (EE 17.03)

EE LINUX WINDOWS SERVER DEPLOY EE-17.03.0-EE-1 UCP 2.1.0 DTR-2.2.1 CSE-1.13.1-CS1

Introduction

Docker Enterprise Edition (Docker EE) is the enterprise container platform from Docker to be used across the entire software supply chain. It is a fully-integrated solution for container-based application development, deployment, and management. With integrated end-to-end security, Docker EE enables application portability by abstracting your infrastructure so that applications can move seamlessly from development to production.

What You Will Learn

This reference architecture describes a standard, production-grade, Docker EE deployment. It also details the components of Docker EE, how they work, how to automate deployments, how to manage users and teams, how to provide high availability for the platform, and how to manage the infrastructure.

Some environment-specific configuration details are not be provided. For instance, load balancers vary greatly between cloud platforms and on-premises infrastructure platform. For these types of components, general guidelines to environment-specific resources are provided.

Understanding Docker Components

From development to production, Docker EE provides a seamless platform for containerized applications both on-premises and on the Cloud. Docker EE is available in three tiers to meet different application needs. Both Docker EE Standard (formerly known as Docker Datacenter) and Docker EE Advanced include the following components:

- Docker CS Engine, the commercially supported Docker Engine
- Universal Control Plane (UCP), the web-based, unified cluster and application management solution
- Docker Trusted Registry (DTR), a resilient and secure image management repository

Together they provide an integrated solution with the following design goals:

- Agility — the Docker API is used to interface with the platform so that operational features do not slow down application delivery
- Portability — the platform abstracts details of the infrastructure for applications
- Control — the environment is secure by default, provides robust access control, and logging of all operations

To achieve these goals the platform must be resilient and highly available. This reference architecture demonstrates this robust configuration.

Docker CS Engine

The building block of Docker EE, the Docker Commercially Supported Engine (CS Engine) is responsible for container-level operations, interaction with the OS, providing the Docker API, and running the Swarm cluster. The Engine is also the integration point for infrastructure, including the OS resources, networking, and storage.

Universal Control Plane

UCP extends CS Engine by providing an integrated application management platform. It is both the main interaction point for users and the integration point for applications. UCP runs an agent on all nodes in the cluster to monitor them and a set of services on the *controller nodes*. This includes *identity services* to manage users, *Certificate Authorities* (CA) for user and cluster PKI, the main *controller* providing the Web UI and API, data stores for UCP state, and a *Classic Swarm* service for backward compatibility.

Docker Trusted Registry

The DTR is an application managed by, and integrated with UCP, that provides Docker images distribution and security services. The DTR uses UCP's identity services to provide Single Sign-On (SSO), and establish a mutual trust to integrate with its PKI. It runs as a set of services on one or several *replicas*: the *registry* to store and distribute images, an image signing service, a Web UI, an API, and data stores for image metadata and DTR state.

Swarm Mode

To provide a seamless cluster based on a number of nodes, DDC relies on CS Engine's *swarm mode* (https://docs.docker.com/engine/swarm/key-concepts/) capability. Swarm mode divides nodes between *workers*, nodes running application workloads defined as services, and *managers*, nodes in charge of maintaining desired state, managing the cluster's internal PKI, and providing an API. Managers can also run workloads. In a Docker EE environment they run UCP controllers and shouldn't run anything else.

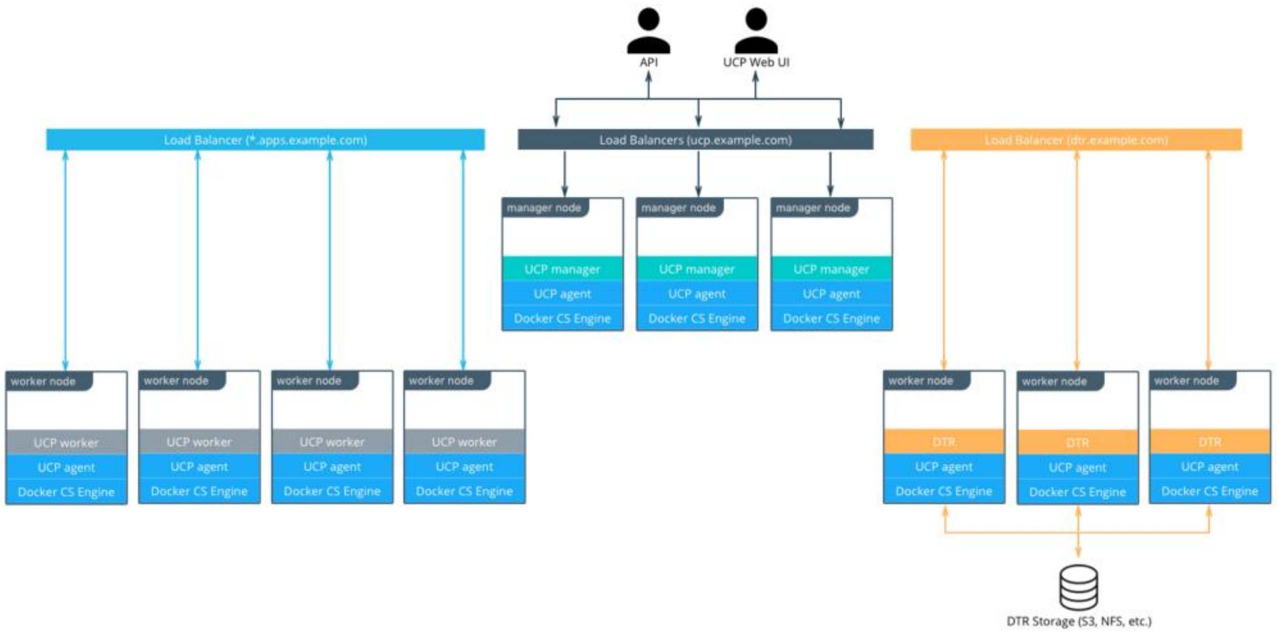
The Swarm mode service model provides a declarative desired state for workloads, scalable to a number of *tasks* (the service's containers), accessible through a stable resolvable name, and optionally exposing an end-point. Exposed services are accessible from any node on a cluster-wide reserved port, reaching tasks through the *routing mesh*, a fast routing layer using high-performance switching in the Linux kernel. This set of features enable internal and external discoverability for services, UCP's *HTTP Routing Mesh* (HRM) adding hostname-to-service mapping.

A Standard Deployment Architecture

This section demonstrates a standard, production-level architecture for Docker EE using 10 nodes: 3 UCP controllers, 3 nodes for the DTR, and 4 worker nodes for application workloads. The number of worker nodes is arbitrary, most environments will have more depending on applications needs, it doesn't change the architecture or the cluster configuration.

Access to the environment is done through 3 Load Balancers (or 3 load balancer virtual hosts) with corresponding DNS entries for the UCP controllers, the DTR replicas, and the applications running in the cluster.

The DTR replicas use shared storage for images. S3-compatible object storage (the default) and NFS storage are both covered in this section.



Node Size

A node is a machine in the cluster (virtual or physical) with Docker Engine running on it. When provisioning each node, assign it a role: UCP Controller, DTR, or worker node so that they are protected from running application workloads.

To decide what size the node should be in terms of CPU, RAM, and storage resources, consider the following:

1. All nodes should at least fulfill the minimal requirements for UCP 2.0: 2 GB of RAM and 3 GB of storage. More detailed requirements are in the UCP documentation (https://docs.docker.com/datacenter/ucp/2.0/guides/installation/system-requirements/).
2. UCP Controller nodes should be provided with more than the minimal requirements but won't need much more if nothing else runs on them.
3. Ideal Worker nodes size will vary based on your workloads, so it is impossible to define a universal standard size.
4. Other considerations like target density (average number of container per node), whether one standard node type or several are preferred, and other operational considerations might also influence sizing.

If possible, node size should be determined by experimentation and testing actual workloads, and they should be refined iteratively. A good starting point is to select a standard or default machine type in your environment and use this size only. If your standard machine type provides more resources than the UCP Controllers need, it makes sense to have a smaller node size for these. Whatever the starting choice, it’s important to monitoring resource usage and cost to improve the model.

Two example scenarios:

- One standard node size: 2 VCPU, 8 GB RAM, 20 GB storage
- Two node sizes: 2 VCPU, 8 GB RAM, 20 GB storage for UCP Controllers and 4 VCPU, 64 GB RAM, 40 GB storage for worker nodes

Depending on your OS of choice, storage configuration for Docker Engine might require some planning. Refer to the Support Matrix (https://success.docker.com/Policies/Compatibility_Matrix) to see what storage drivers (https://docs.docker.com/engine/userguide/storagedriver/selectadriver/) are supported for your host OS. This is especially important if you are using RHEL or CentOS, which use device mapper with direct-lvm (https://docs.docker.com/engine/userguide/storagedriver/device-mapper-driver/#/configure-direct-lvm-mode-for-production).

Load Balancers

Load balancers configuration should be done before installation, including the creation of DNS entries. Most load balancers should work with Docker EE. The only requirements are TCP passthrough and the ability to do health checks on an HTTPS endpoint.

In our example architecture, the three UCP controllers ensure UCP resiliency in case of node failure or controller reconfiguration. Access to UCP through the GUI or API is always done using TLS. The load balancer is configured for simple TCP pass-through on port 443, using a custom HTTPS health check at `https://<ucp_controller>/_ping` .

Be sure to create a DNS entry for the UCP host such as `ucp.example.com` and point to the load balancer.

The setup for the three DTR replicas is similar to setting up the UCP controllers. Again, use TCP pass-through to port 443 on the nodes, except the health check is at `https://<dtr_replica_node>/health` .

Create a DNS entry for the DTR host such as `dtr.example.com` and point it to the load balancer. It's important to keep it as concise as possible because it will be part of the full name of images. For example, `user_a` 's `webserver` image will be named `dtr.example.com/user_a/webserver` .

The application load balancer provides access to services HTTP endpoints exposed through UCP's HTTP Routing Mesh (HRM). HRM provides a reverse-proxy to map domain names to services that expose ports and are attached to the `ucp-hrm` overlay network. As an example, the `voting` application exposes the `vote` service's port `80` . The HRM maps `[http://vote.apps.example.com](http://vote.apps.example.com "http://vote.apps.example.com")` to this port on the `ucp-hrm` overlay network, and the application LB itself maps `*.apps.example.com` to nodes in the cluster.

For more details on load balancing for applications on UCP, see Universal Control Plane 2.0 Service Discovery and Load Balancing (https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Universal_Control_Plane_2.0_Service_Discovery_and_Load_Balancing).

DTR Storage

The DTR usually needs to store a large number of images. It uses external storage (S3, NFS, etc.), not node storage so that it can be shared between DTR replicas. The DTR replicates metadata and configuration information between replicas, but not image layers themselves. To determine storage size, start with the size of the existing images used in the environment and evolve from there.

It’s best to use an existing storage solution in your environment so that image storage can benefit from existing operations experience. If you have to chose a new solution, consider using S3-compatible object storage, which maps more closely to registry operations.

Refer to An Introduction to Storage Solutions for Docker CaaS (https://success.docker.com/Architecture/An_Introduction_to_Storage_Solutions_for_Docker_CaaS) for more information about selecting storage solutions.

Recommendations for the Docker EE Installation

This section details the installation process for the architecture and provide a checklist. It is not a substitute for the documentation, which provides more details and is authoritative in any case. The goal is to help you define a repeatable (and ideally automated) process to deploy, configure, upgrade and expand your Docker EE environment.

The three main stages of a Docker EE Standard or Advanced installation are as follows:

1. Deploying and configuring the infrastructure (hosts, network, storage)
2. Installing and configuring the Docker Engine, running as an application on the hosts
3. Installing and configuring UCP and the DTR, delivered as containers running on the Engine

Infrastructure Considerations

The installation documentation (https://docs.docker.com/datacenter/install/linux/) details infrastructure requirements for Docker EE Standard and Advanced. It is recommended to use existing or platform specific tools in your environment to provide standardized and repeatable configuration for infrastructure components.

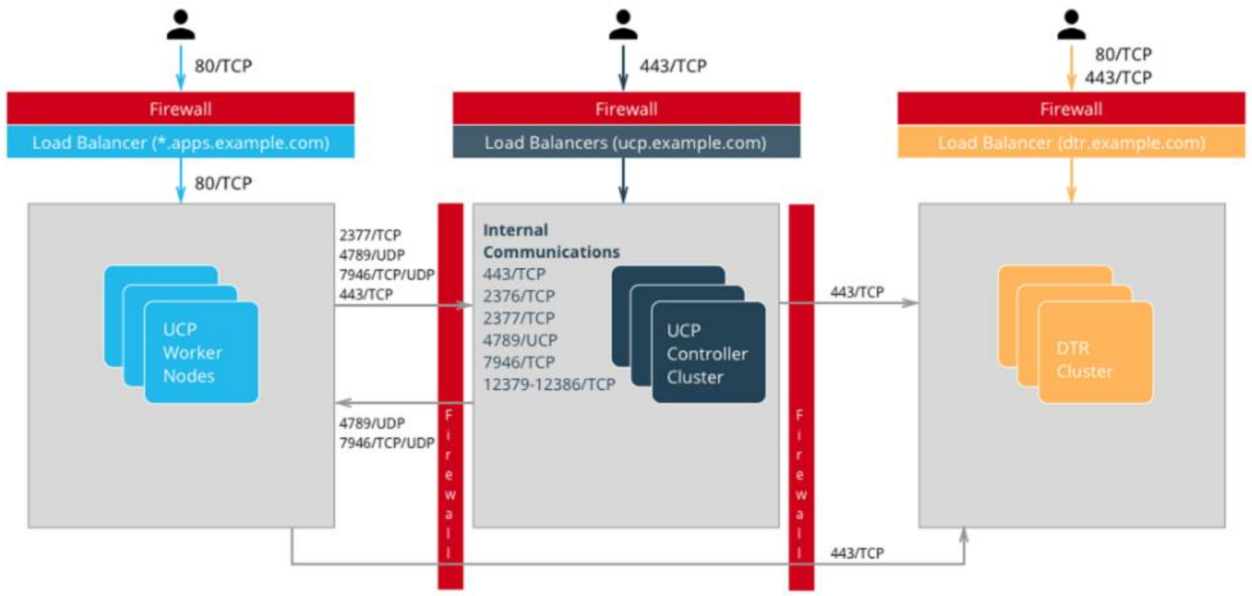
Network

Docker components need to communicate over the network, and the documentation lists the ports that need to be open (https://docs.docker.com/datacenter/ucp/2.0/guides/installation/system-requirements/) for internal cluster communication. Misconfiguration of the cluster's internal network can lead to issues that might be difficult to track down. It's better to start with a relatively simple environment. This reference architecture assumes a single subnet for all hosts and overlay networking for containers.

To get more details and evaluate options, consult Docker Reference Architecture: Designing Scalable, Portable Docker Container Networks (https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Designing_Scalable%2C_Portable_Docker_Container_Networks).

Firewall

Access to Docker EE is done using port 443 only (ports 443 and 80 for DTR), whether it’s to access the Web UI or the remote API. This makes external firewall configuration simple. In most cases you only need to open ports 80, 443, and 22. Port 22 is for SSH access is optional since Docker EE doesn’t require SSH access. Access to applications is through a load balancer using HTTPS. If you expose other TCP services to the outside world, open these ports on the firewall. As explained in the previous section, several ports need to be open for communication inside the cluster. If you have a firewall between some nodes in the cluster, for example, to separate controllers from worker nodes, open the relevant port there too.



Load Balancers

Load balancers are detailed in the previous section. They must be in place before installation and must be provisioned with the right hostnames. External (load balancer) hostnames are used for HA and also for TLS certificates. It’s easier not having to reconfigure them during or after installation.

Refer to the previous section Understanding Docker Components (https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Docker_EE_Best_Practices_and_Design_Considerations#ddc-components) for details about load balancer configuration.

Shared Storage

DTR shared storage, used for images in the registry, must be ready and accessible from the DTR nodes. Test it works using an S3 or NFS command line client to avoid having to debug DTR storage configuration.

Host Configuration

Host configuration varies based on the OS used and existing configuration standards, but there are some important steps that must be followed after OS installation:

1. Clock synchronization using NTP or similar service. Clock skew can produce hard to debug errors, especially where the Raft algorithm is used (UCP and the DTR).
2. Static IPs are required for UCP for all hosts.
3. Hostnames are used for node identification in the cluster. The hostname must be set in an non-ephemeral way.

4. Host firewalls must allow intra-cluster traffic on all the ports specified in the installation docs.
5. Storage must be configured if needed. For example, the `devicemapper` driver need a logical volume configured before Docker Engine installation.

Docker CS Engine Installation Considerations

Detailed installation instructions for CS Engine are provided by the documentation (https://docs.docker.com/cs-engine/1.13/). To install on machines that don't have Internet access, add these package to your own private repository. After installing the package, make sure the `docker` service is configured to start on system boot.

The best way to change parameters for CS Engine is to use the `/etc/docker/daemon.json` configuration file. This ensures that the configuration can be reused across different systems and OS in a consistent way. See the documentation for the full list of options (https://docs.docker.com/engine/reference/commandline/dockerd/#/daemon-configuration-file).

Make sure Engine is configured correctly by starting the `docker` service and verifying the parameters with `docker info` .

UCP Installation Considerations

The UCP installer creates a functional cluster from a set of machines running CS Engine. That includes creating a Swarm cluster and installing the UCP controllers. The default installation mode as described in the Installation Guide (https://docs.docker.com/datacenter/ucp/2.0/guides/installation/#/step-4-install-ucp) is interactive.

To perform fully-automated, repeatable deployments, provide more information to the installer:

```
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
  -v /tmp/docker_subscription.lic:/config/docker_subscription.lic \
  -e UCP_ADMIN_PASSWORD=password --name ucp docker/ucp install --host-address IP_or_in
terface \
  --san manager1.example.com --san ucp.example.com
```

Each of these options are explained in the following sections.

External Certificates

By default UCP uses self-signed TLS certificates. For production deployments, it is recommended to use certificates generated by a trusted CA. In most cases, this is your organization’s internal CA.

The certificates and keys needed are as follows:

1. The root CA's public certificate `ca.pem` .
2. The TLS certificate `cert.pem` . It must also include the public certificates of any intermediate CA and have SANs for all addresses used to access UCP, including the load balancer's hostname (e.g. `ucp.example.com`) and the individual controllers' hostnames (e.g. `ucp-controller1.example.com`) to access them directly.
3. The TLS private key, `key.pem` .

To add these automatically during installation, add these files with the exact names to a volume named `ucp-controller-server-certs` on the machine where you install UCP, and use the `--external-server-cert` install parameter.

It is also possible to add the certificates through the Web UI after installation.

License File

The license file can be provided for installation via the command line as well using a bind-mount (volume) in `/config` . Specify its location with `-v /path/to/docker_subscription.lic:/config/docker_subscription.lic` .

Admin Password

To make the installation entirely non-interactive, the admin password must be passed using the `--admin-password` install parameter. The admin username is `admin` by default. Use `--admin-username` to change it.

The full list of install parameters is provided in the install command documentation (https://docs.docker.com/datacenter/ucp/2.0/reference/cli/install/#/options).

Adding Nodes

Once the installation has finished for the first controller node, the two additional controllers must be installed by joining them to the cluster. UCP configures a full controller replica on the manager node in the cluster, so the only command needed on the two controllers is a `docker swarm join` with the right token. The exact command can be obtained by running `docker swarm join-token manager` on the first controller.

To join the worker nodes, the equivalent command can be obtained with `docker swarm join-token worker` on any controller:

```
docker swarm join-token worker
```

To add a worker to this swarm, run the following command:

```
docker swarm join \
--token SWMTKN-1-00gqkzjo07dxcxb53qs4brml51vm6ca2e8fjnd6dds8lyn9ng1-092vhgjxz3jixvjf08
1sdge3p \
192.168.65.2:2377
```

To make sure everything is running correctly, log into UCP at `[https://ucp.example.com](https://ucp.example.com "https://ucp.example.com")`.

DTR Installation Considerations

Installation of the DTR is similar to that of UCP. Install and configure one node, and then join replicas to form a full, highly-available setup. For installation of the first instance as well as replicas, point the installer to the node in the cluster it will install on.

Certificates and image storage must be configured after installation. Once shared storage is configured, the two replicas can be added with the `join` command.

Validating the Deployment

When installation of everything has finished, tests can be done to validate the deployment. Disable scheduling of workloads on UCP controllers and nodes running the DTR.

Basic tests to consider:

1. Log in through `[https://ucp.example.com](https://ucp.example.com "https://ucp.example.com")` as well as directly to a manager node, eg. `[https://manager1.example.com](https://manager1.example.com "https://manager1.example.com")`. Make sure the cluster and all nodes are healthy.
2. Test that you can deploy an application following the example in the documentation (https://docs.docker.com/datacenter/ucp/2.0/guides/applications/).
3. Test that users can download a bundle and connect to the cluster from the CLI. Test that they can use docker-compose (https://docs.docker.com/compose/overview/).
4. Test DTR with a full image workflow. Make sure storage isn't misconfigured and images are stored in the right place.

Consider building a standard automated test suite to validate new environments and updates. Just testing standard functionality should hit most configuration issues. Make sure you run these tests with a non-admin user, the test user should have similar rights as users of the platform. Measuring time taken by each test can also pinpoint issues with underlying infrastructure configuration. Fully deploying an actual application from your organization should be part of this test suite.

High Availability in Docker EE

In a production environment, it is vital that critical services have minimal downtime. It is important to understand how high availability (HA) is achieved in UCP and the DTR, and what to do to when it fails. UCP and DTR use the same principles to provide HA, but UCP is more directly tied to Swarm's features. The general principle is to have core services replicated in a cluster, which allows another node to take over when one fails. Load balancers make that transparent to the user by providing a stable hostname, independent of the actual node processing the request. It's the underlying clustering mechanism that provides HA.

Swarm

The foundation of UCP HA is provided by Swarm, the clustering functionality of Docker Engine. As detailed in the Docker Engine documentation (https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/), there are two algorithms involved in managing a Swarm cluster: a Gossip protocol for worker nodes and the Raft consensus algorithm for managers. Gossip protocols are *eventually consistent*, which means that different parts of the cluster might have different versions of a value while new information spreads in the cluster (they are also called *epidemic protocols* because information spreads like a virus). This allows for very large scale cluster because it's not necessary to wait for the whole cluster to agree on a value, while still allowing fast propagation of information to reach consistency in an acceptable time. Managers handle tasks that need to be based on highly consistent information because they need to make decisions based on global cluster and services state.

In practice, *high consistency* can be difficult to achieve without impeding availability because each write needs to be acknowledged by all participants, and a participant being unavailable or slow to respond will impact the whole cluster. This is explained by the CAP Theorem (https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/), which (to simplify) states that in the presence of partitions (P) in a distributed system, we have to chose between consistency (C) or availability (A). Consensus algorithms like Raft address this trade-off using a *quorum*: if a majority of participant agree on a value, it is good enough, the minority participant eventually get the new value. That means that a write needs only acknowledgement from 2 out of 3, 3 out of 5, or 4 out of 7 nodes.

Because of the way consensus works, an odd number of nodes is recommended when configuring Swarm. With 3 manager nodes in Swarm, the cluster can temporarily lose 1 and still have a functional cluster, with 5 you can lose 2, and so on. Conversely, you need 2 managers to acknowledge a write in a 3 manager cluster, but 3 with 5 managers, so more managers don't provide more performance or scalability — you are actually replicating more data. Having 4 managers doesn't add any benefits since you still can only lose 1 (majority is 3), and more data is replicated than with just 3. In practice, it is more fragile.

If you have 3 managers and lose 2, your cluster is non-functional. Existing services and containers keep running, but new requests are not processed. A single remaining manager in a cluster doesn't "switch" to single manager mode. It is just a minority node. You also cannot just promote worker nodes to manager to regain quorum. The failed nodes are still members of the consensus group and need to come back online.

UCP

UCP runs a global service across all cluster nodes called `ucp-agent` . This agent installs a UCP controller on all Swarm manager nodes. There is a one-to-one correspondence between Swarm managers and UCP controllers, but they have different roles. Using its agent, UCP relies on Swarm for HA, but it also includes replicated data stores that rely on their own raft consensus groups that are distinct from Swarm: ucp-auth-store, a replicated database for identity management data, and ucp-kv, a replicated key-value store for UCP configuration data.

DTR

The DTR has a replication model that is similar to how UCP works, but it doesn't synchronize with Swarm. It has one replicated component, its datastore, which might also have a lot of state to replicate at one time. It relies on raft consensus.

Both UCP controllers and DTR replicas may have a lot more state to replicate when (re)joining the cluster. Some reconfiguration operations can make a cluster member temporary unavailable. With 3 members, it's good practice to wait for the one you reconfigured to get back in sync before reconfiguring a second one, or they could lose quorum. Temporary losses in quorum are easily recoverable, but it still means the cluster is in an unhealthy state. Monitoring the state of controllers to ensure the cluster doesn't stay in that state is critical.

Backup and Restore

The HA setup using multiple nodes works well to provide continuous availability in the case of temporary failure, including planned node downtime for maintenance. For other cases, including the loss of the full cluster, permanent loss of quorum, and data loss due to storage faults, restoring from backup is necessary.

UCP Backup

UCP backup is done using the `docker/ucp backup` command on a controller node. It stops the UCP containers on the node and perform a full backup of the configuration and state of UCP. Some of this information is sensitive, so you should always use the `--passphrase` option to encrypt the backup. The backup also includes the orgs, teams, and users used by DTR as well as UCP. Regular backups should be scheduled. Here is an example showing how to run the command without user input:

```
UCPID=$(docker run --rm -i --name ucp -v /var/run/docker.sock:/var/run/docker.sock doc
ker/ucp id)
docker run --rm -i --name ucp -v /var/run/docker.sock:/var/run/docker.sock docker/ucp
backup --id $UCPID --passphrase "secret" > /tmp/backup.tar
```

There are two ways to use the backup: - To restore a controller using the `docker/ucp restore` command (only the backup from that controller can be used) - To install a new cluster using the `docker/ucp install --from-backup` command (preserves users and configuration)

DTR Backup

A DTR backup includes configuration information, images metadata, and certificates. Images themselves need to be backed up separately directly from storage. Remember that users and organizations are managed and backed up by UCP.

The backup can only be used to create a new DTR, using the `docker/dtr restore` command.

Identity Management

Accessing resources (images, containers, volumes, networks etc) and functionality within the components of Docker EE (UCP & DTR) require at a minimum, an account and a corresponding password to be accessed. Accounts within Docker EE are identities stored within an internal database, but the source of creating those accounts and the associated access control can be manual (managed or internal) or external through a connection to a directory server (LDAP) or Active Directory (AD). Managing the authorization for these accounts is an extension of coarse and fine grained permissions that are described in the sections below.

Managed (Internal) Authentication

Managed mode of authentication and authorization is the default mode in Docker EE Standard and Advanced. In this mode, accounts are directly created using the Docker EE API. User accounts can be created manually by accessing the User Management —> Create User form in the UCP UI. Accounts can also be created and managed in an automated fashion by making HTTP requests to the authentication and authorization RESTful service known as eNZi.

User management using the "Managed" mode is recommended only for demo purposes or where the number of users needing to access Docker EE is very small.

Pros:

- Easy and quick to setup
- Simple to troubleshoot
- Appropriate for a small set of users with static roles
- Managed without leaving the UCP interface

Cons:

- User Account management gets cumbersome for larger numbers or when roles need to be managed for several applications.
- All lifecycle changes such as adding / removing permissions of users need to be accomplished manually user by user.
- Users must be deleted manually, meaning access may not get cleaned up quickly, making the system less secure.
- Sophisticated setups of integrating application creation and deployment through LDAP or external systems is not possible.

LDAP / AD Integration

The LDAP method of user account authentication can be turned on to manage user access. As the name suggests, this mode enables automatically synchronization of user accounts from a directory server such as Active Directory or OpenLDAP.

This method is particularly applicable for enterprise use cases where organizations have a large set of users, typically maintained in a centralized identity store that manages both authentication and authorization. Most of these stores are based on a directory server such as Microsoft's Active Directory or a system that supports the LDAP protocol. Additionally, such enterprises already have mature processes for on-boarding, off-boarding, and managing the lifecycle of changes to user and system accounts. All these can be leveraged to provide a seamless and efficient process of access control within Docker EE.

Pros:

- Ability to leverage already established access control processes to grant and revoke permissions.
- Ability to continue managing users and permissions from a centralized system based on LDAP.
- Increased security due to self-cleaning nature of this mode, where non-existent LDAP users are automatically removed from Docker EE on the next sync.
- Ability to configure complex upstream systems such as flat files, database tables using an LDAP proxy, and the automatic time-based de-provisioning of access through AD/LDAP groups.

Cons:

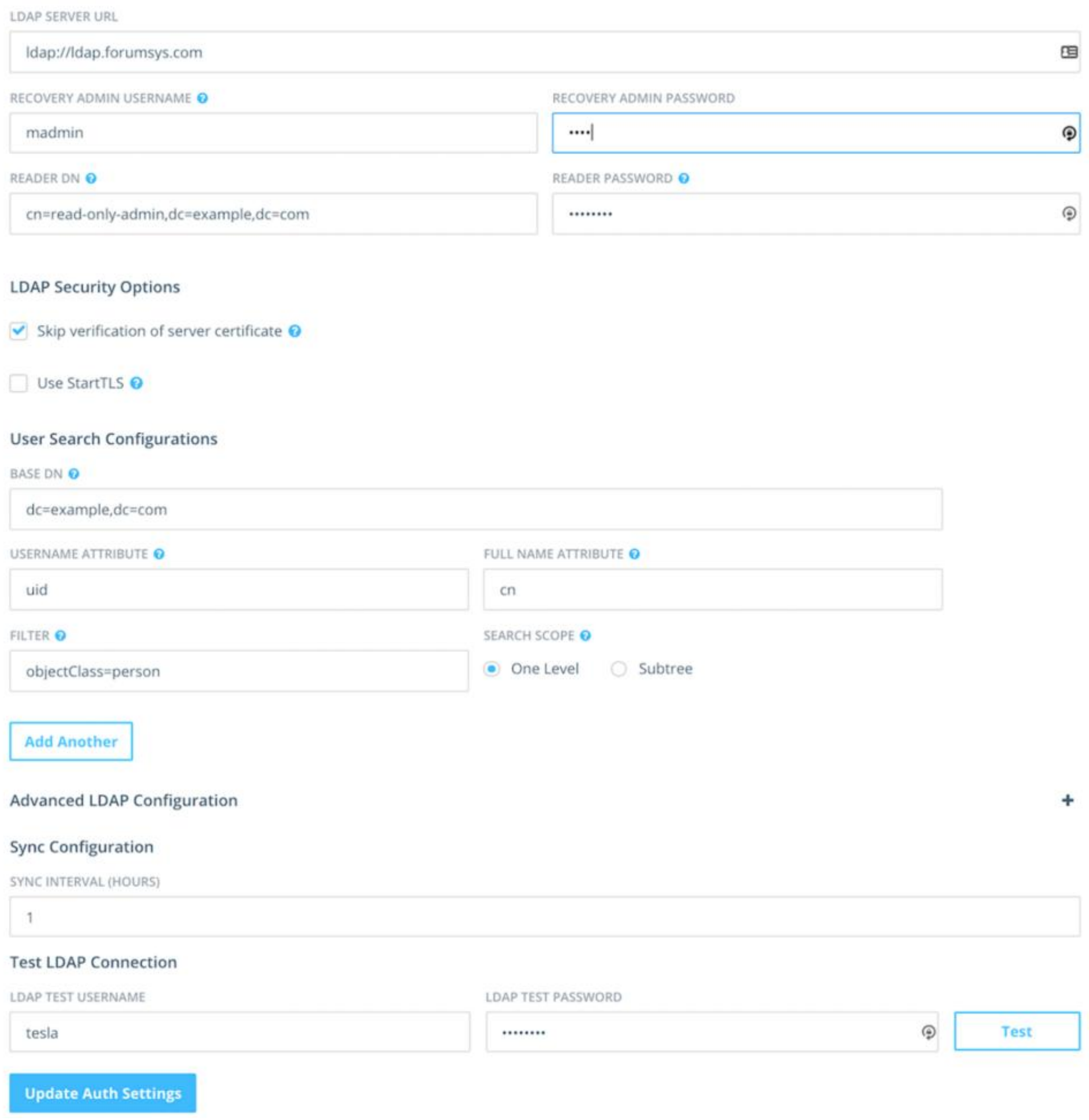
- Increased complexity compared to Managed mode
- Higher admin requirements since knowledge on an external system (LDAP) is needed
- Greater time needed to troubleshoot issues with an extra components (LDAP) in the mix

- Unexpected changes to Docker EE due to changes made to upstream LDAP/AD systems, which can have unexpected effects on Docker EE

A recommended best practice is to use group membership to control access of users accounts to resources. Ideally, the management of such group membership is achieved through a centralized Identity Management or a Role Based Access Control system. This provides a standard, flexible, and scalable model to control the authentication and authorization rules within Docker EE through a centralized directory server. Through the Identity Management system, this directory server is kept in sync with user on-boarding, off-boarding, and changes in roles and responsibilities.

To change the mode of authentication, use the form at Admin Settings —> Auth in the UCP UI. In this form, change the METHOD field under Authentication from the default value of Managed to LDAP.

Accounts that are discovered and subsequently synced from the directory server can be automatically assigned default permissions. It is recommended to leave the option for default permission to No Access Any necessary additional access can be granted based on group memberships.



LDAP SERVER URL
ldap://ldap.forumsys.com

RECOVERY ADMIN USERNAME
madmin

RECOVERY ADMIN PASSWORD
••••

READER DN
cn=read-only-admin,dc=example,dc=com

READER PASSWORD
••••••••

LDAP Security Options

[✓] Skip verification of server certificate

[ ] Use StartTLS

User Search Configurations

BASE DN
dc=example,dc=com

USERNAME ATTRIBUTE
uid

FULL NAME ATTRIBUTE
cn

FILTER
objectClass=person

SEARCH SCOPE
[✓ One Level] Subtree

Advanced LDAP Configuration +

Sync Configuration

SYNC INTERVAL (HOURS)
1

Test LDAP Connection

LDAP TEST USERNAME
tesla

LDAP TEST PASSWORD
••••••••

For details about the LDAP configuration options, read the Integrate with LDAP documentation (https://docs.docker.com/datacenter/ucp/2.0/guides/configuration/integrate-with-ldap/).

This section highlights important configuration options to consider when setting up LDAP authentication.

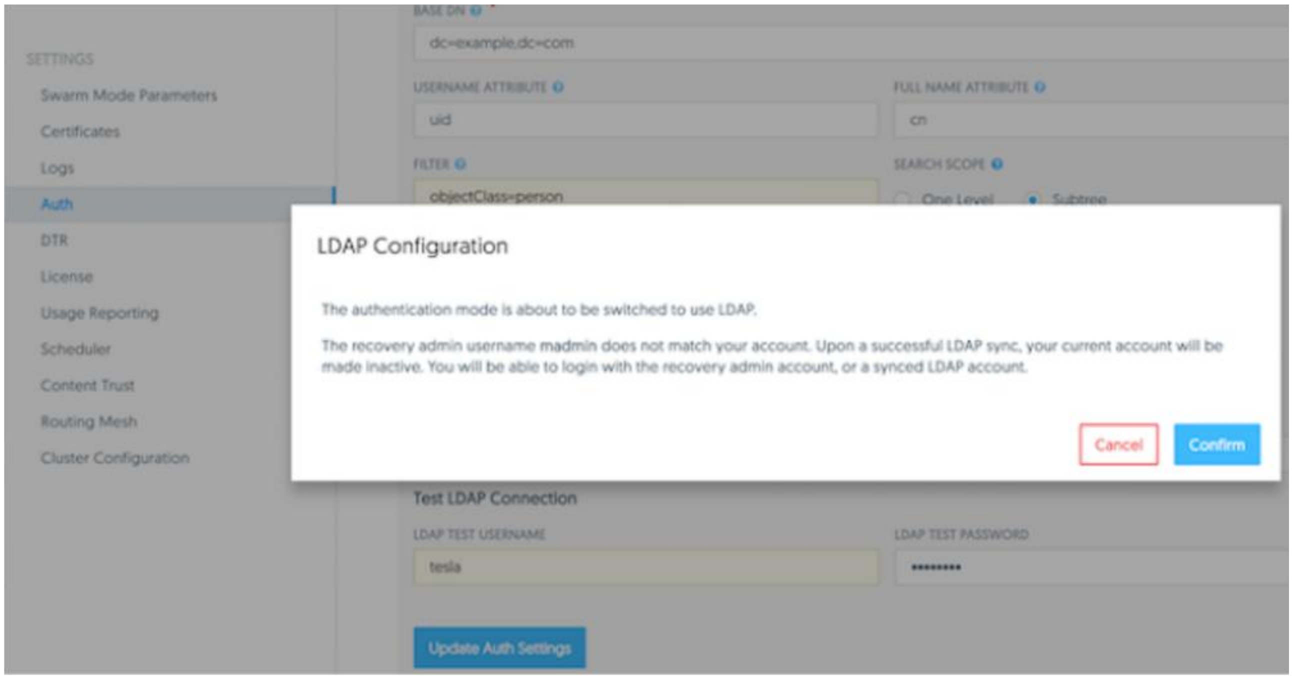
In LDAP auth mode, when accounts are discovered and subsequently imported, all previously created accounts in a Managed mode including the admin accounts are deleted. A new admin account is created when authentication is switched to LDAP. This admin account is a local account and is intended to be useable independent of the LDAP settings (for example, in a situation when there is a loss of connectivity to the directory server). The details of this admin account can be set during the sync using RECOVERY ADMIN USERNAME and RECOVERY ADMIN PASSWORD.

A user account on the directory server has to be configured to discover and import accounts from the directory server. This user account need not be a very powerful account. In fact, it is recommended that it is a read-only account that can view the necessary organizationalUnits (`ou`) and query for group memberships. The details for this account are configured using the fields READER DN and READER PASSWORD. The READER DN must be in the distinguishedName format.

Use secure LDAP if at all possible.

Use the Test LDAP Connection section to make sure you can connect before switching to LDAP authentication.

Once the form is filled out and the test connection succeeds, the sync button provides an option to run a sync immediately without waiting for the next interval. Doing so initiates an LDAP connection and runs the filters to import the users. Any previously logged in user is logged out due to the disablement of all pre-existing accounts.



BASE DN *
dc=example,dc=com
USERNAME ATTRIBUTE
uid
FULL NAME ATTRIBUTE
cn
FILTER
objectClass=person
SEARCH SCOPE
One Level [✓ Subtree]

LDAP Configuration

The authentication mode is about to be switched to use LDAP.

The recovery admin username madmin does not match your account. Upon a successful LDAP sync, your current account will be made inactive. You will be able to login with the recovery admin account, or a synced LDAP account.

Test LDAP Connection
LDAP TEST USERNAME
tesla
LDAP TEST PASSWORD
•••••••••

After the sync is complete, it should be possible to log in using the recovery account and password. Be sure to test this recovery account. It should also be possible to log in using a sync-ed LDAP/AD account (the only supported attributes are `uid` and `sAMAccountName`) and its corresponding LDAP/AD password. The sync-ed account should be in good standing within the LDAP/AD system for the login to be successful within Docker EE.

The progress / status of the sync and any errors that occur can be viewed and analyzed on the controllers by running:

```
docker logs ucp-controller
```

Teams

User accounts that exist within Docker EE, either through an LDAP sync or manually managed, can be organized into teams. Each team created can be assigned customized permissions and managed as a group of users as opposed to granting permissions to each user account on an individual basis.

Teams can be created in the UCP UI using the TEAMS +Create interface under User Management. The same can be accomplished by using the eNZi API that is described in a later section. Members can be added to the team one-by-one. An alternative method of adding members to a team can be employed if using the LDAP Auth mode. This is based on an automatic sync of discovered accounts from the directory server that was configured to enable the LDAP Auth mode. Finer filters can be applied here which determine which discovered accounts are placed into which teams. A team can have multiple users, and a user can be a member of zero to multiple teams.

RBAC and Managing Team Level Access to Resources

Permissions to resources through UCP are managed through default access levels. The default access permissions are `No Access`, `View Only`, `Restricted Control` and `Full Control`. Description about these permissions and how they relate to each other are detailed in the 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). These permissions can be further extended by adding the user accounts to specific teams. In other words, the default permissions assigned directly to a user provide a coarse-grained access control. The user should then be added to teams to attain higher granularity in what a user account can do on specific containers, images, and other resources. The teams are assigned attributes in the form of labels that provide additional flexibility through a fine-grained access control. In some ways this mimics a traditional Role Based Access Control (RBAC), where the teams are the roles assigned to user accounts, but there are specks of the finer Attribute Based Access Control (ABAC) through the assignment of one of more labels to each team, which ultimately define the level of access of the account to UCP resources.

While teams can be associated with as many labels as necessary, the resources within UCP like `services`, `images`, `volumes`, `networks`, and `secrets` can also be associated with any one of these labels to tie the two together. It should be noted that any labels assigned to a service are automatically inherited by the tasks which currently are container workloads. So this behaves in just the same way as assigning meta information in the form of labels to containers (or any other resource for that matter). Internally, the resources are assigned a label with the key `com.docker.ucp.access.label` with the value as specified.

It would be easier to understand with a concrete example as described in the following section.

Strategy for Using LDAP Filters

This section demonstrates a typical use case that uses the principles of "least privilege / permission" as well as "separation of duties."

Suppose you have a simple application called `www`, which is a web server based on the `nginx` official image. Say there are three teams that need access to this application — `developers`, `testers`, and `operations`. Typically, `testers` need `View Only` access and nothing further, while the `operations` team usually needs `Full Control`. The `developers` team needs access to troubleshoot, restart, and control the lifecycle of the application but should be forbidden from any other activity involving the need to access the host file systems, or starting up privileged containers. This type of special access is termed `Restricted Control`.

The users needing to access UCP are all sourced from the corporate Directory Server system. These users are the admin users needed to manage the Docker EE infrastructure as well as all members of each of the teams configured within UCP. Also assume that the total universe of users needing access to Docker EE (includes admins, developers, testers and operations) is a subset of the gamut of users within the Directory Server.

A recommended strategy to use when organizing users is to create an overarching membership group that identifies all users of Docker EE irrespective of which team they are a part of. Let us call this group `Docker_Users`. No user should be made a member of this group directly. Instead, the `Docker_Users` group should have other groups in it and only groups as its members. Per our example, let us call these groups `dev`, `test`, and `ops`. In our example, these groups are part of what is known as a *nested group* structure within the directory server. Nested groups allow the inheritance of permissions from one group to each of its sub-groups.

> NOTE: Some directory servers do not support the feature of nested groups or even the `memberOf` attribute by default. If so, then they would need to be enabled. If the choice of directory server does not support these features at all, then alternate means of organizing users and querying them should be used. Microsoft Active Directory supports both these features out of the box.

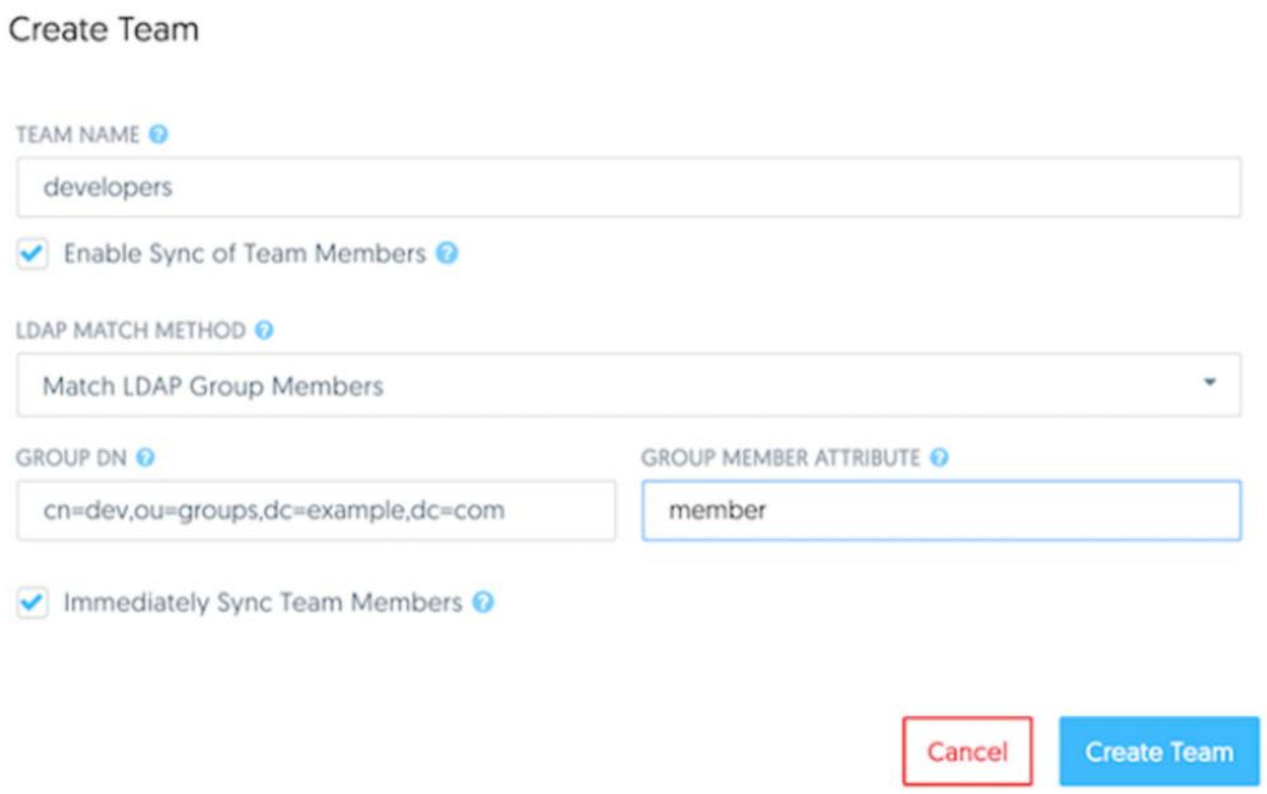
User accounts should be added as members of these sub-groups in the directory server. This should not impact any existing layout in the organization units or pre-existing group membership for these users.

Within UCP, the first step is to enable the LDAP Auth mode. Under User Search Configurations, for the FILTER field, the value based on our example above should be `(&(objectClass=person)(memberOf=cn=Docker_Users,ou=groups,dc=example,dc=com))`.

Performing a sync with this setting discovers and import all users that are members of the group

`Docker_Users` (through the nested sub-groups).

Next, using the TEAMS +Create link under User Management, the individual teams can be created. An example of creating the `developers` team is shown below. The value of the Group DN field reflects the sub-group created under `Docker_Users` called `dev` .


Create Team

TEAM NAME: developers

[✓] Enable Sync of Team Members

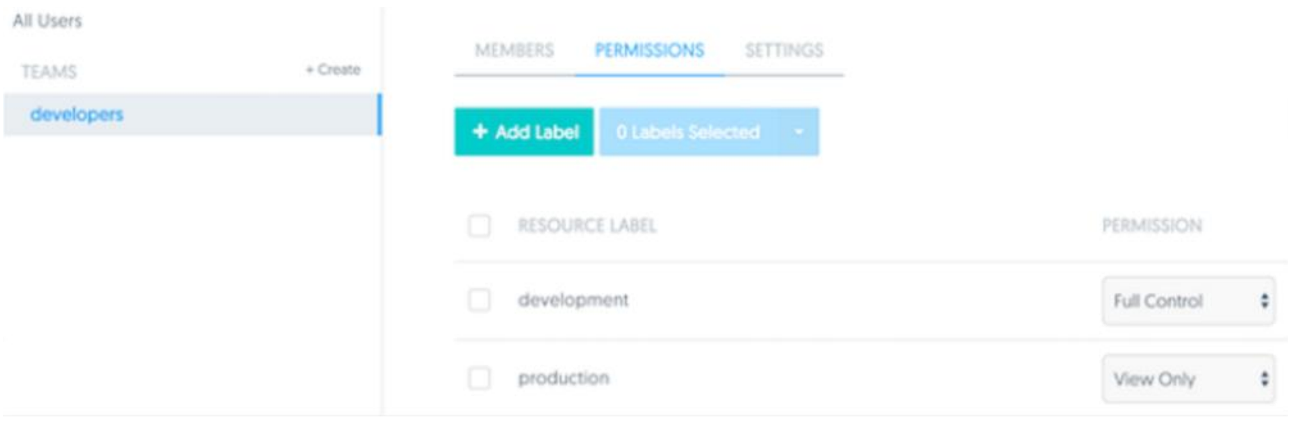
LDAP MATCH METHOD: Match LDAP Group Members

GROUP DN: cn=dev,ou=groups,dc=example,dc=com

GROUP MEMBER ATTRIBUTE: member

[✓] Immediately Sync Team Members

After the team is created, bring up the Add Label form by clicking on the PERMISSIONS link. Here, as many permissions as needed can be added. In the image below, two labels are added to denote that members of the `developers` team have `View Only` access to any resource labeled as `production` . An additional label grants the members of the `developers` team `Full Control` on resources labeled as `development` .



RESOURCE LABEL	PERMISSION
development	Full Control
production	View Only

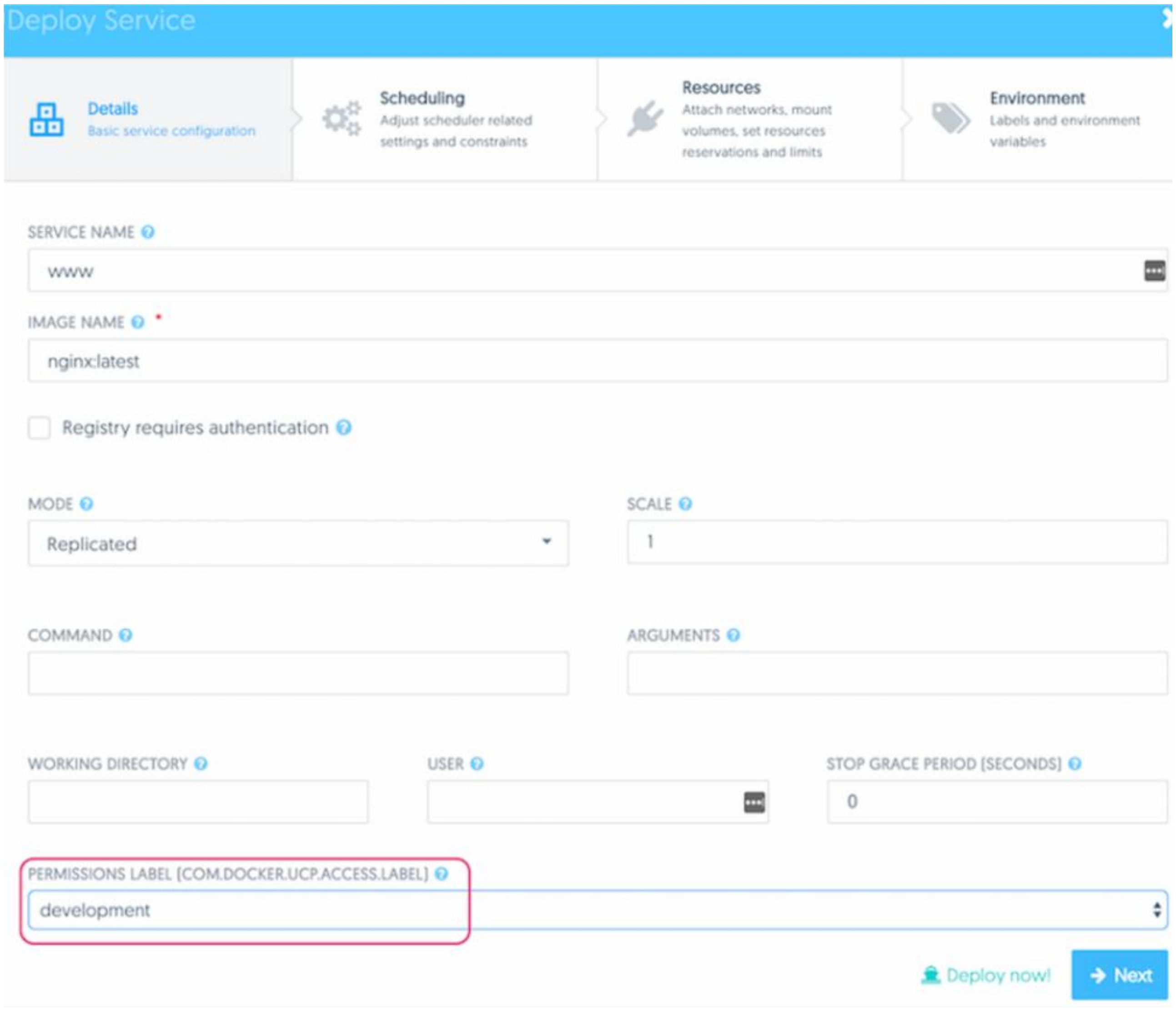
Similarly, the `testers` and `operations` teams are created and assigned permissions per the table below:

Team	Group	Permissions
`testers`	`cn=test,ou=groups,dc=example,dc=com`	`production`: `No Access`
`development`: `View Only`		
`operations`	`cn=ops,ou=groups,dc=example,dc=com`	`production`: `Full Control`
`development`: `Full Control`		

Now, you can now create the service `www` and assign it the appropriate `access label` to allow only specific users access to the resource using a command like below:		

```
docker service create --name www -p 80:80 --label com.docker.ucp.access.label=developm
ent nginx
```

The same can be done in the UCP UI while creating a service by selecting an appropriate label (`development` in this example) from the PERMISSIONS LABEL dropdown as shown below. The label is applied to the service for enforcing role-based access control.


Deploy Service

Details
Basic service configuration

Scheduling
Adjust scheduler related settings and constraints

Resources
Attach networks, mount volumes, set resources reservations and limits

Environment
Labels and environment variables

SERVICE NAME: www

IMAGE NAME *: nginx:latest

☐ Registry requires authentication

MODE: Replicated

SCALE: 1

COMMAND:

ARGUMENTS:

WORKING DIRECTORY:

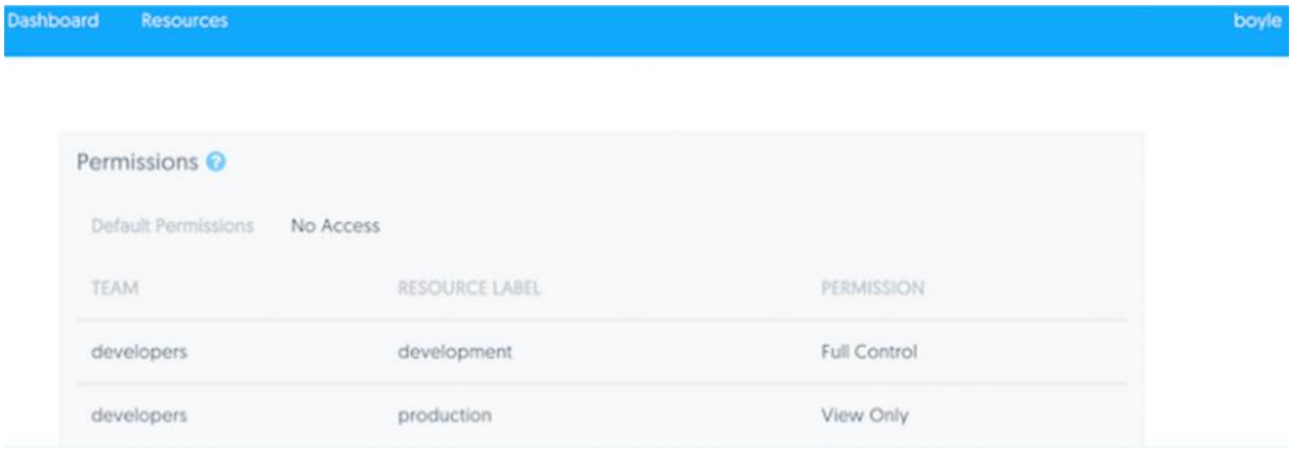
USER:

STOP GRACE PERIOD (SECONDS): 0

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

Only those labels that the logged in user has access to shows up in the dropdown. An admin user or a user assigned `Full Control` permission is shown all available labels regardless of the labels assigned to the team of which the user is a member of.

Now, create another service `www-prod` similarly based on the `nginx` image but with the Permission Label of `production` applied to it while logged in as a user `boyle` who is a member of the developers team. His permissions look as follows:



Permissions

Default Permissions No Access

TEAM	RESOURCE LABEL	PERMISSION
developers	development	Full Control
developers	production	View Only

The account `boyle` can see both services, `www` and `www-prod`, but only the `www` service allows edits and other control because the permission label of `development` matches with the team's resource label of `development` granted with `Full Control` permissions. The `www-prod` service forbids any edits using the account `boyle` because it has a permission label of `production`, which has been marked as `View Only` in the team `developers'` resource label. For example, if the account `boyle` was used to scale up the number of replicas of each service, from 1 to 2 (for instance), an error of `Access Denied` would be thrown for the `www-prod` service, while the same would work for `www` service.

Further, as evident in the profile image, `boyle` has `Default Permissions` of `No Access`. This prevents the `boyle` account from accessing any other resource unless the permission label and team resource label match and provide explicit permissions that override the `No Access` default permission.

Finally, if and when it becomes necessary to terminate all access for any user account, removing the group membership of the account from just the one group `Docker_Users` would remove all access for the user. Due to the nature of how nested groups work, all additional access within Docker is automatically cleaned — the user account is removed from any and all team memberships at the time of the next sync without need for manual intervention or additional steps. This step can be integrated into a standard on-boarding / off-boarding automated provisioning step within a corporate Identity Management system.

Authentication API (eNZi)

The AuthN API or eNZi (as it is known internally and pronounced N-Z) is a centralized authentication and authorization service and framework for Docker EE. This API is completely integrated and configured into Docker EE and works seamlessly with UCP as well as DTR. This is the component and service under the hood that manages accounts, teams and organizations, user sessions, permissions and access control through labels, Single-Sign-On (Web SSO) through OpenID Connect, and synchronization of account details from an external LDAP-based system into Docker EE.

For regular day-to-day activities, users and operators need not be concerned with the AuthN API and how it works, however its features can be leveraged to automate many common functions and/or bypass the UCP UI altogether to manage and manipulate the data directly.

The RESTFul AuthN API endpoints are documented in the AuthN API (https://docs.docker.com/apidocs/v2.0.1/#!/accounts/func1).

Interaction with AuthN can be accomplished in two ways: via the exposed RESTful AuthN API over HTTP or via the `enzi` command.

For example, the command below uses `curl` and `jq` to fetch all user accounts in Docker EE via the AuthN API over HTTP:

```
curl --silent --insecure --header "Authorization: Bearer $(curl --silent --insecure \
     --data '{"username":"<admin-username>","password":"<admin-password>"}' \
    https://<UCP-domain-name>/auth/login | jq --raw-output .auth_token)" \
    https://<UCP-domain-name>/enzi/v0/accounts | jq .
```

The AuthN service can also be invoked on the CLI on a UCP controller. To connect into it, run the following on a UCP controller:

```
docker exec -it ucp-auth-api sh
```

At the resulting prompt (`#`), type the `enzi` command with a sub-command such as the one below to list the database table status:

```
enzi db-status
```

> Note: Refer to Recovering the Admin Password for Docker EE Standard and Advanced (https://success.docker.com/api/asset/.%2Frefarch%2Fdocker-ee-best-practices-17-03%2F%2Farticle%2FRecovering_the_Admin_Password_for_Docker_EE_Standard_and_Advanced) for a detailed example.

What is Docker (https://www.docker.com/what-docker)

What is a Container (https://www.docker.com/what-container)

Use Cases (https://www.docker.com/use-cases)

Customers (https://www.docker.com/customers)

For Government (https://www.docker.com/industry-government)

For IT Pros (https://www.docker.com/itpro)

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)

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

Pricing (https://www.docker.com/pricing)

Community Edition (https://www.docker.com/community-edition)

Enterprise Edition (https://www.docker.com/enterprise-edition)

Docker Datacenter (https://www.docker.com/enterprise-edition#container_management)

Docker Cloud (https://cloud.docker.com/)

Docker Store (https://store.docker.com/)

Get Docker (https://www.docker.com/get-docker)

Docker for Mac (https://www.docker.com/docker-mac)

Docker for Windows(PC) (https://www.docker.com/docker-windows)

Docker for AWS (https://www.docker.com/docker-aws)

Docker for Azure (https://www.docker.com/docker-azure)

Docker for Windows Server (https://www.docker.com/docker-windows-server)

Docker for Debian (https://www.docker.com/docker-debian)

Docker for Fedora® (https://www.docker.com/docker-fedora)

Docker for Oracle Linux (https://www.docker.com/docker-oracle-linux)

Docker for RHEL (https://www.docker.com/docker-red-hat-enterprise-linux-rhel)

Docker for SLES (https://www.docker.com/docker-suse-linux-enterprise-server-sles)

Docker for Ubuntu (https://www.docker.com/docker-ubuntu)

Documentation (https://docs.docker.com/)

Blog (https://blog.docker.com/)

RSS Feed (https://blog.docker.com/feed/)

Training (https://training.docker.com/)

Knowledge Base (https://success.docker.com/kbase)

Resources (https://www.docker.com/products/resources)

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

Open Source (https://www.docker.com/technologies/overview)

Events (https://www.docker.com/community/events)

Forums (https://forums.docker.com/)

Docker Captains (https://www.docker.com/community/docker-captains)

Scholarships (https://www.docker.com/community-partnerships)

Community News (https://blog.docker.com/curated/)

Status (http://status.docker.com/) Security (https://www.docker.com/docker-security)

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

(http: (http: (https (https (https (https (http: (http:

Authored by: Anoop Kumar (/author/anoop.kumar) and Alex Drahon (/author/alex.drahon)

32 0 Share PDF (https://success.docker.com/api/articles/docker-ee-best-practices-17-03/pdf)

Docker Reference Architecture: Docker EE Best Practices and Design Considerations (EE 17.03)

EE LINUX WINDOWS SERVER DEPLOY EE-17.03.0-EE-1 UCP 2.1.0 DTR-2.2.1 CSE-1.13.1-CS1

Introduction

Docker Enterprise Edition (Docker EE) is the enterprise container platform from Docker to be used across the entire software supply chain. It is a fully-integrated solution for container-based application development, deployment, and management. With integrated end-to-end security, Docker EE enables application portability by abstracting your infrastructure so that applications can move seamlessly from development to production.

What You Will Learn

This reference architecture describes a standard, production-grade, Docker EE deployment. It also details the components of Docker EE, how they work, how to automate deployments, how to manage users and teams, how to provide high availability for the platform, and how to manage the infrastructure.

Some environment-specific configuration details are not be provided. For instance, load balancers vary greatly between cloud platforms and on-premises infrastructure platform. For these types of components, general guidelines to environment-specific resources are provided.

Understanding Docker Components

From development to production, Docker EE provides a seamless platform for containerized applications both on-premises and on the Cloud. Docker EE is available in three tiers to meet different application needs. Both Docker EE Standard (formerly known as Docker Datacenter) and Docker EE Advanced include the following components:

- Docker CS Engine, the commercially supported Docker Engine
- Universal Control Plane (UCP), the web-based, unified cluster and application management solution
- Docker Trusted Registry (DTR), a resilient and secure image management repository

Together they provide an integrated solution with the following design goals:

- Agility — the Docker API is used to interface with the platform so that operational features do not slow down application delivery
- Portability — the platform abstracts details of the infrastructure for applications
- Control — the environment is secure by default, provides robust access control, and logging of all operations

To achieve these goals the platform must be resilient and highly available. This reference architecture demonstrates this robust configuration.

Docker CS Engine

The building block of Docker EE, the Docker Commercially Supported Engine (CS Engine) is responsible for container-level operations, interaction with the OS, providing the Docker API, and running the Swarm cluster. The Engine is also the integration point for infrastructure, including the OS resources, networking, and storage.

Universal Control Plane

UCP extends CS Engine by providing an integrated application management platform. It is both the main interaction point for users and the integration point for applications. UCP runs an agent on all nodes in the cluster to monitor them and a set of services on the *controller nodes*. This includes *identity services* to manage users, *Certificate Authorities* (CA) for user and cluster PKI, the main *controller* providing the Web UI and API, data stores for UCP state, and a *Classic Swarm* service for backward compatibility.

Docker Trusted Registry

The DTR is an application managed by, and integrated with UCP, that provides Docker images distribution and security services. The DTR uses UCP's identity services to provide Single Sign-On (SSO), and establish a mutual trust to integrate with its PKI. It runs as a set of services on one or several *replicas*: the *registry* to store and distribute images, an image signing service, a Web UI, an API, and data stores for image metadata and DTR state.

Swarm Mode

To provide a seamless cluster based on a number of nodes, DDC relies on CS Engine's *swarm mode* (https://docs.docker.com/engine/swarm/key-concepts/) capability. Swarm mode divides nodes between *workers*, nodes running application workloads defined as services, and *managers*, nodes in charge of maintaining desired state, managing the cluster's internal PKI, and providing an API. Managers can also run workloads. In a Docker EE environment they run UCP controllers and shouldn't run anything else.

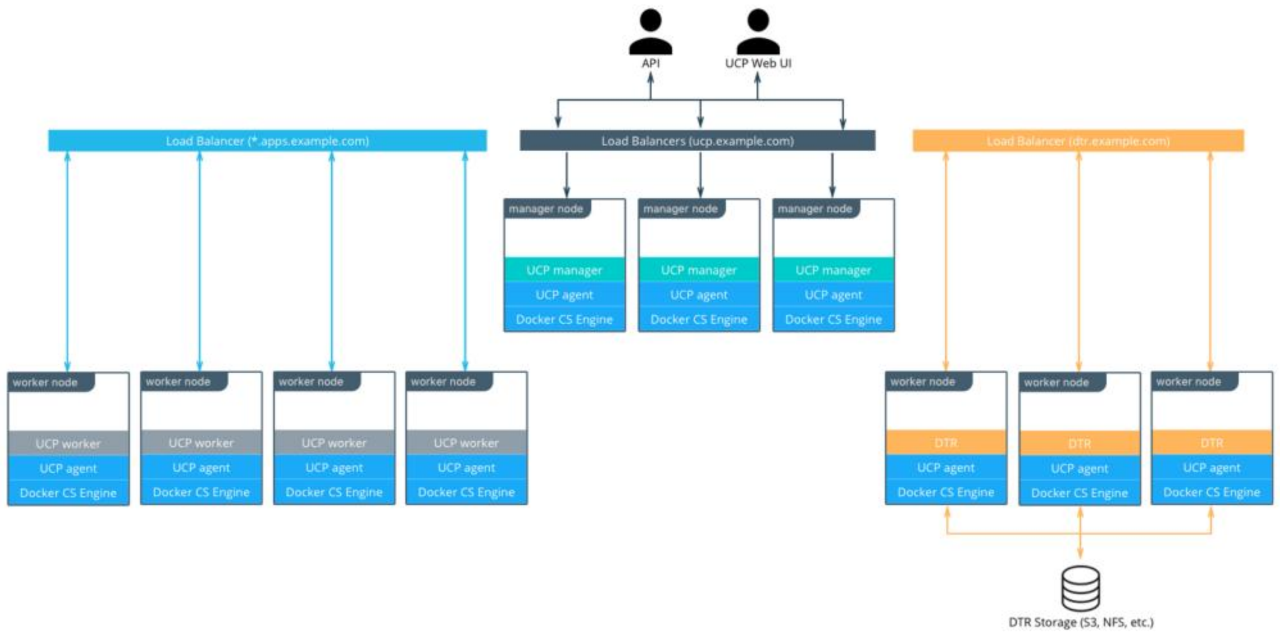
The Swarm mode service model provides a declarative desired state for workloads, scalable to a number of *tasks* (the service's containers), accessible through a stable resolvable name, and optionally exposing an end-point. Exposed services are accessible from any node on a cluster-wide reserved port, reaching tasks through the *routing mesh*, a fast routing layer using high-performance switching in the Linux kernel. This set of features enable internal and external discoverability for services, UCP's *HTTP Routing Mesh* (HRM) adding hostname-to-service mapping.

A Standard Deployment Architecture

This section demonstrates a standard, production-level architecture for Docker EE using 10 nodes: 3 UCP controllers, 3 nodes for the DTR, and 4 worker nodes for application workloads. The number of worker nodes is arbitrary, most environments will have more depending on applications needs, it doesn't change the architecture or the cluster configuration.

Access to the environment is done through 3 Load Balancers (or 3 load balancer virtual hosts) with corresponding DNS entries for the UCP controllers, the DTR replicas, and the applications running in the cluster.

The DTR replicas use shared storage for images. S3-compatible object storage (the default) and NFS storage are both covered in this section.



Node Size

A node is a machine in the cluster (virtual or physical) with Docker Engine running on it. When provisioning each node, assign it a role: UCP Controller, DTR, or worker node so that they are protected from running application workloads.

To decide what size the node should be in terms of CPU, RAM, and storage resources, consider the following:

1. All nodes should at least fulfill the minimal requirements for UCP 2.0: 2 GB of RAM and 3 GB of storage. More detailed requirements are in the UCP documentation (https://docs.docker.com/datacenter/ucp/2.0/guides/installation/system-requirements/).
2. UCP Controller nodes should be provided with more than the minimal requirements but won't need much more if nothing else runs on them.
3. Ideal Worker nodes size will vary based on your workloads, so it is impossible to define a universal standard size.
4. Other considerations like target density (average number of container per node), whether one standard node type or several are preferred, and other operational considerations might also influence sizing.

If possible, node size should be determined by experimentation and testing actual workloads, and they should be refined iteratively. A good starting point is to select a standard or default machine type in your environment and use this size only. If your standard machine type provides more resources than the UCP Controllers need, it makes sense to have a smaller node size for these. Whatever the starting choice, it's important to monitoring resource usage and cost to improve the model.

Two example scenarios:

- One standard node size: 2 VCPU, 8 GB RAM, 20 GB storage
- Two node sizes: 2 VCPU, 8 GB RAM, 20 GB storage for UCP Controllers and 4 VCPU, 64 GB RAM, 40 GB storage for worker nodes

Depending on your OS of choice, storage configuration for Docker Engine might require some planning. Refer to the Support Matrix (https://success.docker.com/Policies/Compatibility_Matrix) to see what storage drivers (https://docs.docker.com/engine/userguide/storagedriver/selectadriver/) are supported for your host OS. This is especially important if you are using RHEL or CentOS, which use device mapper with direct-lvm (https://docs.docker.com/engine/userguide/storagedriver/device-mapper-driver/#/configure-direct-lvm-mode-for-production).

Load Balancers

Load balancers configuration should be done before installation, including the creation of DNS entries. Most load balancers should work with Docker EE. The only requirements are TCP passthrough and the ability to do health checks on an HTTPS endpoint.

In our example architecture, the three UCP controllers ensure UCP resiliency in case of node failure or controller reconfiguration. Access to UCP through the GUI or API is always done using TLS. The load balancer is configured for simple TCP pass-through on port 443, using a custom HTTPS health check at `https://<ucp_controller>/_ping` .

Be sure to create a DNS entry for the UCP host such as `ucp.example.com` and point to the load balancer.

The setup for the three DTR replicas is similar to setting up the UCP controllers. Again, use TCP pass-through to port 443 on the nodes, except the health check is at `https://<dtr_replica_node>/health` .

Create a DNS entry for the DTR host such as `dtr.example.com` and point it to the load balancer. It's important to keep it as concise as possible because it will be part of the full name of images. For example, `user_a` 's `webserver` image will be named `dtr.example.com/user_a/webserver` .

The application load balancer provides access to services HTTP endpoints exposed through UCP's HTTP Routing Mesh (HRM). HRM provides a reverse-proxy to map domain names to services that expose ports and are attached to the `ucp-hrm` overlay network. As an example, the `voting` application exposes the `vote` service's port `80` . The HRM maps `[http://vote.apps.example.com](http://vote.apps.example.com "http://vote.apps.example.com")` to this port on the `ucp-hrm` overlay network, and the application LB itself maps `*.apps.example.com` to nodes in the cluster.

For more details on load balancing for applications on UCP, see Universal Control Plane 2.0 Service Discovery and Load Balancing (https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Universal_Control_Plane_2.0_Service_Discovery_and_Load_Balancing).

DTR Storage

The DTR usually needs to store a large number of images. It uses external storage (S3, NFS, etc.), not node storage so that it can be shared between DTR replicas. The DTR replicates metadata and configuration information between replicas, but not image layers themselves. To determine storage size, start with the size of the existing images used in the environment and evolve from there.

It's best to use an existing storage solution in your environment so that image storage can benefit from existing operations experience. If you have to chose a new solution, consider using S3-compatible object storage, which maps more closely to registry operations.

Refer to An Introduction to Storage Solutions for Docker CaaS (https://success.docker.com/Architecture/An_Introduction_to_Storage_Solutions_for_Docker_CaaS) for more information about selecting storage solutions.

Recommendations for the Docker EE Installation

This section details the installation process for the architecture and provide a checklist. It is not a substitute for the documentation, which provides more details and is authoritative in any case. The goal is to help you define a repeatable (and ideally automated) process to deploy, configure, upgrade and expand your Docker EE environment.

The three main stages of a Docker EE Standard or Advanced installation are as follows:

1. Deploying and configuring the infrastructure (hosts, network, storage)
2. Installing and configuring the Docker Engine, running as an application on the hosts
3. Installing and configuring UCP and the DTR, delivered as containers running on the Engine

Infrastructure Considerations

The installation documentation (https://docs.docker.com/datacenter/install/linux/) details infrastructure requirements for Docker EE Standard and Advanced. It is recommended to use existing or platform specific tools in your environment to provide standardized and repeatable configuration for infrastructure components.

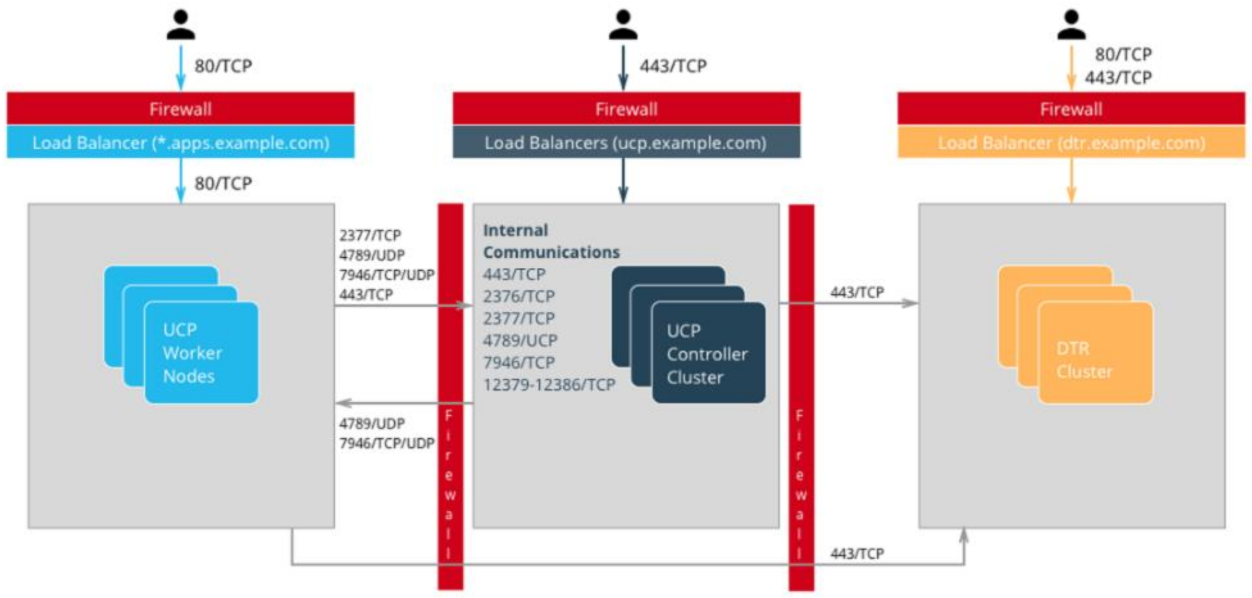
Network

Docker components need to communicate over the network, and the documentation lists the ports that need to be open (https://docs.docker.com/datacenter/ucp/2.0/guides/installation/system-requirements/) for internal cluster communication. Misconfiguration of the cluster's internal network can lead to issues that might be difficult to track down. It's better to start with a relatively simple environment. This reference architecture assumes a single subnet for all hosts and overlay networking for containers.

To get more details and evaluate options, consult Docker Reference Architecture: Designing Scalable, Portable Docker Container Networks (https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Designing_Scalable%2C_Portable_Docker_Container_Networks).

Firewall

Access to Docker EE is done using port 443 only (ports 443 and 80 for DTR), whether it’s to access the Web UI or the remote API. This makes external firewall configuration simple. In most cases you only need to open ports 80, 443, and 22. Port 22 is for SSH access is optional since Docker EE doesn’t require SSH access. Access to applications is through a load balancer using HTTPS. If you expose other TCP services to the outside world, open these ports on the firewall. As explained in the previous section, several ports need to be open for communication inside the cluster. If you have a firewall between some nodes in the cluster, for example, to separate controllers from worker nodes, open the relevant port there too.



Load Balancers

Load balancers are detailed in the previous section. They must be in place before installation and must be provisioned with the right hostnames. External (load balancer) hostnames are used for HA and also for TLS certificates. It’s easier not having to reconfigure them during or after installation.

Refer to the previous section Understanding Docker Components (https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Docker_EE_Best_Practices_and_Design_Considerations#ddc-components) for details about load balancer configuration.

Shared Storage

DTR shared storage, used for images in the registry, must be ready and accessible from the DTR nodes. Test it works using an S3 or NFS command line client to avoid having to debug DTR storage configuration.

Host Configuration

Host configuration varies based on the OS used and existing configuration standards, but there are some important steps that must be followed after OS installation:

1. Clock synchronization using NTP or similar service. Clock skew can produce hard to debug errors, especially where the Raft algorithm is used (UCP and the DTR).
2. Static IPs are required for UCP for all hosts.
3. Hostnames are used for node identification in the cluster. The hostname must be set in an non-ephemeral way.

4. Host firewalls must allow intra-cluster traffic on all the ports specified in the installation docs.
5. Storage must be configured if needed. For example, the `devicemapper` driver need a logical volume configured before Docker Engine installation.

Docker CS Engine Installation Considerations

Detailed installation instructions for CS Engine are provided by the documentation (https://docs.docker.com/cs-engine/1.13/). To install on machines that don't have Internet access, add these package to your own private repository. After installing the package, make sure the `docker` service is configured to start on system boot.

The best way to change parameters for CS Engine is to use the `/etc/docker/daemon.json` configuration file. This ensures that the configuration can be reused across different systems and OS in a consistent way. See the documentation for the full list of options (https://docs.docker.com/engine/reference/commandline/dockerd/#/daemon-configuration-file).

Make sure Engine is configured correctly by starting the `docker` service and verifying the parameters with `docker info` .

UCP Installation Considerations

The UCP installer creates a functional cluster from a set of machines running CS Engine. That includes creating a Swarm cluster and installing the UCP controllers. The default installation mode as described in the Installation Guide (https://docs.docker.com/datacenter/ucp/2.0/guides/installation/#/step-4-install-ucp) is interactive.

To perform fully-automated, repeatable deployments, provide more information to the installer:

```
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
  -v /tmp/docker_subscription.lic:/config/docker_subscription.lic \
  -e UCP_ADMIN_PASSWORD=password --name ucp docker/ucp install --host-address IP_or_in
terface \
  --san manager1.example.com --san ucp.example.com
```

Each of these options are explained in the following sections.

External Certificates

By default UCP uses self-signed TLS certificates. For production deployments, it is recommended to use certificates generated by a trusted CA. In most cases, this is your organization’s internal CA.

The certificates and keys needed are as follows:

1. The root CA's public certificate `ca.pem`.
2. The TLS certificate `cert.pem`. It must also include the public certificates of any intermediate CA and have SANs for all addresses used to access UCP, including the load balancer's hostname (e.g. `ucp.example.com`) and the individual controllers' hostnames (e.g. `ucp-controller1.example.com`) to access them directly.
3. The TLS private key, `key.pem`.

To add these automatically during installation, add these files with the exact names to a volume named `ucp-controller-server-certs` on the machine where you install UCP, and use the `--external-server-cert` install parameter.

It is also possible to add the certificates through the Web UI after installation.

License File

The license file can be provided for installation via the command line as well using a bind-mount (volume) in `/config`. Specify its location with `-v /path/to/docker_subscription.lic:/config/docker_subscription.lic`.

Admin Password

To make the installation entirely non-interactive, the admin password must be passed using the `--admin-password` install parameter. The admin username is `admin` by default. Use `--admin-username` to change it.

The full list of install parameters is provided in the install command documentation (https://docs.docker.com/datacenter/ucp/2.0/reference/cli/install/#/options).

Adding Nodes

Once the installation has finished for the first controller node, the two additional controllers must be installed by joining them to the cluster. UCP configures a full controller replica on the manager node in the cluster, so the only command needed on the two controllers is a `docker swarm join` with the right token. The exact command can be obtained by running `docker swarm join-token manager` on the first controller.

To join the worker nodes, the equivalent command can be obtained with `docker swarm join-token worker` on any controller:

```
docker swarm join-token worker
```

To add a worker to this swarm, run the following command:

```
docker swarm join \
--token SWMTKN-1-00gqkzjo07dxcxb53qs4brml51vm6ca2e8fjnd6dds8lyn9ng1-092vhgjxz3jixvjf08
1sdge3p \
192.168.65.2:2377
```

To make sure everything is running correctly, log into UCP at `[https://ucp.example.com](https://ucp.example.com "https://ucp.example.com")`.

DTR Installation Considerations

Installation of the DTR is similar to that of UCP. Install and configure one node, and then join replicas to form a full, highly-available setup. For installation of the first instance as well as replicas, point the installer to the node in the cluster it will install on.

Certificates and image storage must be configured after installation. Once shared storage is configured, the two replicas can be added with the `join` command.

Validating the Deployment

When installation of everything has finished, tests can be done to validate the deployment. Disable scheduling of workloads on UCP controllers and nodes running the DTR.

Basic tests to consider:

1. Log in through `[https://ucp.example.com](https://ucp.example.com "https://ucp.example.com")` as well as directly to a manager node, eg. `[https://manager1.example.com](https://manager1.example.com "https://manager1.example.com")`. Make sure the cluster and all nodes are healthy.
2. Test that you can deploy an application following the example in the documentation (https://docs.docker.com/datacenter/ucp/2.0/guides/applications/).
3. Test that users can download a bundle and connect to the cluster from the CLI. Test that they can use docker-compose (https://docs.docker.com/compose/overview/).
4. Test DTR with a full image workflow. Make sure storage isn't misconfigured and images are stored in the right place.

Consider building a standard automated test suite to validate new environments and updates. Just testing standard functionality should hit most configuration issues. Make sure you run these tests with a non-admin user, the test user should have similar rights as users of the platform. Measuring time taken by each test can also pinpoint issues with underlying infrastructure configuration. Fully deploying an actual application from your organization should be part of this test suite.

High Availability in Docker EE

In a production environment, it is vital that critical services have minimal downtime. It is important to understand how high availability (HA) is achieved in UCP and the DTR, and what to do to when it fails. UCP and DTR use the same principles to provide HA, but UCP is more directly tied to Swarm's features. The general principle is to have core services replicated in a cluster, which allows another node to take over when one fails. Load balancers make that transparent to the user by providing a stable hostname, independent of the actual node processing the request. It's the underlying clustering mechanism that provides HA.

Swarm

The foundation of UCP HA is provided by Swarm, the clustering functionality of Docker Engine. As detailed in the Docker Engine documentation (https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/), there are two algorithms involved in managing a Swarm cluster: a Gossip protocol for worker nodes and the Raft consensus algorithm for managers. Gossip protocols are *eventually consistent*, which means that different parts of the cluster might have different versions of a value while new information spreads in the cluster (they are also called *epidemic protocols* because information spreads like a virus). This allows for very large scale cluster because it's not necessary to wait for the whole cluster to agree on a value, while still allowing fast propagation of information to reach consistency in an acceptable time. Managers handle tasks that need to be based on highly consistent information because they need to make decisions based on global cluster and services state.

In practice, *high consistency* can be difficult to achieve without impeding availability because each write needs to be acknowledged by all participants, and a participant being unavailable or slow to respond will impact the whole cluster. This is explained by the CAP Theorem (https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/), which (to simplify) states that in the presence of partitions (P) in a distributed system, we have to chose between consistency (C) or availability (A). Consensus algorithms like Raft address this trade-off using a *quorum*: if a majority of participant agree on a value, it is good enough, the minority participant eventually get the new value. That means that a write needs only acknowledgement from 2 out of 3, 3 out of 5, or 4 out of 7 nodes.

Because of the way consensus works, an odd number of nodes is recommended when configuring Swarm. With 3 manager nodes in Swarm, the cluster can temporarily lose 1 and still have a functional cluster, with 5 you can lose 2, and so on. Conversely, you need 2 managers to acknowledge a write in a 3 manager cluster, but 3 with 5 managers, so more managers don't provide more performance or scalability — you are actually replicating more data. Having 4 managers doesn't add any benefits since you still can only lose 1 (majority is 3), and more data is replicated than with just 3. In practice, it is more fragile.

If you have 3 managers and lose 2, your cluster is non-functional. Existing services and containers keep running, but new requests are not processed. A single remaining manager in a cluster doesn't "switch" to single manager mode. It is just a minority node. You also cannot just promote worker nodes to manager to regain quorum. The failed nodes are still members of the consensus group and need to come back online.

UCP

UCP runs a global service across all cluster nodes called `ucp-agent` . This agent installs a UCP controller on all Swarm manager nodes. There is a one-to-one correspondence between Swarm managers and UCP controllers, but they have different roles. Using its agent, UCP relies on Swarm for HA, but it also includes replicated data stores that rely on their own raft consensus groups that are distinct from Swarm: ucp-auth-store, a replicated database for identity management data, and ucp-kv, a replicated key-value store for UCP configuration data.

DTR

The DTR has a replication model that is similar to how UCP works, but it doesn't synchronize with Swarm. It has one replicated component, its datastore, which might also have a lot of state to replicate at one time. It relies on raft consensus.

Both UCP controllers and DTR replicas may have a lot more state to replicate when (re)joining the cluster. Some reconfiguration operations can make a cluster member temporary unavailable. With 3 members, it's good practice to wait for the one you reconfigured to get back in sync before reconfiguring a second one, or they could lose quorum. Temporary losses in quorum are easily recoverable, but it still means the cluster is in an unhealthy state. Monitoring the state of controllers to ensure the cluster doesn't stay in that state is critical.

Backup and Restore

The HA setup using multiple nodes works well to provide continuous availability in the case of temporary failure, including planned node downtime for maintenance. For other cases, including the loss of the full cluster, permanent loss of quorum, and data loss due to storage faults, restoring from backup is necessary.

UCP Backup

UCP backup is done using the `docker/ucp backup` command on a controller node. It stops the UCP containers on the node and perform a full backup of the configuration and state of UCP. Some of this information is sensitive, so you should always use the `--passphrase` option to encrypt the backup. The backup also includes the orgs, teams, and users used by DTR as well as UCP. Regular backups should be scheduled. Here is an example showing how to run the command without user input:

```
UCPID=$(docker run --rm -i --name ucp -v /var/run/docker.sock:/var/run/docker.sock doc
ker/ucp id)
docker run --rm -i --name ucp -v /var/run/docker.sock:/var/run/docker.sock docker/ucp
backup --id $UCPID --passphrase "secret" > /tmp/backup.tar
```

There are two ways to use the backup: - To restore a controller using the `docker/ucp restore` command (only the backup from that controller can be used) - To install a new cluster using the `docker/ucp install --from-backup` command (preserves users and configuration)

DTR Backup

A DTR backup includes configuration information, images metadata, and certificates. Images themselves need to be backed up separately directly from storage. Remember that users and organizations are managed and backed up by UCP.

The backup can only be used to create a new DTR, using the `docker/dtr restore` command.

Identity Management

Accessing resources (images, containers, volumes, networks etc) and functionality within the components of Docker EE (UCP & DTR) require at a minimum, an account and a corresponding password to be accessed. Accounts within Docker EE are identities stored within an internal database, but the source of creating those accounts and the associated access control can be manual (managed or internal) or external through a connection to a directory server (LDAP) or Active Directory (AD). Managing the authorization for these accounts is an extension of coarse and fine grained permissions that are described in the sections below.

Managed (Internal) Authentication

Managed mode of authentication and authorization is the default mode in Docker EE Standard and Advanced. In this mode, accounts are directly created using the Docker EE API. User accounts can be created manually by accessing the User Management —> Create User form in the UCP UI. Accounts can also be created and managed in an automated fashion by making HTTP requests to the authentication and authorization RESTful service known as eNZi.

User management using the "Managed" mode is recommended only for demo purposes or where the number of users needing to access Docker EE is very small.

Pros:

- Easy and quick to setup
- Simple to troubleshoot
- Appropriate for a small set of users with static roles
- Managed without leaving the UCP interface

Cons:

- User Account management gets cumbersome for larger numbers or when roles need to be managed for several applications.
- All lifecycle changes such as adding / removing permissions of users need to be accomplished manually user by user.
- Users must be deleted manually, meaning access may not get cleaned up quickly, making the system less secure.
- Sophisticated setups of integrating application creation and deployment through LDAP or external systems is not possible.

LDAP / AD Integration

The LDAP method of user account authentication can be turned on to manage user access. As the name suggests, this mode enables automatically synchronization of user accounts from a directory server such as Active Directory or OpenLDAP.

This method is particularly applicable for enterprise use cases where organizations have a large set of users, typically maintained in a centralized identity store that manages both authentication and authorization. Most of these stores are based on a directory server such as Microsoft's Active Directory or a system that supports the LDAP protocol. Additionally, such enterprises already have mature processes for on-boarding, off-boarding, and managing the lifecycle of changes to user and system accounts. All these can be leveraged to provide a seamless and efficient process of access control within Docker EE.

Pros:

- Ability to leverage already established access control processes to grant and revoke permissions.
- Ability to continue managing users and permissions from a centralized system based on LDAP.
- Increased security due to self-cleaning nature of this mode, where non-existent LDAP users are automatically removed from Docker EE on the next sync.
- Ability to configure complex upstream systems such as flat files, database tables using an LDAP proxy, and the automatic time-based de-provisioning of access through AD/LDAP groups.

Cons:

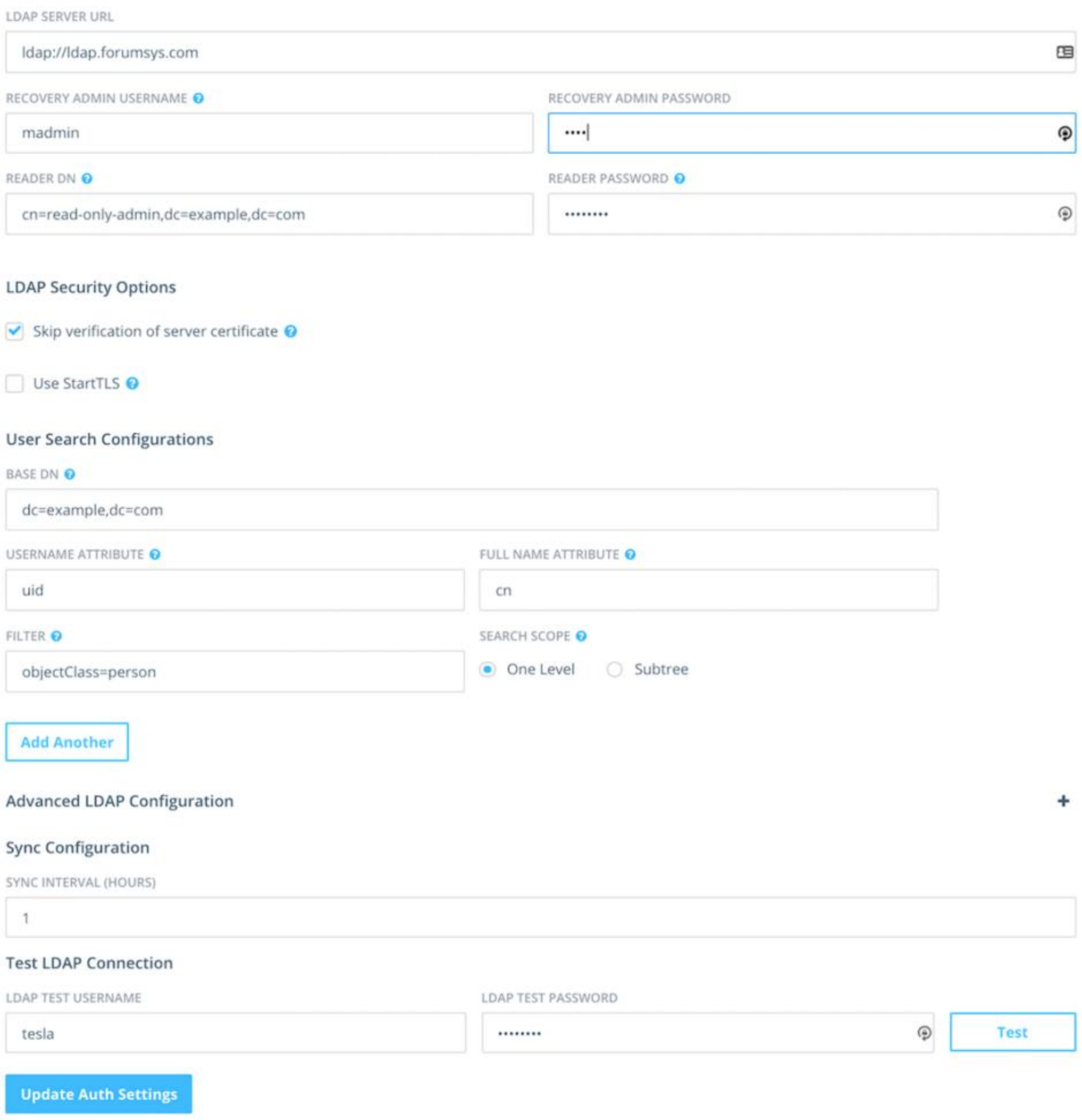
- Increased complexity compared to Managed mode
- Higher admin requirements since knowledge on an external system (LDAP) is needed
- Greater time needed to troubleshoot issues with an extra components (LDAP) in the mix

- Unexpected changes to Docker EE due to changes made to upstream LDAP/AD systems, which can have unexpected effects on Docker EE

A recommended best practice is to use group membership to control access of users accounts to resources. Ideally, the management of such group membership is achieved through a centralized Identity Management or a Role Based Access Control system. This provides a standard, flexible, and scalable model to control the authentication and authorization rules within Docker EE through a centralized directory server. Through the Identity Management system, this directory server is kept in sync with user on-boarding, off-boarding, and changes in roles and responsibilities.

To change the mode of authentication, use the form at Admin Settings —> Auth in the UCP UI. In this form, change the METHOD field under Authentication from the default value of Managed to LDAP.

Accounts that are discovered and subsequently synced from the directory server can be automatically assigned default permissions. It is recommended to leave the option for default permission to No Access Any necessary additional access can be granted based on group memberships.



LDAP SERVER URL: ldap://ldap.forumsys.com

RECOVERY ADMIN USERNAME: madmin

RECOVERY ADMIN PASSWORD: ••••

READER DN: cn=read-only-admin,dc=example,dc=com

READER PASSWORD: ••••••••

LDAP Security Options

[✓] Skip verification of server certificate

[ ] Use StartTLS

User Search Configurations

BASE DN: dc=example,dc=com

USERNAME ATTRIBUTE: uid

FULL NAME ATTRIBUTE: cn

FILTER: objectClass=person

SEARCH SCOPE: ● One Level ○ Subtree

Advanced LDAP Configuration

Sync Configuration

SYNC INTERVAL (HOURS): 1

Test LDAP Connection

LDAP TEST USERNAME: tesla

LDAP TEST PASSWORD: ••••••••

For details about the LDAP configuration options, read the Integrate with LDAP documentation (https://docs.docker.com/datacenter/ucp/2.0/guides/configuration/integrate-with-ldap/).

This section highlights important configuration options to consider when setting up LDAP authentication.

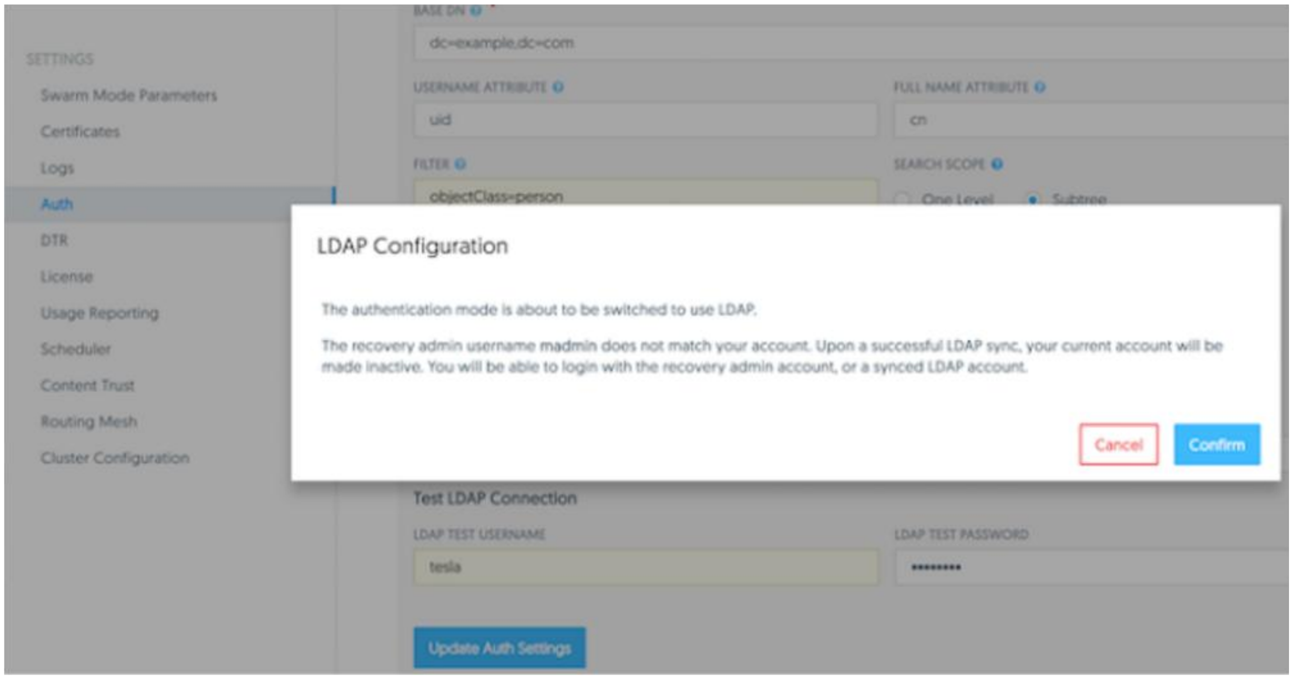
In LDAP auth mode, when accounts are discovered and subsequently imported, all previously created accounts in a Managed mode including the admin accounts are deleted. A new admin account is created when authentication is switched to LDAP. This admin account is a local account and is intended to be useable independent of the LDAP settings (for example, in a situation when there is a loss of connectivity to the directory server). The details of this admin account can be set during the sync using RECOVERY ADMIN USERNAME and RECOVERY ADMIN PASSWORD.

A user account on the directory server has to be configured to discover and import accounts from the directory server. This user account need not be a very powerful account. In fact, it is recommended that it is a read-only account that can view the necessary organizationalUnits (`ou`) and query for group memberships. The details for this account are configured using the fields READER DN and READER PASSWORD. The READER DN must be in the distinguishedName format.

Use secure LDAP if at all possible.

Use the Test LDAP Connection section to make sure you can connect before switching to LDAP authentication.

Once the form is filled out and the test connection succeeds, the sync button provides an option to run a sync immediately without waiting for the next interval. Doing so initiates an LDAP connection and runs the filters to import the users. Any previously logged in user is logged out due to the disablement of all pre-existing accounts.



BASE DN: dc=example,dc=com

USERNAME ATTRIBUTE: uid

FULL NAME ATTRIBUTE: cn

FILTER: objectClass=person

SEARCH SCOPE: One Level [✓ Subtree]

LDAP Configuration

The authentication mode is about to be switched to use LDAP.

The recovery admin username madmin does not match your account. Upon a successful LDAP sync, your current account will be made inactive. You will be able to login with the recovery admin account, or a synced LDAP account.

Test LDAP Connection

LDAP TEST USERNAME: tesla

LDAP TEST PASSWORD: ••••••••

After the sync is complete, it should be possible to log in using the recovery account and password. Be sure to test this recovery account. It should also be possible to log in using a sync-ed LDAP/AD account (the only supported attributes are `uid` and `sAMAccountName`) and its corresponding LDAP/AD password. The sync-ed account should be in good standing within the LDAP/AD system for the login to be successful within Docker EE.

The progress / status of the sync and any errors that occur can be viewed and analyzed on the controllers by running:

```
docker logs ucp-controller
```

Teams

User accounts that exist within Docker EE, either through an LDAP sync or manually managed, can be organized into teams. Each team created can be assigned customized permissions and managed as a group of users as opposed to granting permissions to each user account on an individual basis.

Teams can be created in the UCP UI using the TEAMS +Create interface under User Management. The same can be accomplished by using the eNZi API that is described in a later section. Members can be added to the team one-by-one. An alternative method of adding members to a team can be employed if using the LDAP Auth mode. This is based on an automatic sync of discovered accounts from the directory server that was configured to enable the LDAP Auth mode. Finer filters can be applied here which determine which discovered accounts are placed into which teams. A team can have multiple users, and a user can be a member of zero to multiple teams.

RBAC and Managing Team Level Access to Resources

Permissions to resources through UCP are managed through default access levels. The default access permissions are `No Access` , `View Only` , `Restricted Control` and `Full Control` . Description about these permissions and how they relate to each other are detailed in the 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). These permissions can be further extended by adding the user accounts to specific teams. In other words, the default permissions assigned directly to a user provide a coarse-grained access control. The user should then be added to teams to attain higher granularity in what a user account can do on specific containers, images, and other resources. The teams are assigned attributes in the form of labels that provide additional flexibility through a fine-grained access control. In some ways this mimics a traditional Role Based Access Control (RBAC), where the teams are the roles assigned to user accounts, but there are specks of the finer Attribute Based Access Control (ABAC) through the assignment of one of more labels to each team, which ultimately define the level of access of the account to UCP resources.

While teams can be associated with as many labels as necessary, the resources within UCP like `services` , `images` , `volumes` , `networks` , and `secrets` can also be associated with any one of these labels to tie the two together. It should be noted that any labels assigned to a service are automatically inherited by the tasks which currently are container workloads. So this behaves in just the same way as assigning meta information in the form of labels to containers (or any other resource for that matter). Internally, the resources are assigned a label with the key `com.docker.ucp.access.label` with the value as specified.

It would be easier to understand with a concrete example as described in the following section.

Strategy for Using LDAP Filters

This section demonstrates a typical use case that uses the principles of "least privilege / permission" as well as "separation of duties."

Suppose you have a simple application called `www`, which is a web server based on the `nginx` official image. Say there are three teams that need access to this application — `developers`, `testers`, and `operations`. Typically, `testers` need `View Only` access and nothing further, while the `operations` team usually needs `Full Control`. The `developers` team needs access to troubleshoot, restart, and control the lifecycle of the application but should be forbidden from any other activity involving the need to access the host file systems, or starting up privileged containers. This type of special access is termed `Restricted Control`.

The users needing to access UCP are all sourced from the corporate Directory Server system. These users are the admin users needed to manage the Docker EE infrastructure as well as all members of each of the teams configured within UCP. Also assume that the total universe of users needing access to Docker EE (includes admins, developers, testers and operations) is a subset of the gamut of users within the Directory Server.

A recommended strategy to use when organizing users is to create an overarching membership group that identifies all users of Docker EE irrespective of which team they are a part of. Let us call this group `Docker_Users`. No user should be made a member of this group directly. Instead, the `Docker_Users` group should have other groups in it and only groups as its members. Per our example, let us call these groups `dev`, `test`, and `ops`. In our example, these groups are part of what is known as a *nested group* structure within the directory server. Nested groups allow the inheritance of permissions from one group to each of its sub-groups.

> NOTE: Some directory servers do not support the feature of nested groups or even the `memberOf` attribute by default. If so, then they would need to be enabled. If the choice of directory server does not support these features at all, then alternate means of organizing users and querying them should be used. Microsoft Active Directory supports both these features out of the box.

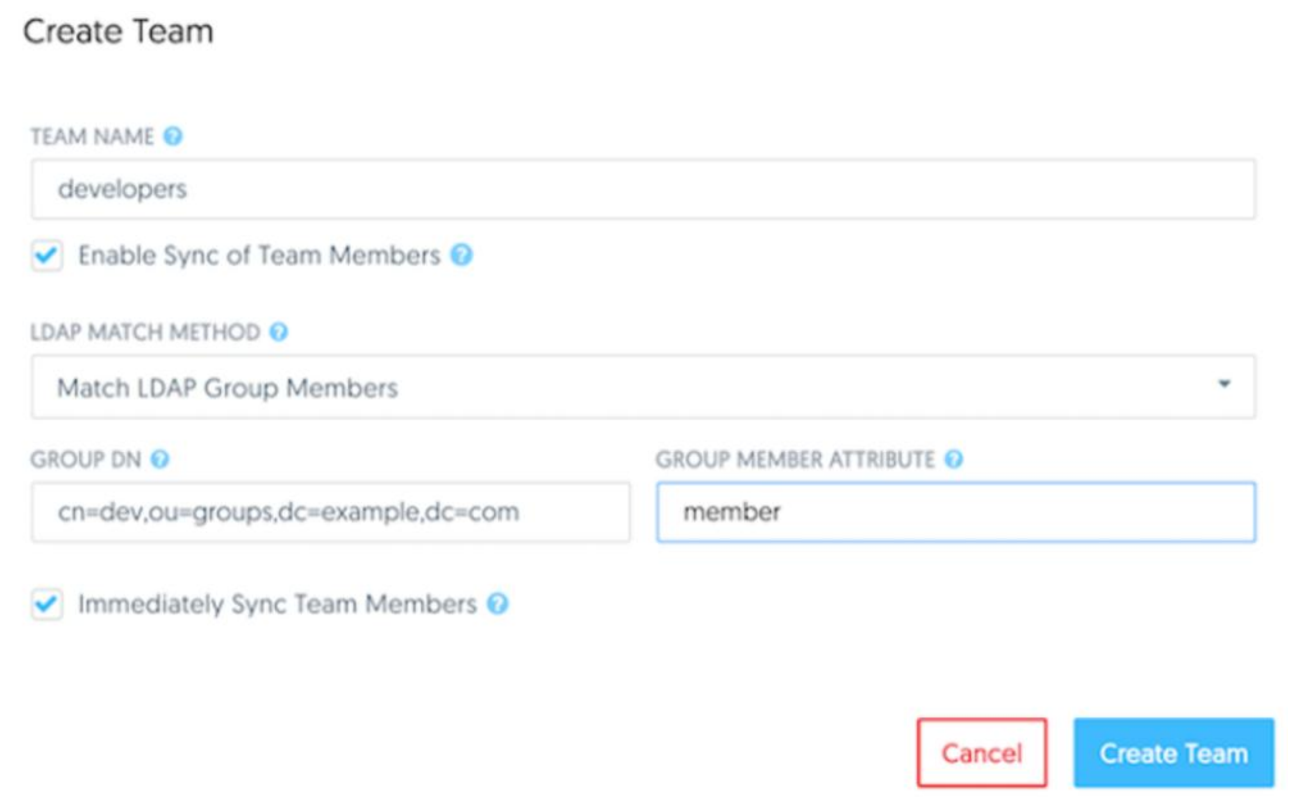
User accounts should be added as members of these sub-groups in the directory server. This should not impact any existing layout in the organization units or pre-existing group membership for these users.

Within UCP, the first step is to enable the LDAP Auth mode. Under User Search Configurations, for the FILTER field, the value based on our example above should be `(&(objectClass=person)(memberOf=cn=Docker_Users,ou=groups,dc=example,dc=com))`.

Performing a sync with this setting discovers and import all users that are members of the group

`Docker_Users` (through the nested sub-groups).

Next, using the TEAMS +Create link under User Management, the individual teams can be created. An example of creating the `developers` team is shown below. The value of the Group DN field reflects the sub-group created under `Docker_Users` called `dev` .


Create Team

TEAM NAME: developers

[✓] Enable Sync of Team Members

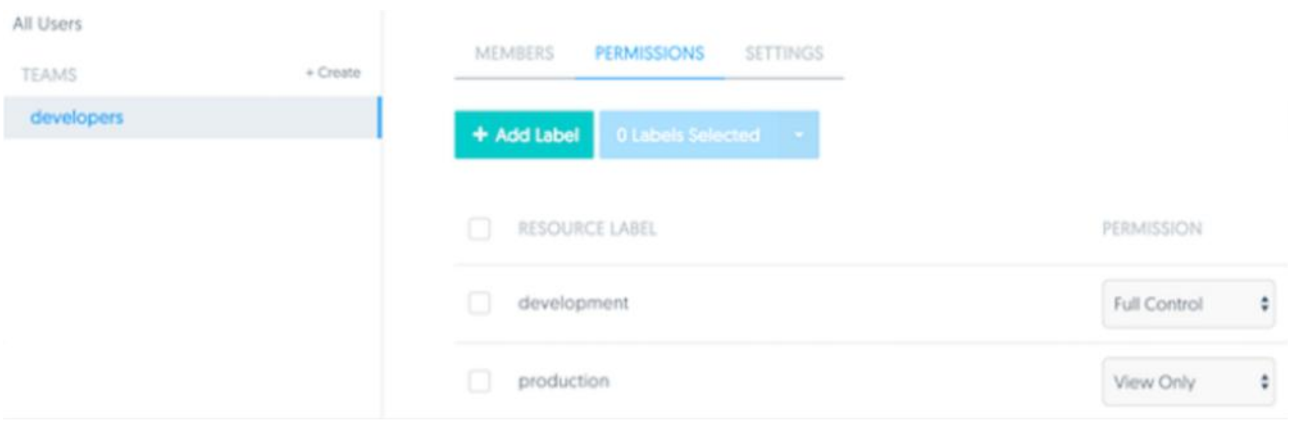
LDAP MATCH METHOD: Match LDAP Group Members

GROUP DN: cn=dev,ou=groups,dc=example,dc=com

GROUP MEMBER ATTRIBUTE: member

[✓] Immediately Sync Team Members

After the team is created, bring up the Add Label form by clicking on the PERMISSIONS link. Here, as many permissions as needed can be added. In the image below, two labels are added to denote that members of the `developers` team have `View Only` access to any resource labeled as `production` . An additional label grants the members of the `developers` team `Full Control` on resources labeled as `development` .



RESOURCE LABEL	PERMISSION
development	Full Control
production	View Only

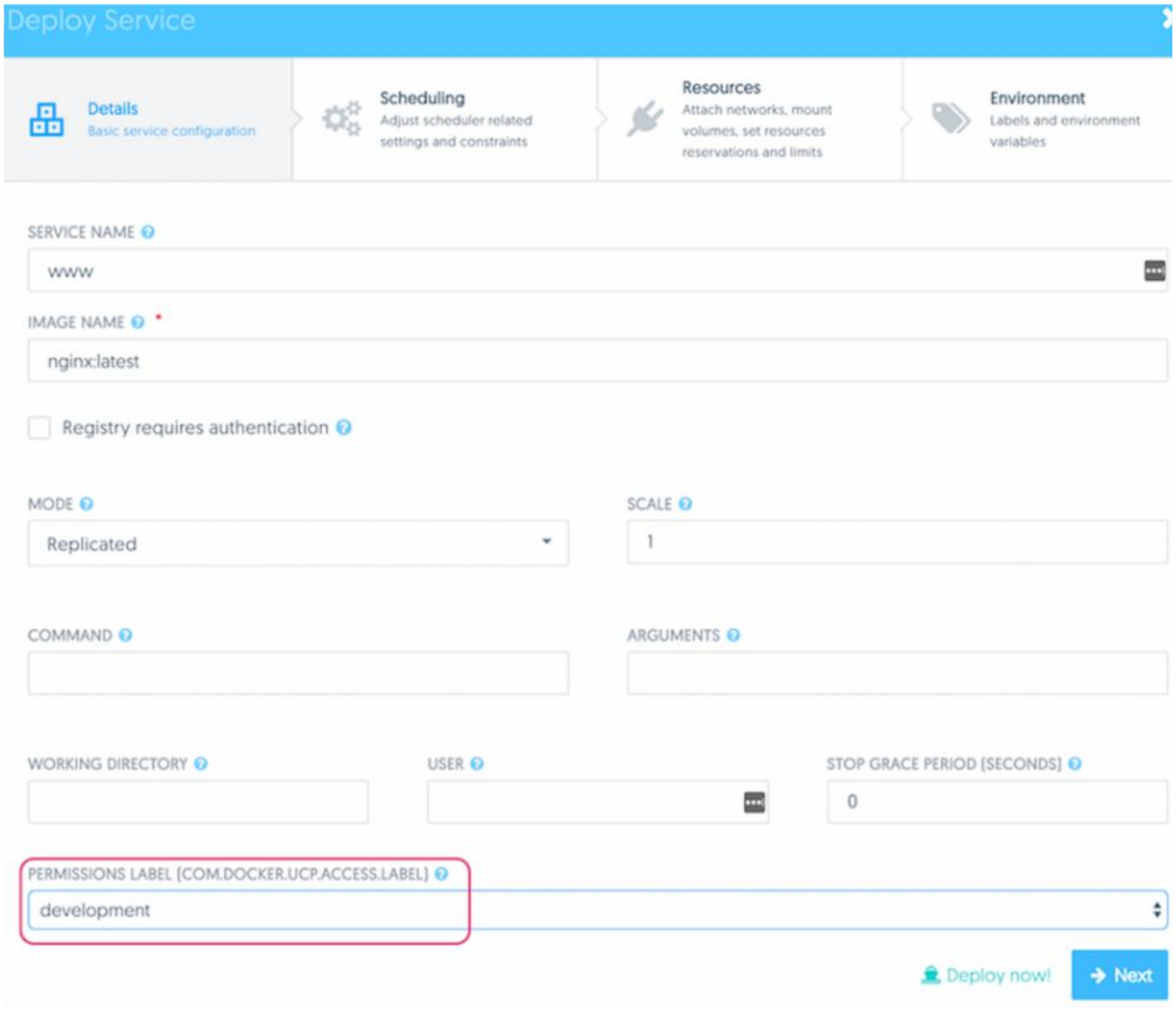
Similarly, the `testers` and `operations` teams are created and assigned permissions per the table below:

Team	Group	Permissions
`testers`	`cn=test,ou=groups,dc=example,dc=com`	`production`: `No Access`
`development`: `View Only`		
`operations`	`cn=ops,ou=groups,dc=example,dc=com`	`production`: `Full Control`
`development`: `Full Control`		

Now, you can now create the service `www` and assign it the appropriate `access label` to allow only specific users access to the resource using a command like below:		

```
docker service create --name www -p 80:80 --label com.docker.ucp.access.label=developm
ent nginx
```

The same can be done in the UCP UI while creating a service by selecting an appropriate label (`development` in this example) from the PERMISSIONS LABEL dropdown as shown below. The label is applied to the service for enforcing role-based access control.


Deploy Service
Details
Basic service configuration
Scheduling
Adjust scheduler related settings and constraints
Resources
Attach networks, mount volumes, set resources reservations and limits
Environment
Labels and environment variables
Deploy now!
Next

SERVICE NAME: www

IMAGE NAME *: nginx:latest

Registry requires authentication

MODE: Replicated

SCALE: 1

COMMAND:

ARGUMENTS:

WORKING DIRECTORY:

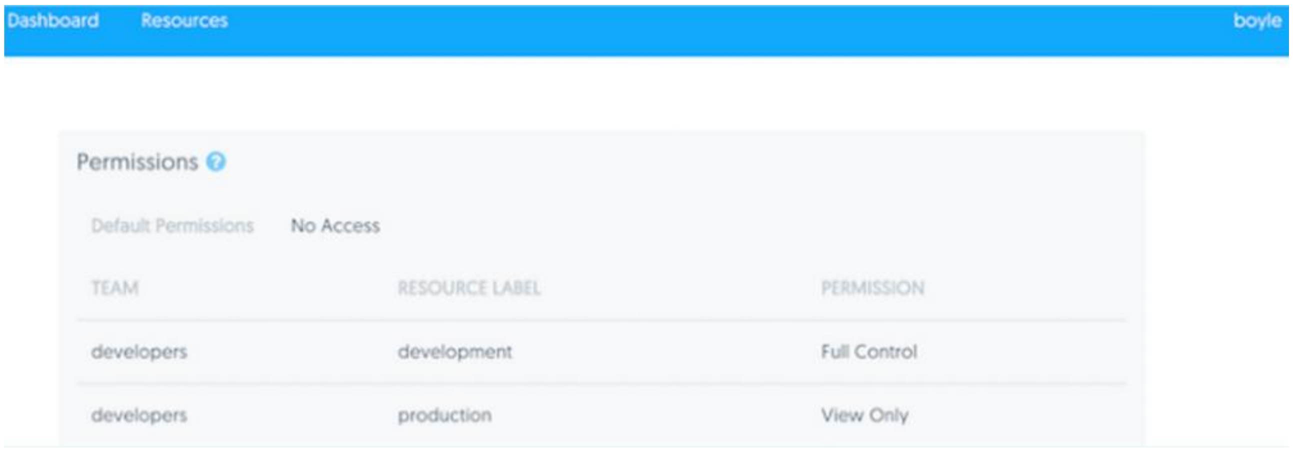
USER:

STOP GRACE PERIOD (SECONDS): 0

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

Only those labels that the logged in user has access to shows up in the dropdown. An admin user or a user assigned `Full Control` permission is shown all available labels regardless of the labels assigned to the team of which the user is a member of.

Now, create another service `www-prod` similarly based on the `nginx` image but with the Permission Label of `production` applied to it while logged in as a user `boyle` who is a member of the developers team. His permissions look as follows:



Permissions

Default Permissions No Access

TEAM	RESOURCE LABEL	PERMISSION
developers	development	Full Control
developers	production	View Only

The account `boyle` can see both services, `www` and `www-prod`, but only the `www` service allows edits and other control because the permission label of `development` matches with the team's resource label of `development` granted with `Full Control` permissions. The `www-prod` service forbids any edits using the account `boyle` because it has a permission label of `production`, which has been marked as `View Only` in the team `developers'` resource label. For example, if the account `boyle` was used to scale up the number of replicas of each service, from 1 to 2 (for instance), an error of `Access Denied` would be thrown for the `www-prod` service, while the same would work for `www` service.

Further, as evident in the profile image, `boyle` has `Default Permissions` of `No Access`. This prevents the `boyle` account from accessing any other resource unless the permission label and team resource label match and provide explicit permissions that override the `No Access` default permission.

Finally, if and when it becomes necessary to terminate all access for any user account, removing the group membership of the account from just the one group `Docker_Users` would remove all access for the user. Due to the nature of how nested groups work, all additional access within Docker is automatically cleaned — the user account is removed from any and all team memberships at the time of the next sync without need for manual intervention or additional steps. This step can be integrated into a standard on-boarding / off-boarding automated provisioning step within a corporate Identity Management system.

Authentication API (eNZi)

The AuthN API or eNZi (as it is known internally and pronounced N-Z) is a centralized authentication and authorization service and framework for Docker EE. This API is completely integrated and configured into Docker EE and works seamlessly with UCP as well as DTR. This is the component and service under the hood that manages accounts, teams and organizations, user sessions, permissions and access control through labels, Single-Sign-On (Web SSO) through OpenID Connect, and synchronization of account details from an external LDAP-based system into Docker EE.

For regular day-to-day activities, users and operators need not be concerned with the AuthN API and how it works, however its features can be leveraged to automate many common functions and/or bypass the UCP UI altogether to manage and manipulate the data directly.

The RESTFul AuthN API endpoints are documented in the AuthN API (https://docs.docker.com/apidocs/v2.0.1/#!/accounts/func1).

Interaction with AuthN can be accomplished in two ways: via the exposed RESTful AuthN API over HTTP or via the `enzi` command.

For example, the command below uses `curl` and `jq` to fetch all user accounts in Docker EE via the AuthN API over HTTP:

```
curl --silent --insecure --header "Authorization: Bearer $(curl --silent --insecure \
     --data '{"username":"<admin-username>","password":"<admin-password>"}' \
    https://<UCP-domain-name>/auth/login | jq --raw-output .auth_token)" \
    https://<UCP-domain-name>/enzi/v0/accounts | jq .
```

The AuthN service can also be invoked on the CLI on a UCP controller. To connect into it, run the following on a UCP controller:

```
docker exec -it ucp-auth-api sh
```

At the resulting prompt (`#`), type the `enzi` command with a sub-command such as the one below to list the database table status:

```
enzi db-status
```

> Note: Refer to Recovering the Admin Password for Docker EE Standard and Advanced (https://success.docker.com/api/asset/.%2Frefarch%2Fdocker-ee-best-practices-17-03%2F%2Farticle%2FRecovering_the_Admin_Password_for_Docker_EE_Standard_and_Advanced) for a detailed example.

What is Docker (https://www.docker.com/what-docker)

What is a Container (https://www.docker.com/what-container)

Use Cases (https://www.docker.com/use-cases)

Customers (https://www.docker.com/customers)

For Government (https://www.docker.com/industry-government)

For IT Pros (https://www.docker.com/itpro)

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)

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

Pricing (https://www.docker.com/pricing)

Community Edition (https://www.docker.com/community-edition)

Enterprise Edition (https://www.docker.com/enterprise-edition)

Docker Datacenter (https://www.docker.com/enterprise-edition#container_management)

Docker Cloud (https://cloud.docker.com/)

Docker Store (https://store.docker.com/)

Get Docker (https://www.docker.com/get-docker)

Docker for Mac (https://www.docker.com/docker-mac)

Docker for Windows(PC) (https://www.docker.com/docker-windows)

Docker for AWS (https://www.docker.com/docker-aws)

Docker for Azure (https://www.docker.com/docker-azure)

Docker for Windows Server (https://www.docker.com/docker-windows-server)

Docker for Debian (https://www.docker.com/docker-debian)

Docker for Fedora® (https://www.docker.com/docker-fedora)

Docker for Oracle Linux (https://www.docker.com/docker-oracle-linux)

Docker for RHEL (https://www.docker.com/docker-red-hat-enterprise-linux-rhel)

Docker for SLES (https://www.docker.com/docker-suse-linux-enterprise-server-sles)

Docker for Ubuntu (https://www.docker.com/docker-ubuntu)

Documentation (https://docs.docker.com/)

Blog (https://blog.docker.com/)

RSS Feed (https://blog.docker.com/feed/)

Training (https://training.docker.com/)

Knowledge Base (https://success.docker.com/kbase)

Resources (https://www.docker.com/products/resources)

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

Open Source (https://www.docker.com/technologies/overview)

Events (https://www.docker.com/community/events)

Forums (https://forums.docker.com/)

Docker Captains (https://www.docker.com/community/docker-captains)

Scholarships (https://www.docker.com/community-partnerships)

Community News (https://blog.docker.com/curated/)

Status (http://status.docker.com/) Security (https://www.docker.com/docker-security)

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

(http: (http: (https (https (https (https (http: (http:

Docker overview

Estimated reading time: 10 minutes

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

The Docker platform

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!

Docker provides tooling and a platform to manage the lifecycle of your containers:

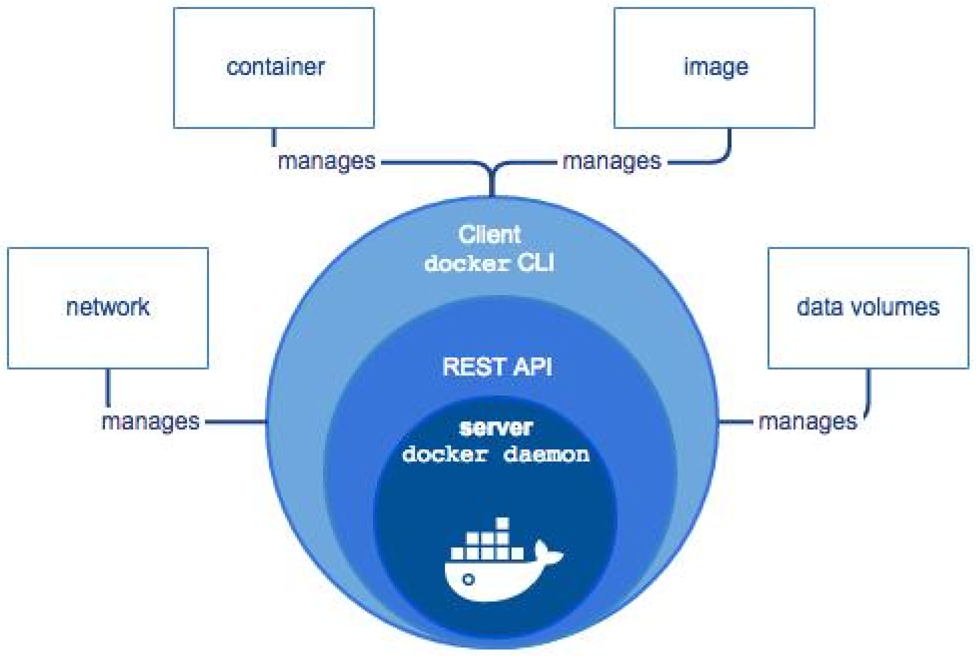
- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

Docker Engine

Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a daemon process (the `dockerd` command).

- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the `docker` command).



The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.

The daemon creates and manages Docker *objects*, such as images, containers, networks, and volumes.

> Note: Docker is licensed under the open source Apache 2.0 license.

For more details, see Docker Architecture (/engine/docker-overview/#docker-architecture) below.

What can I use Docker for?

Fast, consistent delivery of your applications

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

- Your developers write code locally and share their work with their colleagues using Docker containers.
- They use Docker to push their applications into a test environment and execute automated and manual tests.
- When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
- When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Responsive deployment and scaling

Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.

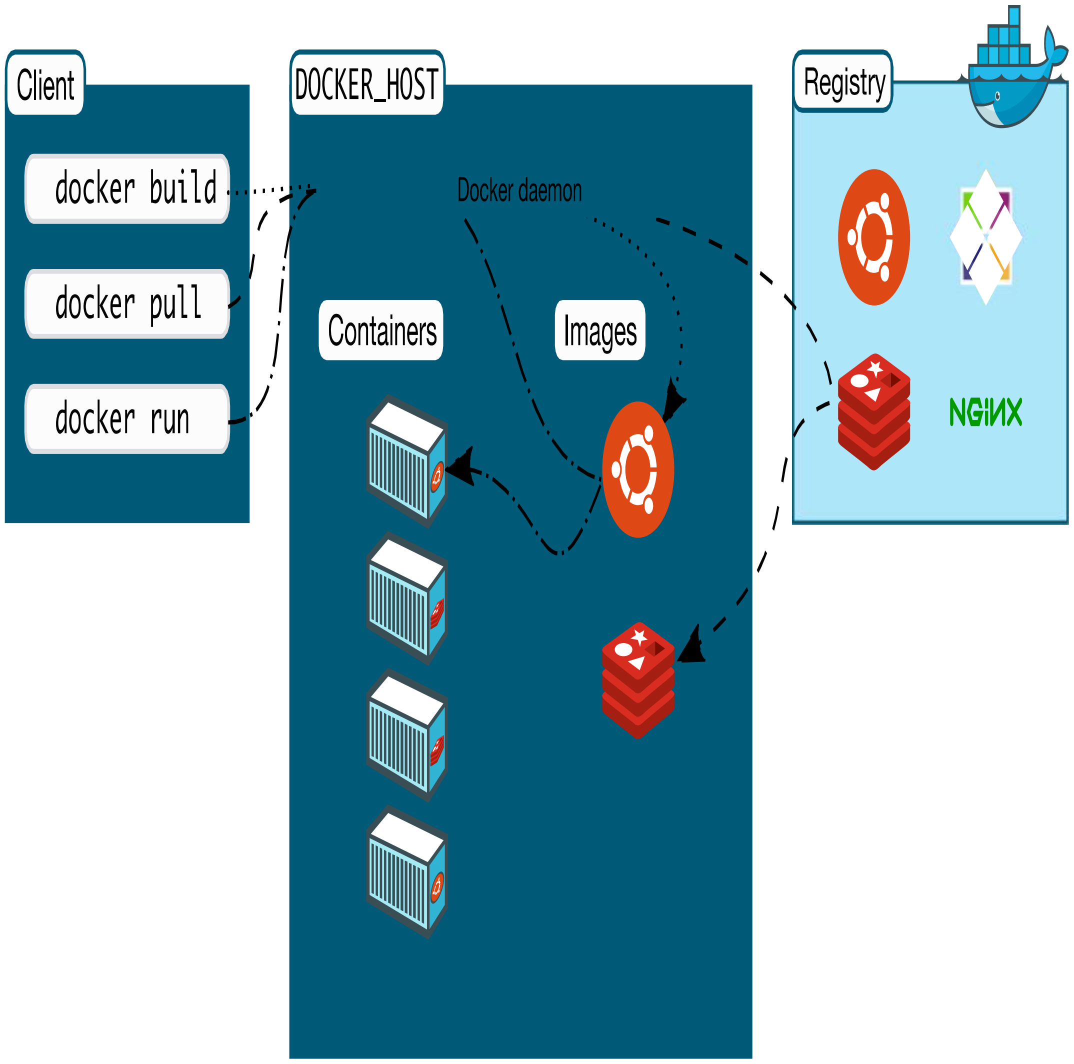
Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

Running more workloads on the same hardware

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

Docker architecture

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



The Docker daemon

The Docker daemon (`dockerd`) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (`docker`) is the primary way that many Docker users interact with Docker. When you use commands such as `docker run` , the client sends these commands to `dockerd` , which carries them out. The `docker` command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

When you use the `docker pull` or `docker run` commands, the required images are pulled from your configured registry. When you use the `docker push` command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

IMAGES

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the `ubuntu` image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

CONTAINERS

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Example `docker run` command

The following command runs an `ubuntu` container, attaches interactively to your local command-line session, and runs `/bin/bash` .

```
$ docker run -i -t ubuntu /bin/bash
```

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you do not have the `ubuntu` image locally, Docker pulls it from your configured registry, as though you had run `docker pull ubuntu` manually.
2. Docker creates a new container, as though you had run a `docker container create` command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.
4. Docker creates a network interface to connect the container to the default network, since you did not specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine's network connection.
5. Docker starts the container and executes `/bin/bash` . Because the container is running interactively and attached to your terminal (due to the `-i` and `-t` flags), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type `exit` to terminate the `/bin/bash` command, the container stops but is not removed. You can start it again or remove it.

SERVICES

Services allow you to scale containers across multiple Docker daemons, which all work together as a *swarm* with multiple *managers* and *workers*. Each member of a swarm is a Docker daemon, and the daemons all communicate using the

Docker API. A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes. To the consumer, the Docker service appears to be a single application. Docker Engine supports swarm mode in Docker 1.12 and higher.

The underlying technology

Docker is written in Go (https://golang.org/) and takes advantage of several features of the Linux kernel to deliver its functionality.

Namespaces

Docker uses a technology called `namespaces` to provide the isolated workspace called the *container*. When you run a container, Docker creates a set of *namespaces* for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

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` . In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones.

Next steps

- Read about installing Docker (https://docs.docker.com/engine/installation/#installation).
- Get hands-on experience with the Getting started with Docker (https://docs.docker.com/engine/getstarted/) tutorial.
- Check out examples and deep dive topics in the Docker Engine user guide (https://docs.docker.com/engine/userguide/).

docker (https://docs.docker.com/glossary/?term=docker), introduction (https://docs.docker.com/glossary/?term=introduction), documentation (https://docs.docker.com/glossary/?term=documentation), about (https://docs.docker.com/glossary/?term=about), technology (https://docs.docker.com/glossary/?term=technology), understanding (https://docs.docker.com/glossary/?term=understanding)

Protect the Docker daemon socket

Estimated reading time: 7 minutes

By default, Docker runs through a non-networked UNIX socket. It can also optionally communicate using an HTTP socket.

If you need Docker to be reachable through the network in a safe manner, you can enable TLS by specifying the `tlsverify` flag and pointing Docker's `tlscacert` flag to a trusted CA certificate.

In the daemon mode, it only allows connections from clients authenticated by a certificate signed by that CA. In the client mode, it only connects to servers with a certificate signed by that CA.

> Advanced topic
>
> Using TLS and managing a CA is an advanced topic. Please familiarize yourself with OpenSSL, x509 and TLS before using it in production.

Create a CA, server and client keys with OpenSSL

> Note: replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon's host.

First, on the Docker daemon's host machine, generate CA private and public keys:

```
$ openssl genrsa -aes256 -out ca-key.pem 4096
Generating RSA private key, 4096 bit long modulus
....................................................................
....................................................................
....................................................++
........++
e is 65537 (0x10001)
Enter pass phrase for ca-key.pem:
Verifying - Enter pass phrase for ca-key.pem:

$ openssl req -new -x509 -days 365 -key ca-key.pem -sha256 -out ca.p
em
Enter pass phrase for ca-key.pem:
You are about to be asked to enter information that will be incorpor
ated
into your certificate request.
What you are about to enter is what is called a Distinguished Name o
r a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:Queensland
Locality Name (eg, city) []:Brisbane
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Docker In
c
Organizational Unit Name (eg, section) []:Sales
Common Name (e.g. server FQDN or YOUR name) []:$HOST
Email Address []:Sven@home.org.au
```

Now that you have a CA, you can create a server key and certificate signing request (CSR). Make sure that “Common Name” matches the hostname you use to connect to Docker:

> Note: replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon’s host.

```
$ openssl genrsa -out server-key.pem 4096
Generating RSA private key, 4096 bit long modulus
.....................................................................
.++
.....................................................................
.............................++
e is 65537 (0x10001)

$ openssl req -subj "/CN=$HOST" -sha256 -new -key server-key.pem -ou
t server.csr
```

Next, we’re going to sign the public key with our CA:

Since TLS connections can be made through IP address as well as DNS name, the IP addresses need to be specified when creating the certificate. For example, to allow connections using `10.10.10.20` and `127.0.0.1` :

```
$ echo subjectAltName = DNS:$HOST,IP:10.10.10.20,IP:127.0.0.1 >> ext
file.cnf
```

Set the Docker daemon key’s extended usage attributes to be used only for server authentication:

```
$ echo extendedKeyUsage = serverAuth >> extfile.cnf
```

Now, generate the signed certificate:

```
$ openssl x509 -req -days 365 -sha256 -in server.csr -CA ca.pem -CAk
ey ca-key.pem \
  -CAcreateserial -out server-cert.pem -extfile extfile.cnf
Signature ok
subject=/CN=your.host.com
Getting CA Private Key
Enter pass phrase for ca-key.pem:
```

Authorization plugins (https://docs.docker.com/engine/extend/plugins_authorization) offer more fine-grained control to supplement authentication from mutual TLS. In addition to other information described in the above document, authorization plugins running on a Docker daemon receive the certificate information for connecting Docker clients.

For client authentication, create a client key and certificate signing request:

> Note: for simplicity of the next couple of steps, you may perform this step on the Docker daemon’s host machine as well.

```
$ openssl genrsa -out key.pem 4096
Generating RSA private key, 4096 bit long modulus
.........................................................++
................++
e is 65537 (0x10001)

$ openssl req -subj '/CN=client' -new -key key.pem -out client.csr
```

To make the key suitable for client authentication, create a new extensions config file:

```
$ echo extendedKeyUsage = clientAuth > extfile-client.cnf
```

Now, generate the signed certificate:

```
$ openssl x509 -req -days 365 -sha256 -in client.csr -CA ca.pem -CAk
ey ca-key.pem \
  -CAcreateserial -out cert.pem -extfile extfile-client.cnf
Signature ok
subject=/CN=client
Getting CA Private Key
Enter pass phrase for ca-key.pem:
```

After generating `cert.pem` and `server-cert.pem` you can safely remove the two certificate signing requests and extensions config files:

```
$ rm -v client.csr server.csr extfile.cnf extfile-client.cnf
```

With a default `umask` of 022, your secret keys are *world-readable* and writable for you and your group.

To protect your keys from accidental damage, remove their write permissions. To make them only readable by you, change file modes as follows:

```
$ chmod -v 0400 ca-key.pem key.pem server-key.pem
```

Certificates can be world-readable, but you might want to remove write access to prevent accidental damage:

```
$ chmod -v 0444 ca.pem server-cert.pem cert.pem
```

Now you can make the Docker daemon only accept connections from clients providing a certificate trusted by your CA:

```
$ dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem -
-tlskey=server-key.pem \
  -H=0.0.0.0:2376
```

To connect to Docker and validate its certificate, provide your client keys, certificates and trusted CA:

> ✓ Run it on the client machine
>
> This step should be run on your Docker client machine. As such, you need to copy your CA certificate, your server certificate, and your client certificate to that machine.

> Note: replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon's host.

```
$ docker --tlsverify --tlscacert=ca.pem --tlscert=cert.pem --tlskey=
key.pem \
  -H=$HOST:2376 version
```

Note: Docker over TLS should run on TCP port 2376.

Warning: As shown in the example above, you don't need to run the `docker` client with `sudo` or the `docker` group when you use certificate authentication. That means anyone with the keys can give any instructions to your Docker daemon, giving them root access to the machine hosting the daemon. Guard these keys as you would a root password!

Secure by default

If you want to secure your Docker client connections by default, you can move the files to the `.docker` directory in your home directory --- and set the `DOCKER_HOST` and `DOCKER_TLS_VERIFY` variables as well (instead of passing `-H=tcp://$HOST:2376` and `--tlsverify` on every call).

```
$ mkdir -pv ~/.docker
$ cp -v {ca,cert,key}.pem ~/.docker

$ export DOCKER_HOST=tcp://$HOST:2376 DOCKER_TLS_VERIFY=1
```

Docker now connects securely by default:

```
$ docker ps
```

Other modes

If you don't want to have complete two-way authentication, you can run Docker in various other modes by mixing the flags.

Daemon modes

- `tlsverify` , `tlscacert` , `tlscert` , `tlskey` set: Authenticate clients
- `tls` , `tlscert` , `tlskey` : Do not authenticate clients

Client modes

- `tls` : Authenticate server based on public/default CA pool

- `tlsverify` , `tlscacert` : Authenticate server based on given CA
- `tls` , `tlscert` , `tlskey` : Authenticate with client certificate, do not authenticate server based on given CA
- `tlsverify` , `tlscacert` , `tlscert` , `tlskey` : Authenticate with client certificate and authenticate server based on given CA

If found, the client sends its client certificate, so you just need to drop your keys into `~/.docker/{ca,cert,key}.pem` . Alternatively, if you want to store your keys in another location, you can specify that location using the environment variable `DOCKER_CERT_PATH` .

```
$ export DOCKER_CERT_PATH=~/.docker/zone1/
$ docker --tlsverify ps
```

Connecting to the secure Docker port using `curl`

To use `curl` to make test API requests, you need to use three extra command line flags:

```
$ curl https://$HOST:2376/images/json \
  --cert ~/.docker/cert.pem \
  --key ~/.docker/key.pem \
  --cacert ~/.docker/ca.pem
```

Related information

- Using certificates for repository client verification (https://docs.docker.com/engine/security/certificates/)
- Use trusted images (https://docs.docker.com/engine/security/trust/)

docker (https://docs.docker.com/glossary/?term=docker), docs (https://docs.docker.com/glossary/?term=docs), article (https://docs.docker.com/glossary/?term=article), example (https://docs.docker.com/glossary/?term=example), https (https://docs.docker.com/glossary/?term=https), daemon (https://docs.docker.com/glossary/?term=daemon), tls (https://docs.docker.com/glossary/?term=tls), ca (https://docs.docker.com/glossary/?term=ca), certificate (https://docs.docker.com/glossary/?term=certificate)

Verify repository client with certificates

Estimated reading time: 2 minutes

In Running Docker with HTTPS (https://docs.docker.com/engine/security/https/), you learned that, by default, Docker runs via a non-networked Unix socket and TLS must be enabled in order to have the Docker client and the daemon communicate securely over HTTPS. TLS ensures authenticity of the registry endpoint and that traffic to/from registry is encrypted.

This article demonstrates how to ensure the traffic between the Docker registry server and the Docker daemon (a client of the registry server) is encrypted and properly authenticated using *certificate-based client-server authentication*.

We show you how to install a Certificate Authority (CA) root certificate for the registry and how to set the client TLS certificate for verification.

Understanding the configuration

A custom certificate is configured by creating a directory under `/etc/docker/certs.d` using the same name as the registry's hostname, such as `localhost` . All `*.crt` files are added to this directory as CA roots.

> Note: As of Docker 1.13, on Linux any root certificates authorities are merged with the system defaults, including as the host's root CA set. On prior versions of Docker, and on Docker Enterprise Edition for Windows Server, the system default certificates are only used when no custom root certificates are configured.

The presence of one or more `<filename>.key/cert` pairs indicates to Docker that there are custom certificates required for access to the desired repository.

> Note: If multiple certificates exist, each is tried in alphabetical order. If there is a 4xx-level or 5xx-level authentication error, Docker continues to try with the next certificate.

The following illustrates a configuration with custom certificates:

```
    /etc/docker/certs.d/        <-- Certificate directory
    └── localhost:5000          <-- Hostname:port
       ├── client.cert          <-- Client certificate
       ├── client.key           <-- Client key
       └── ca.crt               <-- Certificate authority that signe
d
                                    the registry certificate
```

The preceding example is operating-system specific and is for illustrative purposes only. You should consult your operating system documentation for creating an os-provided bundled certificate chain.

Creating the client certificates

Use OpenSSL's `genrsa` and `req` commands to first generate an RSA key and then use the key to create the certificate.

```
$ openssl genrsa -out client.key 4096
$ openssl req -new -x509 -text -key client.key -out client.cert
```

> Note: These TLS commands only generate a working set of certificates on Linux. The version of OpenSSL in macOS is incompatible with the type of certificate Docker requires.

Troubleshooting tips

The Docker daemon interprets `.crt` files as CA certificates and `.cert` files as client certificates. If a CA certificate is accidentally given the extension `.cert` instead of the correct `.crt` extension, the Docker daemon logs the following error message:

```
Missing key KEY_NAME for client certificate CERT_NAME. CA certificat
es should use the extension .crt.
```

If the Docker registry is accessed without a port number, do not add the port to the directory name. The following shows the configuration for a registry on default port 443 which is accessed with `docker login my-https.registry.example.com` :

```
    /etc/docker/certs.d/
    └── my-https.registry.example.com          <-- Hostname without
port
        ├── client.cert
        ├── client.key
        └── ca.crt
```

Related Information

- Use trusted images (https://docs.docker.com/engine/security/)
- Protect the Docker daemon socket (https://docs.docker.com/engine/security/https/)

Usage (https://docs.docker.com/glossary/?term=Usage), registry (https://docs.docker.com/glossary/?term=registry), repository (https://docs.docker.com/glossary/?term=repository), client (https://docs.docker.com/glossary/?term=client), root (https://docs.docker.com/glossary/?term=root), certificate (https://docs.docker.com/glossary/?term=certificate), docker (https://docs.docker.com/glossary/?term=docker), apache (https://docs.docker.com/glossary/?term=apache), ssl (https://docs.docker.com/glossary/?term=ssl), tls (https://docs.docker.com/glossary/?term=tls), documentation (https://docs.docker.com/glossary/?term=documentation), examples (https://docs.docker.com/glossary/?term=examples), articles (https://docs.docker.com/glossary/?term=articles), tutorials (https://docs.docker.com/glossary/?term=tutorials)

Docker for AWS setup & prerequisites

Estimated reading time: 8 minutes

Docker Enterprise Edition (EE) for AWS

This deployment is fully baked and tested, and comes with the latest Docker Enterprise Edition for AWS.

This release is maintained and receives security and critical bug fixes for one year.

Deploy Docker Enterprise Edition (EE) for AWS (https://hub.docker.com/editions/enterprise/docker-ee-aws?tab=description)

Docker Community Edition (CE) for AWS

Quickstart

If your account has the proper permissions (https://docs.docker.com/docker-for-aws/iam-permissions/), you can use the blue button from the stable or edge channel to bootstrap Docker for AWS using CloudFormation. For more about stable and edge channels, see the FAQs (https://docs.docker.com/docker-for-aws/faqs/#stable-and-edge-channels).

Stable channel

This deployment is fully baked and tested, and comes with the latest CE version of Docker.

This is the best channel to use if you want a reliable platform to work with.

Stable is released quarterly and is for users that want an easier-to-maintain release pace.

Deploy Docker Community Edition (CE) for AWS (stable) (https://console.aws.amazon.com/cloudformation/home#/stacks/new?stackName=Docker&templateURL=https://editions-us-east-1.s3.amazonaws.com/aws/stable/Docker.tmpl)

Deploy Docker Community Edition (CE) for AWS (stable) uses your existing VPC (https://console.aws.amazon.com/cloudformation/home#/stacks/new?stackName=Docker&templateURL=https://editions-us-east-1.s3.amazonaws.com/aws/stable/Docker-no-vpc.tmpl)

Edge channel

This deployment offers cutting edge features of the CE version of Docker and comes with experimental features turned on, described in the Docker Experimental Features README (https://github.com/docker/docker-ce/blob/master/components/cli/experimental/README.md) on GitHub. (Adju the branch or tag in the URL to match your version.)

This is the best channel to use if you want to experiment with features under development, and can weather some instability and bugs. Edge is for users wanting a drop of the latest and greatest features every month.

We collect usage data on edges across the board.

Deploy Docker Community Edition (CE) for AWS (edge) (https://console.aws.amazon.com/cloudformation/home#/stacks/new?stackName=Docker&templateURL=https://editions-us-east-1.s3.amazonaws.com/aws/edge/Docker.tmpl)

Deploy Docker Community Edition (CE) for AWS (edge) uses your existing VPC (https://console.aws.amazon.com/cloudformation/home#/stacks/new?stackName=Docker&templateURL=https://editions-us-east-1.s3.amazonaws.com/aws/edge/Docker-no-vpc.tmpl)

**Note* During stable channel updates, edge channel will have the same release (unless it's a patch release)

Deployment options

There are two ways to deploy Docker for AWS:

- With a pre-existing VPC
- With a new VPC created by Docker

We recommend allowing Docker for AWS to create the VPC since it allows Docker to optimize the environment. Installing in an existing VPC requires more work.

CREATE A NEW VPC

This approach creates a new VPC, subnets, gateways, and everything else needed to run Docker for AWS. It is the easiest way to get started, and requires the least amount of work.

All you need to do is run the CloudFormation template, answer some questions, and you are good to go.

INSTALL WITH AN EXISTING VPC

If you need to install Docker for AWS with an existing VPC, you need to do a few preliminary steps. See recommended VPC and Subnet setup (https://docs.docker.com/docker-for-aws/faqs/#recommended-vpc-and-subnet-setup) for more details.

1. Pick a VPC in a region you want to use.
2. Make sure the selected VPC is setup with an Internet Gateway, Subnets, and Route Tables.
3. You need to have three different subnets, ideally each in their own availability zone. If you are running in a region with only two Availability Zones, you need to add more than one subnet into one of the availability zones. For production deployments we recommend only deploying to regions that have three or more Availability Zones.
4. When you launch the docker for AWS CloudFormation stack, make sure you use the one for existing VPCs. This template prompts you for the VPC and subnets that you want to use for Docker for AWS.

Prerequisites

- Access to an AWS account with permissions to use CloudFormation and creating the following objects. Full set of required permissions (https://docs.docker.com/docker-for-aws/iam-permissions/).
    - EC2 instances + Auto Scaling groups
    - IAM profiles
    - DynamoDB Tables
    - SQS Queue
    - VPC + subnets and security groups
    - ELB
    - CloudWatch Log Group
- SSH key in AWS in the region where you want to deploy (required to access the completed Docker install)
- AWS account that support EC2-VPC (See the FAQ for details about EC2-Classic (https://docs.docker.com/docker-for-aws/faqs/))

For more information about adding an SSH key pair to your account, refer to the Amazon EC2 Key Pairs docs (http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html).

The China and US Gov Cloud AWS partitions are not currently supported.

Configuration

Docker for AWS is installed with a CloudFormation template that configures Docker in swarm mode, running on instances backed by custom AMIs. There are two ways you can deploy Docker for AWS. You can use the AWS Management Console (browser based), or use the AWS CLI. Both have the following configuration options.

CONFIGURATION OPTIONS

KeyName

Pick the SSH key to be used when you SSH into the manager nodes.

InstanceType

The EC2 instance type for your worker nodes.

ManagerInstanceType

The EC2 instance type for your manager nodes. The larger your swarm, the larger the instance size you should use.

ClusterSize

The number of workers you want in your swarm (0-1000).

ManagerSize

The number of managers in your swarm. On Docker CE, you can select either 1, 3 or 5 managers. We only recommend 1 manager for testing and dev setups. There are no failover guarantees with 1 manager — if the single manager fails the swarm goes down as well. Additionally, upgrading single-manager swarms is not currently guaranteed to succeed.

On Docker EE, you can choose to run with 3 or 5 managers.

We recommend at least 3 managers, and if you have a lot of workers, you should use 5 managers.

EnableSystemPrune

Enable if you want Docker for AWS to automatically cleanup unused space on your swarm nodes.

When enabled, `docker system prune` runs staggered every day, starting at 1:42AM UTC on both workers and managers. The prune times are staggered slightly so that not all nodes are pruned at the same time. This limits resource spikes on the swarm.

Pruning removes the following:

- All stopped containers
- All volumes not used by at least one container
- All dangling images
- All unused networks

EnableCloudWatchLogs

Enable if you want Docker to send your container logs to CloudWatch. ("yes", "no") Defaults to yes.

WorkerDiskSize

Size of Workers' ephemeral storage volume in GiB (20 - 1024).

WorkerDiskType

Worker ephemeral storage volume type ("standard", "gp2").

ManagerDiskSize

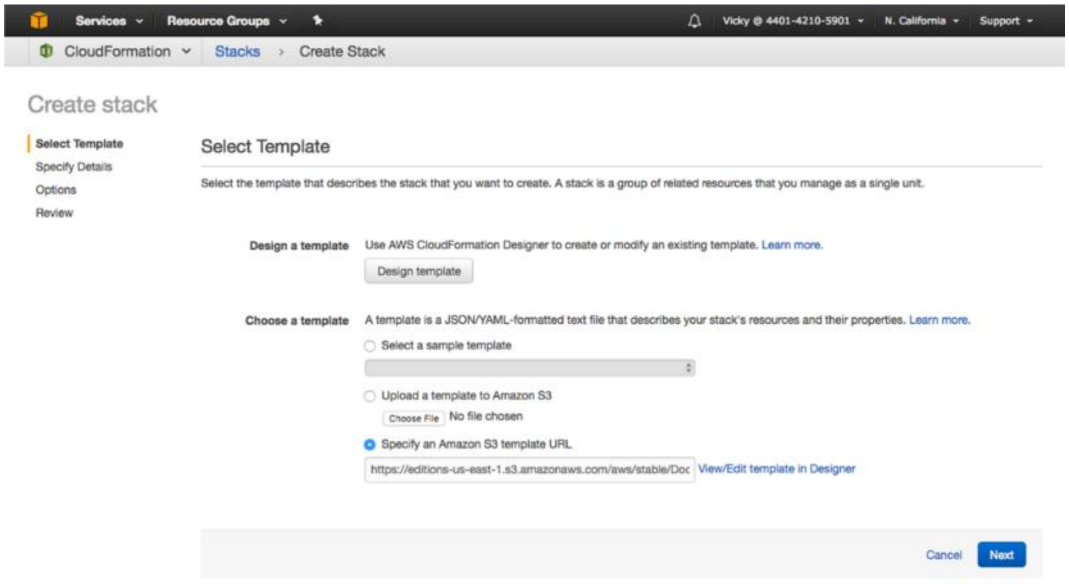
Size of Manager's ephemeral storage volume in GiB (20 - 1024)

ManagerDiskType

Manager ephemeral storage volume type ("standard", "gp2")

INSTALLING WITH THE AWS MANAGEMENT CONSOLE

The simplest way to use the template is to use one of the Quickstart (/docker-for-aws/#docker-community-edition-ce-for-aws) options with the CloudFormation section of the AWS Management Console.



Select Template

Select the template that describes the stack that you want to create. A stack is a group of related resources that you manage as a single unit.

Design a template: Use AWS CloudFormation Designer to create or modify an existing template. Learn more.

Choose a template: A template is a JSON/YAML-formatted text file that describes your stack's resources and their properties. Learn more.

No file chosen

https://editions-us-east-1.s3.amazonaws.com/aws/stable/Doc

INSTALLING WITH THE CLI

You can also invoke the Docker for AWS CloudFormation template from the AWS CLI:

Here is an example of how to use the CLI. Make sure you populate all of the parameters and their values from the above list:

```
$ aws cloudformation create-stack --stack-name teststack --template-url <templateurl> --parameters ParameterKey=<keyna
me>,ParameterValue=<keyvalue> ParameterKey=InstanceType,ParameterValue=t2.micro ParameterKey=ManagerInstanceType,Param
eterValue=t2.micro ParameterKey=ClusterSize,ParameterValue=1 .... --capabilities CAPABILITY_IAM
```

To fully automate installs, you can use the AWS Cloudformation API (http://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/Welcome.html).

How it works

Docker for AWS starts with a CloudFormation template that creates everything that you need from scratch. There are only a few prerequisites that are listed above.

The CloudFormation template first creates a new VPC along with subnets and security groups. After the networking set-up completes, two Auto Scaling Groups are created, one for the managers and one for the workers, and the configured capacity setting is applied. Managers start first and create a quorum using Raft, then the workers start and join the swarm one at a time. At this point, the swarm is comprised of X number of managers and Y number of workers, and you can deploy your applications. See the deployment (https://docs.docker.com/docker-for-aws/deploy/) docs for your next steps.

> To log into your nodes using SSH (https://docs.docker.com/docker-for-aws/deploy/#connecting-via-ssh), use the `docker` user rather than `root` or `ec2-user` .

If you increase the number of instances running in your worker Auto Scaling Group (via the AWS console, or updating the CloudFormation configuration), the new nodes that start up automatically join the swarm.

Elastic Load Balancers (ELBs) are set up to help with routing traffic to your swarm.

Logging

Docker for AWS automatically configures logging to Cloudwatch for containers you run on Docker for AWS. A Log Group is created for each Docker for AWS install, and a log stream for each container.

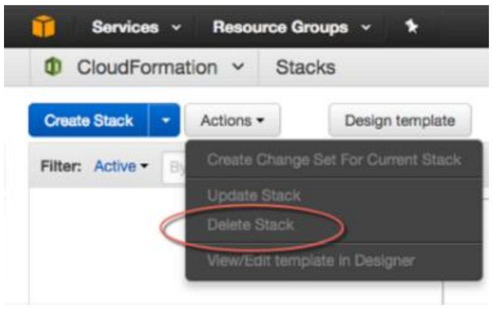
The `docker logs` and `docker service logs` commands are not supported on Docker for AWS when using Cloudwatch for logs. Instead, check container logs in CloudWatch.

System containers

Each node has a few system containers running on it to help run your swarm cluster. For everything to run smoothly, keep those containers running, and don't make any changes. If you make any changes, Docker for AWS does not work correctly.

Uninstalling or removing a stack

To uninstall Docker for AWS, log on to the AWS Console (https://aws.amazon.com/), navigate to Management Tools -> CloudFormation -> Actions -> Delete Stack, and select the Docker stack you want to remove.



Stack removal does not remove EBS and EFS volumes created by the cloudstor volume plugin or the S3 bucket associated with DTR. Those resources must be removed manually. See the cloudstor (https://docs.docker.com/docker-for-aws/persistent-data-volumes/#list-or-remove-volumes-created-by-cloudstor) docs for instructions on removing volumes.

aws (https://docs.docker.com/glossary/?term=aws), amazon (https://docs.docker.com/glossary/?term=amazon), iaas (https://docs.docker.com/glossary/?term=iaas), tutorial (https://docs.docker.com/glossary/?term=tutorial)

Install Docker Trusted Registry

Estimated reading time: 3 minutes

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

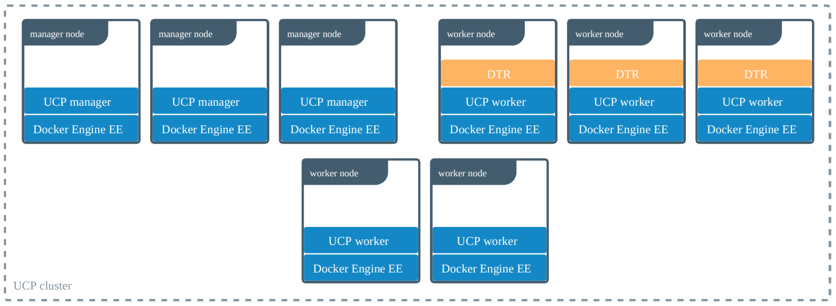
Step 1. Validate the system requirements

Before installing DTR, make sure your infrastructure meets the system requirements (https://docs.docker.com/ee/dtr/admin/install/system-requirements) that DTR needs to run.

Step 2. Install UCP

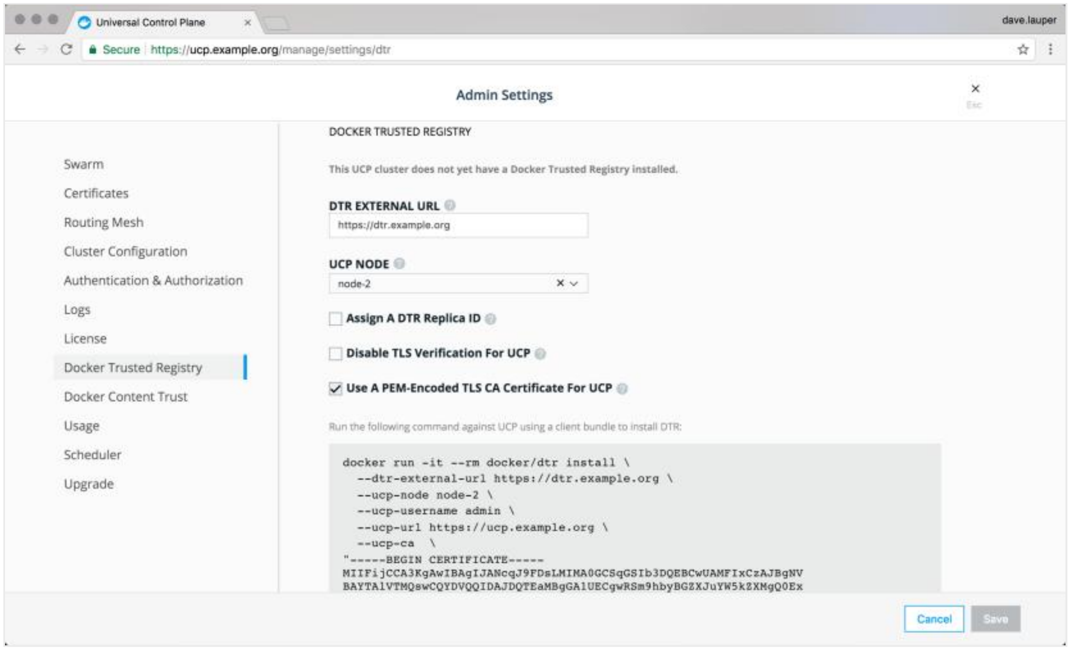
Since DTR requires Docker Universal Control Plane (UCP) to run, you need to install UCP (https://docs.docker.com/ee/dtr/admin/ucp/admin/install/) on all the nodes where you plan to install DTR.

DTR needs to be installed on a worker node that is being managed by UCP. You cannot 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.



DOCKER TRUSTED REGISTRY

This UCP cluster does not yet have a Docker Trusted Registry installed.

DTR EXTERNAL URL: https://dtr.example.org

UCP NODE: node-2

☐ Assign A DTR Replica ID

☐ Disable TLS Verification For UCP

☑ Use A PEM-Encoded TLS CA Certificate For UCP

Run the following command against UCP using a client bundle to install DTR:

```
docker run -it --rm docker/dtr install \
  --dtr-external-url https://dtr.example.org \
  --ucp-node node-2 \
  --ucp-username admin \
  --ucp-url https://ucp.example.org \
  --ucp-ca  \
"-----BEGIN CERTIFICATE-----
MIIFijCCA3KgAwIBAgIJANcqJ9FDsLMIMA0GCSqGSIb3DQEBCwUAMFIxCzAJBgNV
BAYTAlVTMQswCQYDVQQIDAJDQTEaMBgGA1UECgwRSm9hbyBGZXJuYW5kZXMgQ0Ex
```

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

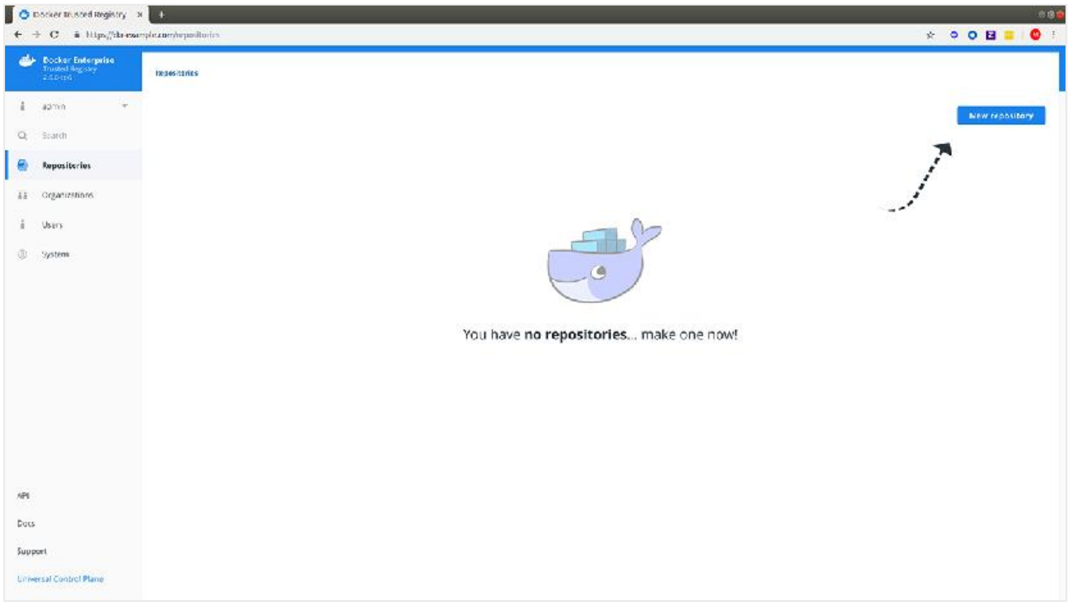
```
# Pull the latest version of DTR
$ docker pull docker/dtr:2.6.0

# Install DTR
$ docker run -it --rm \
  docker/dtr:2.6.0 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/).

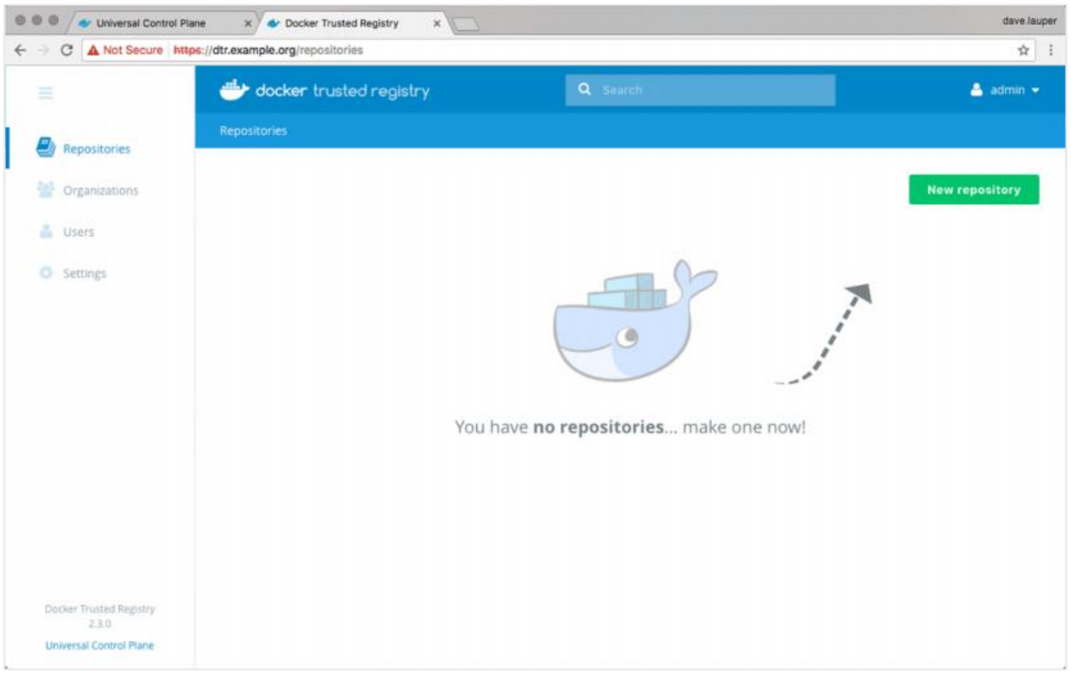
By default DTR is deployed with self-signed certificates, so your UCP deployment might not be able to pull images from DTR. Use the `--dtr-external-url <dtr-domain>:<port>` optional flag while deploying DTR, so that UCP is automatically reconfigured to trust DTR.

Step 4. Check that DTR is running

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



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

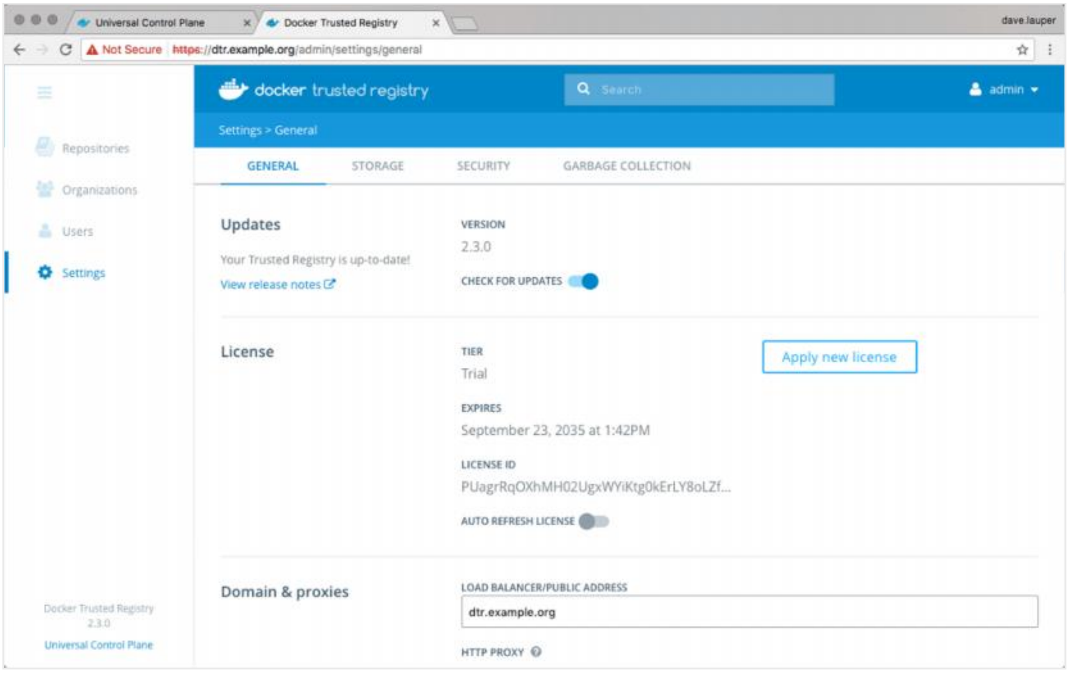


Step 5. Configure DTR

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.



Updates

Your Trusted Registry is up-to-date!

View release notes

VERSION
2.3.0

CHECK FOR UPDATES

License

TIER
Trial

EXPIRES
September 23, 2035 at 1:42PM

LICENSE ID
PUagrRqOXhMH02UgxWYiKtg0kErLY8oLZf...

AUTO REFRESH LICENSE

Domain & proxies

LOAD BALANCER/PUBLIC ADDRESS
dtr.example.org

HTTP PROXY

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/)

Step 7. Join replicas to the cluster

This step is optional.

To set up DTR for high availability, you can add more replicas to your DTR cluster. Adding more replicas allows you to load-balance requests across all replicas, and keep DTR working if a replica fails.

For high-availability you should set 3, 5, or 7 DTR replicas. The nodes where you're going to install these replicas also need to be managed by UCP.

To add replicas to a DTR cluster, use the `docker/dtr join` command:

1. Load your UCP user bundle.
2. Run the join command.

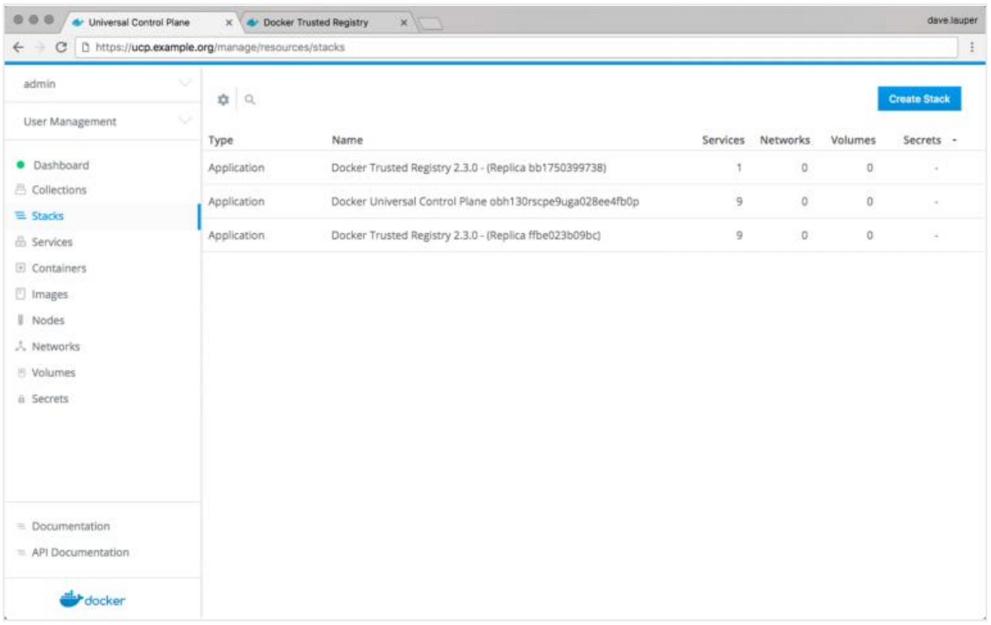
   When you join a replica to a DTR cluster, you need to specify the ID of a replica that is already part of the cluster. You can find an existing replica ID by going to the Applications page on UCP.

   Then run:

   ```
   docker run -it --rm \
     docker/dtr:2.6.0 join \
     --ucp-node <ucp-node-name> \
     --ucp-insecure-tls
   ```

3. Check that all replicas are running.

   In your browser, navigate to the Docker Universal Control Plane web interface, and navigate to the Applications screen. All replicas should be displayed.



https://ucp.example.org/manage/resources/stacks

Type	Name	Services	Networks	Volumes	Secrets
Application	Docker Trusted Registry 2.3.0 - (Replica bb1750399738)	1	0	0	-
Application	Docker Universal Control Plane obh130rscpe9uga028ee4fb0p	9	0	0	-
Application	Docker Trusted Registry 2.3.0 - (Replica ffbe023b09bc)	9	0	0	-

Where to go next

- Install DTR offline (https://docs.docker.com/ee/dtr/admin/install/install-offline/)
- Upgrade DTR (https://docs.docker.com/ee/dtr/admin/upgrade/)

dtr (https://docs.docker.com/glossary/?term=dtr), registry (https://docs.docker.com/glossary/?term=registry), install (https://docs.docker.com/glossary/?term=install)

Install UCP for production

Estimated reading time: 4 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 (UCP) is a containerized application that you can install on-premise or on a cloud infrastructure.

Step 1: Validate the system requirements

The first step to installing UCP is ensuring that your infrastructure has all of the requirements UCP needs to run (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/system-requirements/). Also, you need to ensure that all nodes, physical and virtual, are running the same version of Docker Enterprise Edition.

Step 2: Install Docker EE on all nodes

UCP is a containerized application that requires the commercially supported Docker Engine to run.

Install Docker EE on each host that you plan to manage with UCP. View the supported platforms (https://docs.docker.com/install/#supported-platforms) and click on your platform to get platform-specific instructions for installing Docker EE.

Make sure you install the same Docker EE version on all the nodes. Also, if you're creating virtual machine templates with Docker EE already installed, make sure the `/etc/docker/key.json` file is not included in the virtual machine image. When provisioning the virtual machine, restart the Docker daemon to generate a new `/etc/docker/key.json` file.

Step 3: Customize named volumes

Skip this step if you want to use the defaults provided by UCP.

Docker UCP uses named volumes to persist data. If you want to customize the drivers used to manage these volumes, you can create the volumes before installing UCP. When you install UCP, the installer will notice that the volumes already exist, and will start using them. Learn about the named volumes used by UCP (https://docs.docker.com/datacenter/ucp/2.2/guides/architecture/).

If these volumes don't exist, they'll be automatically created when installing UCP.

Step 4: Install UCP

To install UCP, you use the `docker/ucp` image, which has commands to install and manage UCP.

Make sure you follow the UCP System requirements (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/system-requirements/) in regards to networking ports. Ensure that your hardware or software firewalls are open appropriately or disabled.

To install UCP:

1. Use ssh to log in to the host where you want to install UCP.

2. Run the following command:

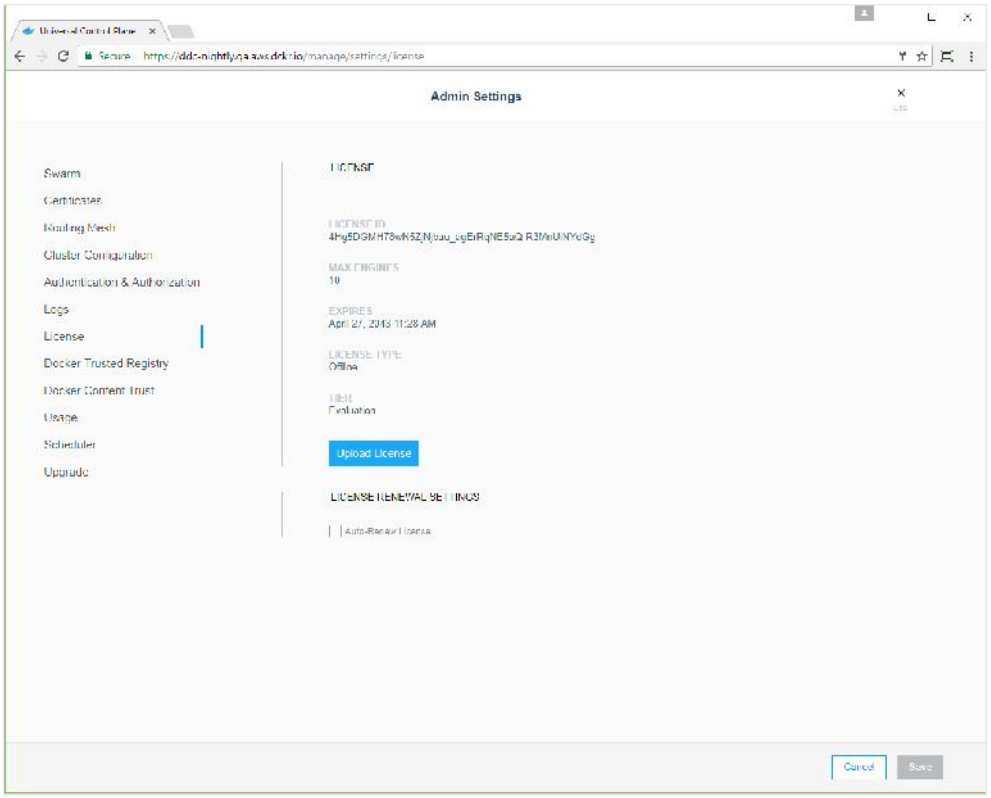
```
# Pull the latest version of UCP
$ docker image pull docker/ucp:2.2.14

# Install UCP
$ docker container run --rm -it --name ucp \
  -v /var/run/docker.sock:/var/run/docker.sock \
  docker/ucp:2.2.14 install \
  --host-address <node-ip-address> \
  --interactive
```

This runs the install command in interactive mode, so that you're prompted for any necessary configuration values. To find what other options are available in the install command, check the reference documentation (https://docs.docker.com/datacenter/ucp/2.2/reference/cli/install/).

Step 5: License your installation

Now that UCP is installed, you need to license it.

1. Go to Docker Hub (https://hub.docker.com/editions/enterprise/docker-ee-trial/trial) to get a free trial license.
2. In your browser, navigate to the UCP web UI, log in with your administrator credentials and upload your license. Navigate to the Admin Settings page and in the left pane, click License.



Admin Settings

LICENSE

LICENSE ID
4Hg5DGMH78wN5ZjNjbau_sgErRqNE5aQ R3MnUINYdGg

MAX ENGINES
10

EXPIRES
April 27, 2018 11:28 AM

LICENSE TYPE
Offline

TIER
Evaluation

LICENSE RENEWAL SETTINGS

Auto-Renew License

3. Click Upload License and navigate to your license (.lic) file. When you're finished selecting the license, UCP updates with the new settings.

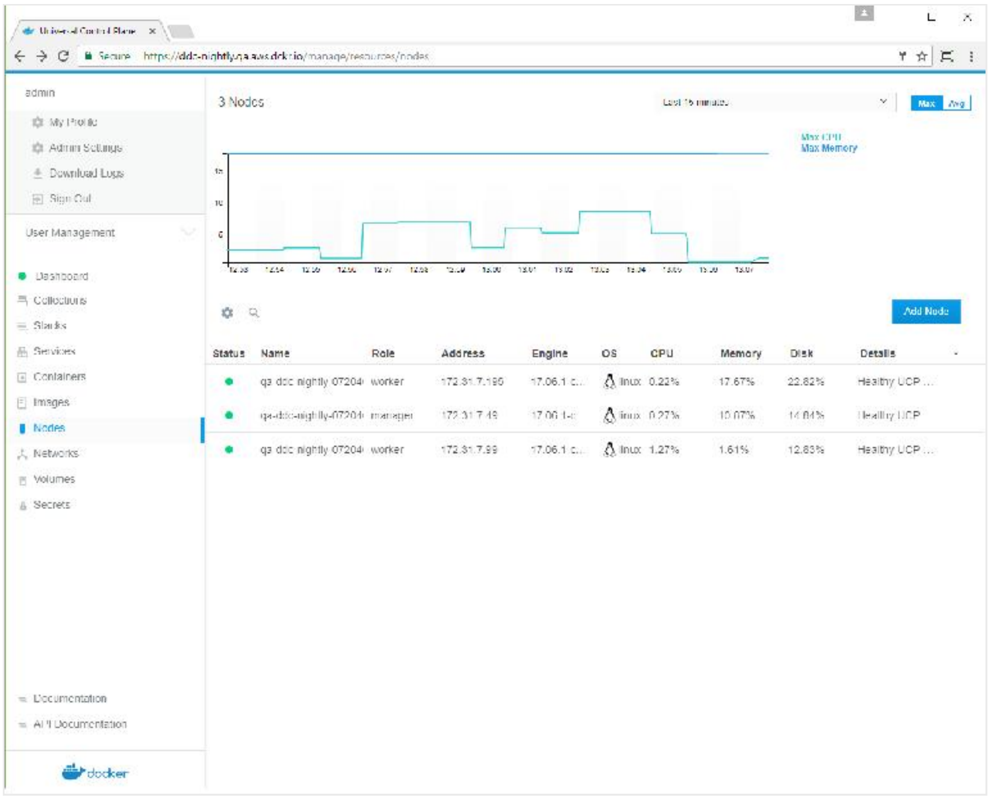
Step 6: Join manager nodes

Skip this step if you don't want UCP to be highly available.

To make your Docker swarm and UCP fault-tolerant and highly available, you can join more manager nodes to it. Manager nodes are the nodes in the swarm that perform the orchestration and swarm management tasks, and dispatch tasks for worker nodes to execute.

To join manager nodes to the swarm,

1. In the UCP web UI, navigate to the Nodes page, and click the Add Node button to add a new node.



Status	Name	Role	Address	Engine	OS	CPU	Memory	Disk	Details
●	qa-ddc-nightly-07204	worker	172.31.7.195	17.06.1-c...	linux	0.22%	17.67%	22.82%	Healthy UCP ...
●	qa-ddc-nightly-07204	manager	172.31.7.49	17.06.1-c	linux	0.27%	10.07%	14.04%	Healthy UCP
●	qa-ddc-nightly-07204	worker	172.31.7.99	17.06.1-c...	linux	1.27%	1.61%	12.83%	Healthy UCP ...

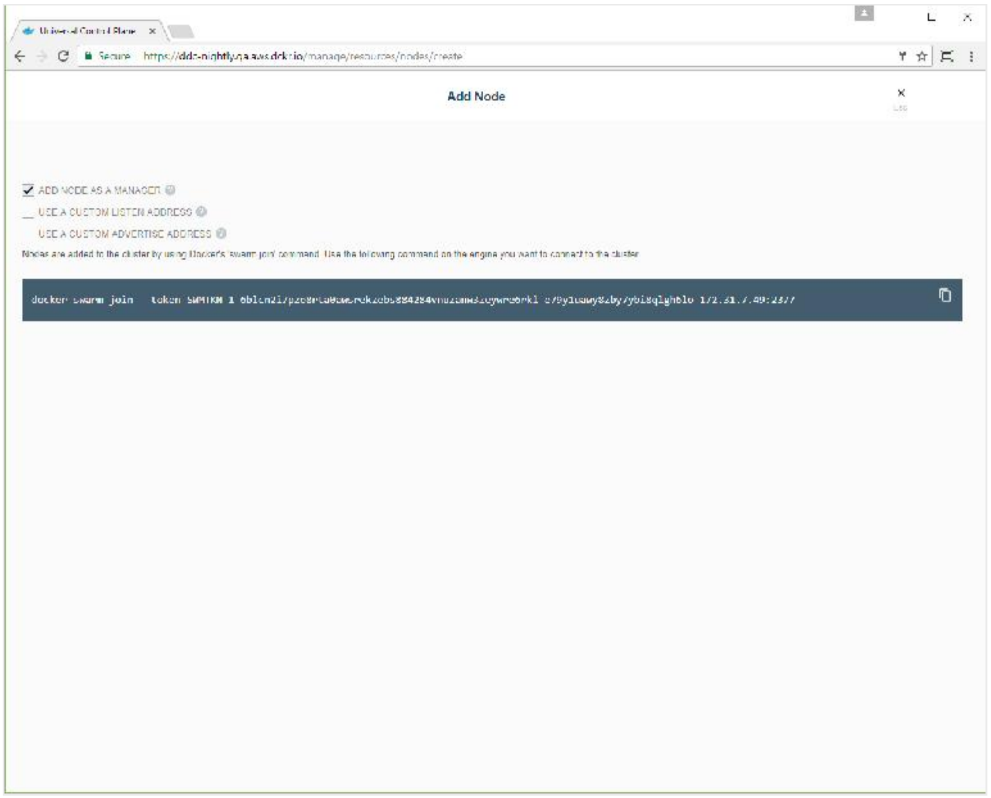
2. In the Add Node page, check Add node as a manager to turn this node into a manager and replicate UCP for high-availability.

3. If you want to customize the network and port where the new node listens for swarm management traffic, click Use a custom listen address. Enter the IP address and port for the node to listen for inbound cluster management traffic. The format is `interface:port` or `ip:port` . The default is `0.0.0.0:2377` .

4. If you want to customize the network and port that the new node advertises to other swarm members for API access, click Use a custom advertise address and enter the IP address and port. By default, this is also the outbound address used by the new node to contact UCP. The joining node can contact itself at this address. The format is `interface:port` or `ip:port` .

   Click the copy icon

   to copy the `docker swarm join` command that nodes use to join the swarm.


Universal Control Plane
Secure https://ddc-nightly.qa.aws.dckr.io/manage/resources/nodes/create
Add Node

[✓] ADD NODE AS A MANAGER
[ ] USE A CUSTOM LISTEN ADDRESS
USE A CUSTOM ADVERTISE ADDRESS

Nodes are added to the cluster by using Docker's 'swarm join' command. Use the following command on the engine you want to connect to the cluster.

```
docker swarm join --token SWMTKN-1-6blcn2i7pz68rta0bme rckzobs884284vnuzamasioywreOrkl-c79y1uaay8zby7yb18qlgh6lo 172.31.7.49:2377
```

5. For each manager node that you want to join to the swarm, log in using ssh and run the join command that you copied. After the join command completes, the node appears on the Nodes page in the UCP web UI.

Step 7: Join worker nodes

Skip this step if you don't want to add more nodes to run and scale your apps.

To add more computational resources to your swarm, you can join worker nodes. These nodes execute tasks assigned to them by the manager nodes. Follow the same steps as before, but don't check the Add node as a manager option.

Where to go next

- Use your own TLS certificates (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/use-your-own-tls-certificates/)
- Scale your cluster (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/scale-your-cluster/)

Universal Control Plane (https://docs.docker.com/glossary/?term=Universal Control Plane), UCP (https://docs.docker.com/glossary/?term=UCP), install (https://docs.docker.com/glossary/?term=install), Docker EE (https://docs.docker.com/glossary/?term=Docker EE)

Backups and disaster recovery

Estimated reading time: 6 minutes

These are the docs for UCP version 2.2.14

To select a different version, use the selector below.

2.2.14

When you decide to start using Docker Universal Control Plane on a production setting, you should configure it for high availability (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/set-up-high-availability/).

The next step is creating a backup policy and disaster recovery plan.

Data managed by UCP

UCP maintains data about:

Data	Description
Configurations	The UCP cluster configurations, as shown by `docker config ls` , including Docker EE license and swarm and client CAs
Access control	Permissions for teams to swarm resources, including collections, grants, and roles
Certificates and keys	The certificates, public keys, and private keys that are used for authentication and mutual TLS communication
Metrics data	Monitoring data gathered by UCP
Organizations	Your users, teams, and orgs
Volumes	All UCP named volumes (https://docs.docker.com/datacenter/ucp/2.2/guides/architecture/#volumes-used-by-ucp), which include all UCP component certs and data

This data is persisted on the host running UCP, using named volumes. Learn more about UCP named volumes (https://docs.docker.com/datacenter/ucp/2.2/guides/architecture/).

Backup steps

Back up your Docker EE components in the following order:

1. Back up your swarm (https://docs.docker.com/engine/swarm/admin_guide/#back-up-the-swarm)
2. Back up UCP
3. Back up DTR (https://docs.docker.com/datacenter/dtr/2.3/guides/admin/backups-and-disaster-recovery/)

Backup policy

As part of your backup policy you should regularly create backups of UCP. DTR is backed up independently. Learn about DTR backups and recovery (https://docs.docker.com/datacenter/dtr/2.3/guides/admin/backups-and-disaster-recovery/).

To create a UCP backup, run the `docker/ucp:2.2.14 backup` command on a single UCP manager. This command creates a tar archive with the contents of all the volumes used by UCP (https://docs.docker.com/datacenter/ucp/2.2/guides/architecture/) to persist data and streams it to stdout. The backup doesn't include the swarm-mode state, like service definitions and overlay network definitions.

You only need to run the backup command on a single UCP manager node. Since UCP stores the same data on all manager nodes, you only need to take periodic backups of a single manager node.

To create a consistent backup, the backup command temporarily stops the UCP containers running on the node where the backup is being performed. User resources, such as services, containers, and stacks are not affected by this operation and will continue operating as expected. Any long-lasting `exec`, `logs`, `events`, or `attach` operations on the affected manager node will be disconnected.

Additionally, if UCP is not configured for high availability, you will be temporarily unable to:

- Log in to the UCP Web UI
- Perform CLI operations using existing client bundles

To minimize the impact of the backup policy on your business, you should:

- Configure UCP for high availability (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/set-up-high-availability/). This allows load-balancing user requests across multiple UCP manager nodes.
- Schedule the backup to take place outside business hours.

Backup command

The example below shows how to create a backup of a UCP manager node and verify its contents:

```
# Create a backup, encrypt it, and store it on /tmp/backup.tar
docker container run \
  --log-driver none --rm \
  --interactive \
  --name ucp \
  -Ấ /var/run/docker.sock:/var/run/docker.sock \
  docker/ucp:2.2.14 backup \
  --id <ucp-instance-id> \
  --passphrase "secret" > /tmp/backup.tar

# Decrypt the backup and list its contents
$ gpg --decrypt /tmp/backup.tar | tar --list
```

Security-Enhanced Linux (SELinux)

For Docker EE 17.06 or higher, if the Docker engine has SELinux enabled, which is typical for RHEL hosts, you need to include `--security-opt label=disable` in the `docker` command:

```
$ docker container run --security-opt label=disable --log-driver none --r
m -i --name ucp \
  -Ấ /var/run/docker.sock:/var/run/docker.sock \
  docker/ucp:2.2.14 backup --interactive > /tmp/backup.tar
```

To find out whether SELinux is enabled in the engine, view the host's `/etc/docker/daemon.json` file and search for the string `"selinux-enabled":"true"` .

Restore UCP

To restore an existing UCP installation from a backup, you need to uninstall UCP from the swarm by using the `uninstall-ucp` command. Learn to uninstall UCP (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/install/uninstall/).

When restoring, make sure you use the same version of the `docker/ucp` image that you've used to create the backup. The example below shows how to restore UCP from an existing backup file, presumed to be located at `/tmp/backup.tar` :

```
$ docker container run --rm -i --name ucp \
  -v /var/run/docker.sock:/var/run/docker.sock  \
  docker/ucp:2.2.14 restore < /tmp/backup.tar
```

If the backup file is encrypted with a passphrase, you will need to provide the passphrase to the restore operation:

```
$ docker container run --rm -i --name ucp \
  -v /var/run/docker.sock:/var/run/docker.sock  \
  docker/ucp:2.2.14 restore --passphrase "secret" < /tmp/backup.tar
```

The restore command may also be invoked in interactive mode, in which case the backup file should be mounted to the container rather than streamed through stdin:

```
$ docker container run --rm -i --name ucp \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /tmp/backup.tar:/config/backup.tar \
  docker/ucp:2.2.14 restore -i
```

UCP and Swarm

UCP restore recovers the following assets from the backup file:

- Users, teams, and permissions.
- All UCP configuration options available under `Admin Settings` , like the Docker EE subscription license, scheduling options, content trust, and authentication backends.

UCP restore does not include swarm assets such as cluster membership, services, networks, secrets, etc. Learn to backup a swarm (https://docs.docker.com/engine/swarm/admin_guide/#back-up-the-swarm).

There are two ways to restore UCP:

- On a manager node of an existing swarm which does not have UCP installed. In this case, UCP restore will use the existing swarm.

- On a docker engine that isn't participating in a swarm. In this case, a new swarm is created and UCP is restored on top.

Disaster recovery

In the event where half or more manager nodes are lost and cannot be recovered to a healthy state, the system is considered to have lost quorum and can only be restored through the following disaster recovery procedure. If your cluster has lost quorum, you can still take a backup of one of the remaining nodes, but we recommend making backups regularly.

This procedure is not guaranteed to succeed with no loss of running services or configuration data. To properly protect against manager failures, the system should be configured for high availability (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/set-up-high-availability/).

1. On one of the remaining manager nodes, perform `docker swarm init --force-new-cluster` . You may also need to specify an `--advertise-addr` parameter which is equivalent to the `--host-address` parameter of the `docker/ucp install` operation. This will instantiate a new single-manager swarm by recovering as much state as possible from the existing manager. This is a disruptive operation and existing tasks may be either terminated or suspended.
2. Obtain a backup of one of the remaining manager nodes if one is not already available.
3. If UCP is still installed on the swarm, uninstall UCP using the `uninstall-ucp` command.
4. Perform a restore operation on the recovered swarm manager node.
5. Log in to UCP and browse to the nodes page, or use the CLI `docker node ls` command.
6. If any nodes are listed as `down` , you need to manually remove these nodes (https://docs.docker.com/datacenter/ucp/2.2/guides/configure/scale-your-cluster/) from the swarm and then re-join them using a `docker swarm join` operation with the swarm's new join-token.

Where to go next

- Set up high availability (https://docs.docker.com/datacenter/ucp/2.2/guides/admin/configure/set-up-high-availability/)
- UCP architecture (https://docs.docker.com/datacenter/ucp/2.2/guides/architecture/)

ucp (https://docs.docker.com/glossary/?term=ucp), backup (https://docs.docker.com/glossary/?term=backup), restore

(https://docs.docker.com/glossary/?term=restore), recovery
(https://docs.docker.com/glossary/?term=recovery)

DTR backups and recovery

Estimated reading time: 7 minutes

✓ These are the docs for DTR version 2.3.9

To select a different version, use the selector below.

2.3.9 ▾

DTR requires that a majority (n/2 + 1) of its replicas are healthy at all times for it to work. So if a majority of replicas is unhealthy or lost, the only way to restore DTR to a working state, is by recovering from a backup. This is why it's important to ensure replicas are healthy and perform frequent backups.

Data managed by DTR

Docker Trusted Registry maintains data about:

Data	Description
Configurations	The DTR cluster configurations
Repository metadata	The metadata about the repositories and images deployed
Access control to repos and images	Permissions for teams and repositories
Notary data	Notary tags and signatures
Scan results	Security scanning results for images
Certificates and keys	The certificates, public keys, and private keys that are used for mutual TLS communication

Data	Description
Images content	The images you push to DTR. This can be stored on the filesystem of the node running DTR, or other storage system, depending on the configuration

This data is persisted on the host running DTR, using named volumes. Learn more about DTR named volumes (https://docs.docker.com/datacenter/dtr/2.3/guides/architecture/).

To perform a backup of a DTR node, run the `docker/dtr backup` command. This command backups up the following data:

Data	Backed up	Description
Configurations	yes	DTR settings
Repository metadata	yes	Metadata like image architecture and size
Access control to repos and images	yes	Data about who has access to which images
Notary data	yes	Signatures and digests for images that are signed
Scan results	yes	Information about vulnerabilities in your images
Certificates and keys	yes	TLS certificates and keys used by DTR
Image content	no	Needs to be backed up separately, depends on DTR configuration
Users, orgs, teams	no	Create a UCP backup to backup this data
Vulnerability database	no	Can be re-downloaded after a restore

Backup DTR data

To create a backup of DTR you need to:

1. Backup image content
2. Backup DTR metadata

You should always create backups from the same DTR replica, to ensure a smoother restore.

Backup image content

Since you can configure the storage backend that DTR uses to store images, the way you backup images depends on the storage backend you're using.

If you've configured DTR to store images on the local filesystem or NFS mount, you can backup the images by using ssh to log into a node where DTR is running, and creating a tar archive of the dtr-registry volume (https://docs.docker.com/datacenter/dtr/2.3/guides/architecture/):

```
sudo tar -cf backup-images.tar \
  $(dirname $(docker volume inspect --format '{{.Mountpoint}}' dtr-r
egistry-<replica-id>))
```

If you're using a different storage backend, follow the best practices recommended for that system.

Backup DTR metadata

To create a DTR backup, load your UCP client bundle, and run the following command, replacing the placeholders for the real values:

```
read -sp 'ucp password: ' UCP_PASSWORD; \
docker run --log-driver none -i --rm \
  --env UCP_PASSWORD=$UCP_PASSWORD \
  docker/dtr:2.3.9 backup \
  --ucp-url <ucp-url> \
  --ucp-insecure-tls \
  --ucp-username <ucp-username> \
  --existing-replica-id <replica-id> > backup-metadata.tar
```

Where:

- `<ucp-url>` is the url you use to access UCP
- `<ucp-username>` is the username of a UCP administrator

- `<replica-id>` is the id of the DTR replica to backup

This prompts you for the UCP password, backups up the DTR metadata and saves the result into a tar archive. You can learn more about the supported flags in the reference documentation (https://docs.docker.com/datacenter/dtr/2.3/reference/cli/backup/).

The backup command doesn't stop DTR, so that you can take frequent backups without affecting your users. Also, the backup contains sensitive information like private keys, so you can encrypt the backup by running:

```
gpg --symmetric
```

This prompts you for a password to encrypt the backup, copies the backup file and encrypts it.

Test your backups

To validate that the backup was correctly performed, you can print the contents of the tar file created. The backup of the images should look like:

```
tar -tf backup-images.tar

dtr-backup-v2.3.9/
dtr-backup-v2.3.9/rethink/
dtr-backup-v2.3.9/rethink/layers/
```

And the backup of the DTR metadata should look like:

```
tar -tf

# The archive should look like this
dtr-backup-v2.3.9/
dtr-backup-v2.3.9/rethink/
dtr-backup-v2.3.9/rethink/properties/
dtr-backup-v2.3.9/rethink/properties/0
```

If you've encrypted the metadata backup, you can use:

```
gpg -d /tmp/backup.tar.gpg | tar -t
```

You can also create a backup of a UCP cluster and restore it into a new cluster. Then restore DTR on that new cluster to confirm that everything is working as expected.

Restore DTR data

If your DTR has a majority of unhealthy replicas, the one way to restore it to a working state is by restoring from an existing backup.

To restore DTR, you need to:

1. Stop any DTR containers that might be running
2. Restore the images from a backup
3. Restore DTR metadata from a backup
4. Re-fetch the vulnerability database

You need to restore DTR on the same UCP cluster where you've created the backup. If you restore on a different UCP cluster, all DTR resources will be owned by users that don't exist, so you can't manage the resources, even though they're stored in the DTR data store.

When restoring, you need to use the same version of the `docker/dtr` image that you've used when creating the update. Other versions are not guaranteed to work.

Stop DTR containers

Start by removing any DTR container that is still running:

```
docker run -it --rm \
  docker/dtr:2.3.9 destroy \
  --ucp-insecure-tls
```

Restore images

If you had DTR configured to store images on the local filesystem, you can extract your backup:

```
sudo tar -xzf backup-images.tar -C /var/lib/docker/volumes
```

If you're using a different storage backend, follow the best practices recommended for that system. When restoring the DTR metadata, DTR will be deployed with the same configurations it had when creating the backup.

Restore DTR metadata

You can restore the DTR metadata with the `docker/dtr restore` command. This performs a fresh installation of DTR, and reconfigures it with the configuration created during a backup.

Load your UCP client bundle, and run the following command, replacing the placeholders for the real values:

```
read -sp 'ucp password: ' UCP_PASSWORD; \
docker run -i --rm \
  --env UCP_PASSWORD=$UCP_PASSWORD \
  docker/dtr:2.3.9 restore \
  --ucp-url <ucp-url> \
  --ucp-insecure-tls \
  --ucp-username <ucp-username> \
  --ucp-node <hostname> \
  --replica-id <replica-id> \
  --dtr-external-url <dtr-external-url> < backup-metadata.tar
```

Where:

- `<ucp-url>` is the url you use to access UCP
- `<ucp-username>` is the username of a UCP administrator
- `<hostname>` is the hostname of the node where you've restored the images
- `<replica-id>` the id of the replica you backed up
- `<dtr-external-url>` the url that clients use to access DTR

Re-fetch the vulnerability database

If you're scanning images, you now need to download the vulnerability database.

After you successfully restore DTR, you can join new replicas the same way you would after a fresh installation. Learn more (https://docs.docker.com/datacenter/dtr/2.3/guides/admin/configure/set-up-vulnerability-scans/).

Where to go next

- Set up high availability (https://docs.docker.com/datacenter/dtr/2.3/guides/admin/configure/set-up-high-availability/)
- DTR architecture (https://docs.docker.com/datacenter/dtr/2.3/guides/architecture/)

registry (https://docs.docker.com/glossary/?term=registry), high-availability (https://docs.docker.com/glossary/?term=high-availability), backup (https://docs.docker.com/glossary/?term=backup), recovery (https://docs.docker.com/glossary/?term=recovery)

Post-installation steps for Linux

Estimated reading time: 15 minutes

This section contains optional procedures for configuring Linux hosts to work better with Docker.

Manage Docker as a non-root user

The Docker daemon binds to a Unix socket instead of a TCP port. By default that Unix socket is owned by the user `root` and other users can only access it using `sudo`. The Docker daemon always runs as the `root` user.

If you don't want to preface the `docker` command with `sudo`, create a Unix group called `docker` and add users to it. When the Docker daemon starts, it creates a Unix socket accessible by members of the `docker` group.

> Warning
>
> The `docker` group grants privileges equivalent to the `root` user. For details on how this impacts security in your system, see *Docker Daemon Attack Surface* (https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface).

To create the `docker` group and add your user:

1. Create the `docker` group.

   ```
   $ sudo groupadd docker
   ```

2. Add your user to the `docker` group.

   ```
   $ sudo usermod -aG docker $USER
   ```

3. Log out and log back in so that your group membership is re-evaluated.

   If testing on a virtual machine, it may be necessary to restart the virtual machine for changes to take effect.

   On a desktop Linux environment such as X Windows, log out of your session completely and then log back in.

4. Verify that you can run `docker` commands without `sudo` .

   ```
   $ docker run hello-world
   ```

   This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

   If you initially ran Docker CLI commands using `sudo` before adding your user to the `docker` group, you may see the following error, which indicates that your `~/.docker/` directory was created with incorrect permissions due to the `sudo` commands.

   ```
   WARNING: Error loading config file: /home/user/.docker/config.json -
   stat /home/user/.docker/config.json: permission denied
   ```

   To fix this problem, either remove the `~/.docker/` directory (it is recreated automatically, but any custom settings are lost), or change its ownership and permissions using the following commands:

   ```
   $ sudo chown "$USER":"$USER" /home/"$USER"/.docker -R
   $ sudo chmod g+rwx "$HOME/.docker" -R
   ```

Configure Docker to start on boot

Most current Linux distributions (RHEL, CentOS, Fedora, Ubuntu 16.04 and higher) use `systemd` (/install/linux/linux-postinstall/#systemd) to manage which services start when the system boots. Ubuntu 14.10 and below use `upstart` (/install/linux/linux-postinstall/#upstart).

`systemd`

```
$ sudo systemctl enable docker
```

To disable this behavior, use `disable` instead.

```
$ sudo systemctl disable docker
```

If you need to add an HTTP Proxy, set a different directory or partition for the Docker runtime files, or make other customizations, see customize your systemd Docker daemon options (https://docs.docker.com/engine/admin/systemd/).

`upstart`

Docker is automatically configured to start on boot using `upstart`. To disable this behavior, use the following command:

```
$ echo manual | sudo tee /etc/init/docker.override
```

`chkconfig`

```
$ sudo chkconfig docker on
```

Use a different storage engine

For information about the different storage engines, see Storage drivers (https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/). The default storage engine and the list of supported storage engines depend on your host's Linux distribution and available kernel drivers.

Configure where the Docker daemon listens for connections

By default, the Docker daemon listens for connections on a UNIX socket to accept requests from local clients. It is possible to allow Docker to accept requests from remote hosts by configuring it to listen on an IP address and port as well as the UNIX socket. For more detailed information on this configuration option take a look at "Bind Docker to another host/port or a unix socket" section of the Docker CLI Reference (https://docs.docker.com/engine/reference/commandline/dockerd/) article.

✓ Docker EE customers

Docker EE customers can get remote CLI access to UCP with the UCP client bundle. A UCP Client Bundle is generated by UCP and secured by mutual TLS. See the document on CLI access for UCP (https://docs.docker.com/ee/ucp/user-access/cli/) for more information.

Secure your connection

Before configuring Docker to accept connections from remote hosts it is critically important that you understand the security implications of opening docker to the network. If steps are not taken to secure the connection, it is possible for remote non-root users to gain root access on the host. For more information on how to use TLS certificates to secure this connection, check this article on how to protect the Docker daemon socket (https://docs.docker.com/engine/security/https/).

Configuring Docker to accept remote connections can be done with the `docker.service` systemd unit file for Linux distributions using systemd, such as recent versions of RedHat, CentOS, Ubuntu and SLES, or with the `daemon.json` file which is recommended for Linux distributions that do not use systemd.

systemd vs daemon.json

Configuring Docker to listen for connections using both the `systemd` unit file and the `daemon.json` file causes a conflict that prevents Docker from starting.

Configuring remote access with `systemd` unit file

1. Use the command `sudo systemctl edit docker.service` to open an override file for `docker.service` in a text editor.

2. Add or modify the following lines, substituting your own values.

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://127.0.0.1:2375
```

3. Save the file.

4. Reload the `systemctl` configuration.

```
$ sudo systemctl daemon-reload
```

5. Restart Docker.

```
$ sudo systemctl restart docker.service
```

6. Check to see whether the change was honored by reviewing the output of `netstat` to confirm `dockerd` is listening on the configured port.

```
$ sudo netstat -lntp | grep dockerd
tcp        0      0 127.0.0.1:2375          0.0.0.0:*               LISTEN
      3758/dockerd
```

Configuring remote access with `daemon.json`

1. Set the `hosts` array in the `/etc/docker/daemon.json` to connect to the UNIX socket and an IP address, as follows:

   ```
   {
   "hosts": ["unix:///var/run/docker.sock", "tcp://127.0.0.1:2375"]
   }
   ```

2. Restart Docker.

3. Check to see whether the change was honored by reviewing the output of `netstat` to confirm `dockerd` is listening on the configured port.

   ```
   $ sudo netstat -lntp | grep dockerd
   tcp        0      0 127.0.0.1:2375          0.0.0.0:*               LISTEN
         3758/dockerd
   ```

Enable IPv6 on the Docker daemon

To enable IPv6 on the Docker daemon, see Enable IPv6 support (https://docs.docker.com/config/daemon/ipv6/).

Troubleshooting

Kernel compatibility

Docker cannot run correctly if your kernel is older than version 3.10 or if it is missing some modules. To check kernel compatibility, you can download and run the `check-config.sh` (https://raw.githubusercontent.com/docker/docker/master/contrib/check-config.sh) script.

```
$ curl https://raw.githubusercontent.com/docker/docker/master/contrib/check-conf
ig.sh > check-config.sh

$ bash ./check-config.sh
```

The script only works on Linux, not macOS.

Cannot connect to the Docker daemon

If you see an error such as the following, your Docker client may be configured to connect to a Docker daemon on a different host, and that host may not be reachable.

```
Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?
```

To see which host your client is configured to connect to, check the value of the `DOCKER_HOST` variable in your environment.

```
$ env | grep DOCKER_HOST
```

If this command returns a value, the Docker client is set to connect to a Docker daemon running on that host. If it is unset, the Docker client is set to connect to the Docker daemon running on the local host. If it is set in error, use the following command to unset it:

```
$ unset DOCKER_HOST
```

You may need to edit your environment in files such as `~/.bashrc` or `~/.profile` to prevent the `DOCKER_HOST` variable from being set erroneously.

If `DOCKER_HOST` is set as intended, verify that the Docker daemon is running on the remote host and that a firewall or network outage is not preventing you from connecting.

IP forwarding problems

If you manually configure your network using `systemd-network` with `systemd` version 219 or higher, Docker containers may not be able to access your network. Beginning with `systemd` version 220, the forwarding setting for a given network (`net.ipv4.conf.<interface>.forwarding`) defaults to *off*. This setting prevents IP forwarding. It also conflicts with Docker's behavior of enabling the `net.ipv4.conf.all.forwarding` setting within containers.

To work around this on RHEL, CentOS, or Fedora, edit the `<interface>.network` file in `/usr/lib/systemd/network/` on your Docker host (ex: `/usr/lib/systemd/network/80-container-host0.network`) and add the following block within the `[Network]` section.

```
[Network]
...
IPForward=kernel
# OR
IPForward=true
...
```

This configuration allows IP forwarding from the container as expected.

`DNS resolver found in resolv.conf and containers can't use it`

Linux systems which use a GUI often have a network manager running, which uses a `dnsmasq` instance running on a loopback address such as `127.0.0.1` or `127.0.1.1` to cache DNS requests, and adds this entry to `/etc/resolv.conf`. The `dnsmasq` service speeds up DNS look-ups and also provides DHCP services. This configuration does not work within a Docker container which has its own network namespace, because the Docker container resolves loopback addresses such as `127.0.0.1` to itself, and it is very unlikely to be running a DNS server on its own loopback address.

If Docker detects that no DNS server referenced in `/etc/resolv.conf` is a fully functional DNS server, the following warning occurs and Docker uses the public DNS servers provided by Google at `8.8.8.8` and `8.8.4.4` for DNS resolution.

```
WARNING: Local (127.0.0.1) DNS resolver found in resolv.conf and containers
can't use it. Using default external servers : [8.8.8.8 8.8.4.4]
```

If you see this warning, first check to see if you use `dnsmasq`:

```
$ ps aux |grep dnsmasq
```

If your container needs to resolve hosts which are internal to your network, the public nameservers are not adequate. You have two choices:

- You can specify a DNS server for Docker to use, or
- You can disable `dnsmasq` in NetworkManager. If you do this, NetworkManager adds your true DNS nameserver to `/etc/resolv.conf`, but you lose the possible benefits of `dnsmasq`.

You only need to use one of these methods.

Specify DNS servers for Docker

The default location of the configuration file is `/etc/docker/daemon.json`. You can change the location of the configuration file using the `--config-file` daemon flag. The documentation below assumes the configuration file is located at `/etc/docker/daemon.json`.

1. Create or edit the Docker daemon configuration file, which defaults to `/etc/docker/daemon.json` file, which controls the Docker daemon configuration.

   ```
   $ sudo nano /etc/docker/daemon.json
   ```

2. Add a `dns` key with one or more IP addresses as values. If the file has existing contents, you only need to add or edit the `dns` line.

   ```
   {
           "dns": ["8.8.8.8", "8.8.4.4"]
   }
   ```

   If your internal DNS server cannot resolve public IP addresses, include at least one DNS server which can, so that you can connect to Docker Hub and so that your containers can resolve internet domain names.

   Save and close the file.

3. Restart the Docker daemon.

   ```
   $ sudo service docker restart
   ```

4. Verify that Docker can resolve external IP addresses by trying to pull an image:

   ```
   $ docker pull hello-world
   ```

5. If necessary, verify that Docker containers can resolve an internal hostname by pinging it.

   ```
   $ docker run --rm -it alpine ping -c4 <my_internal_host>

   PING google.com (192.168.1.2): 56 data bytes
   64 bytes from 192.168.1.2: seq=0 ttl=41 time=7.597 ms
   64 bytes from 192.168.1.2: seq=1 ttl=41 time=7.635 ms
   64 bytes from 192.168.1.2: seq=2 ttl=41 time=7.660 ms
   64 bytes from 192.168.1.2: seq=3 ttl=41 time=7.677 ms
   ```

DISABLE `DNSMASQ`

Ubuntu

If you prefer not to change the Docker daemon’s configuration to use a specific IP address, follow these instructions to disable `dnsmasq` in NetworkManager.

1. Edit the `/etc/NetworkManager/NetworkManager.conf` file.
2. Comment out the `dns=dnsmasq` line by adding a `#` character to the beginning of the line.

   ```
   # dns=dnsmasq
   ```

   Save and close the file.
3. Restart both NetworkManager and Docker. As an alternative, you can reboot your system.

   ```
   $ sudo restart network-manager
   $ sudo restart docker
   ```

RHEL, CentOS, or Fedora

To disable `dnsmasq` on RHEL, CentOS, or Fedora:

1. Disable the `dnsmasq` service:

   ```
   $ sudo service dnsmasq stop

   $ sudo systemctl disable dnsmasq
   ```

2. Configure the DNS servers manually using the Red Hat documentation (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Deployment_Guide/s1-networkscripts-interfaces.html).

Allow access to the remote API through a firewall

If you run a firewall on the same host as you run Docker and you want to access the Docker Remote API from another host and remote access is enabled, you need to configure your firewall to allow incoming connections on the Docker port, which defaults to `2376` if TLS encrypted transport is enabled or `2375` otherwise.

Two common firewall daemons are UFW (Uncomplicated Firewall) (https://help.ubuntu.com/community/UFW) (often used for Ubuntu systems) and firewalld (http://www.firewalld.org/) (often used for RPM-based systems). Consult the documentation for your OS and firewall, but the following information might help you get started. These options are fairly permissive and you may want to use a different configuration that locks your system down more.

- UFW: Set `DEFAULT_FORWARD_POLICY="ACCEPT"` in your configuration.
- firewalld: Add rules similar to the following to your policy (one for incoming requests and one for outgoing requests). Be sure the interface names and chain names are correct.

```
<direct>
  [ <rule ipv="ipv6" table="filter" chain="FORWARD_direct" priority="0"> -
i zt0 -j ACCEPT </rule> ]
  [ <rule ipv="ipv6" table="filter" chain="FORWARD_direct" priority="0"> -
o zt0 -j ACCEPT </rule> ]
</direct>
```

Your kernel does not support cgroup swap limit capabilities

On Ubuntu or Debian hosts, You may see messages similar to the following when working with an image.

```
WARNING: Your kernel does not support swap limit capabilities. Limitation discar
ded.
```

This warning does not occur on RPM-based systems, which enable these capabilities by default.

If you don't need these capabilities, you can ignore the warning. You can enable these capabilities on Ubuntu or Debian by following these instructions. Memory and swap accounting incur an overhead of about 1% of the total available memory and a 10% overall performance degradation, even if Docker is not running.

1. Log into the Ubuntu or Debian host as a user with `sudo` privileges.
2. Edit the `/etc/default/grub` file. Add or edit the `GRUB_CMDLINE_LINUX` line to add the following two key-value pairs:

   ```
   GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"
   ```

   Save and close the file.
3. Update GRUB.

   ```
   $ sudo update-grub
   ```

   If your GRUB configuration file has incorrect syntax, an error occurs. In this case, repeat steps 3 and 4.

The changes take effect when the system is rebooted.

Next steps

- Continue with the User Guide (https://docs.docker.com/engine/userguide/).

Docker (https://docs.docker.com/glossary/?term=Docker), Docker documentation (https://docs.docker.com/glossary/?term=Docker documentation), requirements (https://docs.docker.com/glossary/?term=requirements), apt (https://docs.docker.com/glossary/?term=apt), installation (https://docs.docker.com/glossary/?term=installation), ubuntu (https://docs.docker.com/glossary/?term=ubuntu), install (https://docs.docker.com/glossary/?term=install), uninstall (https://docs.docker.com/glossary/?term=uninstall), upgrade (https://docs.docker.com/glossary/?term=upgrade), update (https://docs.docker.com/glossary/?term=update)

Get Docker EE for CentOS

Estimated reading time: 9 minutes

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

- YUM repository (/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 (/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.

For Docker Community Edition on CentOS, see Get Docker CE for CentOS (https://docs.docker.com/install/linux/docker-ce/centos/).

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 Hub (https://hub.docker.com/my-content).
- Uninstall old versions of Docker.
- Remove old Docker repos from `/etc/yum.repos.d/` .

Architectures and storage drivers

Docker EE supports Centos 64-bit, versions 7.1 and higher (7.1, 7.2, 7.3, 7.4), running on `x86_64` .

On Centos, Docker EE supports storage drivers, `overlay2` and `devicemapper`. In Docker EE 17.06.2-ee-5 and higher, `overlay2` is the recommended storage driver. The following limitations apply:

- OverlayFS (https://docs.docker.com/storage/storagedriver/overlayfs-driver): If `selinux` is enabled, the `overlay2` storage driver is supported on CentOS 7.4 or higher. If `selinux` is disabled, `overlay2` is supported on CentOS 7.2 or higher with kernel version 3.10.0-693 and higher.
- Device Mapper (https://docs.docker.com/storage/storagedriver/device-mapper-driver/): On production systems using `devicemapper`, you must use `direct-lvm` mode, which requires one or more dedicated block devices. Fast storage such as solid-state media (SSD) is recommended. Do not start Docker until properly configured per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

Find your Docker EE repo URL

To install Docker EE, you will need the URL of the Docker EE repository associated with your trial or subscription:

1. Go to https://hub.docker.com/my-content (https://hub.docker.com/my-content). All of your subscriptions and trials are listed.
2. Click the Setup button for Docker Enterprise Edition for Centos.
3. Copy the URL from Copy and paste this URL to download your Edition and save it for later use.

You will use this URL in a later step to create a variable called, `DOCKERURL`.

Uninstall old Docker versions

The Docker EE package is called `docker-ee`. Older versions were called `docker` or `docker-engine`. Uninstall all older versions and associated dependencies. The contents of `/var/lib/docker/` are preserved, including images, containers, volumes, and networks. If you are upgrading from Docker CE to Docker EE, remove the Docker CE package as well.

```
$ 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 \
                  docker-ce
```

Repo install and upgrade

The advantage of using a repository from which to install Docker EE (or any software) is that it provides a certain level of automation. RPM-based distributions such as Centos, use a tool called YUM that work with your repositories to manage dependencies and provide automatic updates.

Set up the repository

You only need to set up the repository once, after which you can install Docker EE *from* the repo and repeatedly upgrade as necessary.

1. Remove existing Docker repositories from `/etc/yum.repos.d/` :

   ```
   $ sudo rm /etc/yum.repos.d/docker*.repo
   ```

2. Temporarily store the URL (that you copied above (/install/linux/docker-ee/centos/#find-your-docker-ee-repo-url)) in an environment variable. Replace `<DOCKER-EE-URL>` with your URL in the following command. This variable assignment does not persist when the session ends:

   ```
   $ export DOCKERURL="<DOCKER-EE-URL>"
   ```

3. Store the value of the variable, `DOCKERURL` (from the previous step), in a `yum` variable in `/etc/yum/vars/` :

```
$ sudo -E sh -c 'echo "$DOCKERURL/centos" > /etc/yum/vars/docke
rurl'
```

4. 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
```

5. Add the Docker EE stable repository:

```
$ sudo -E yum-config-manager \
    --add-repo \
    "$DOCKERURL/centos/docker-ee.repo"
```

Install from the repository

✓ *NOTE:* If you need to run Docker EE 2.0, please see the following instructions:

- 18.03 (https://docs.docker.com/v18.03/ee/supported-platforms/) - Older Docker EE Engine only release
- 17.06 (https://docs.docker.com/v17.06/engine/installation/) - Docker Enterprise Edition 2.0 (Docker Engine, UCP, and DTR).

1. Install the latest patch release, or go to the next step to install a specific version:

```
$ sudo yum -y install docker-ee
```

If prompted to accept the GPG key, verify that the fingerprint matches `77FE DA13 1A83 1D29 A418 D3E8 99E5 FF2E 7668 2BC9` , and if so, accept it.

2. To install a *specific version* of Docker EE (recommended in production), list versions and install:

a. List and sort the versions available in your repo. This example sorts results by version number, highest to lowest, and is truncated:

```
$ sudo yum list docker-ee  --showduplicates | sort -r

docker-ee.x86_64      18.09.ee.2-1.el7.centos      docker-ee-st
able-18.09
```

The list returned depends on which repositories you enabled, and is specific to your version of Centos (indicated by `.el7` in this example).

b. Install a specific version by its fully qualified package name which is the package name (`docker-ee`) plus the version string (2nd column) up to the hyphen, for example: `docker-ee-18.09.0`

```
$ sudo yum -y install <FULLY-QUALIFIED-PACKAGE-NAME>
```

For example, if you want to install the 18.09 version run the following:

```
sudo yum-config-manager --enable docker-ee-stable-18.09
```

Docker is installed but not started. The `docker` group is created, but no users are added to the group.

3. Start Docker:

> If using `devicemapper` , ensure it is properly configured before starting Docker, per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

```
$ sudo systemctl start docker
```

4. Verify that Docker EE is installed correctly by running the `hello-world` image. This command downloads a test image, runs it in a container, prints an informational message, and exits:

```
$ sudo docker run hello-world
```

Docker EE is installed and running. Use `sudo` to run Docker commands. See Linux postinstall (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands.

Upgrade from the repository

1. Add the new repository (/install/linux/docker-ee/centos/#set-up-the-repository).
2. Follow the installation instructions (/install/linux/docker-ee/centos/#install-from-the-repository) and install a new version.

Package install and upgrade

To manually install Docker EE, download the `.rpm` file for your release. You need to download a new file each time you want to upgrade Docker EE.

Install with a package

1. Go to the Docker EE repository URL associated with your trial or subscription in your browser. Go to `centos/7/x86_64/stable-<VERSION>/Packages` and download the `.rpm` file for the Docker version you want to install.
2. Install Docker EE, changing the path below to the path where you downloaded the Docker package.

   ```
   $ sudo yum install /path/to/package.rpm
   ```

   Docker is installed but not started. The `docker` group is created, but no users are added to the group.
3. Start Docker:

> If using `devicemapper` , ensure it is properly configured before starting Docker, per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

```
$ sudo systemctl start docker
```

4. Verify that Docker EE is installed correctly by running the `hello-world` image. This command downloads a test image, runs it in a container, prints an informational message, and exits:

   ```
   $ sudo docker run hello-world
   ```

   Docker EE is installed and running. Use `sudo` to run Docker commands. See Linux postinstall (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands.

Upgrade with a package

1. Download the newer package file.
2. Repeat the installation procedure (/install/linux/docker-ee/centos/#install-with-a-package), using `yum -y upgrade` instead of `yum -y install` , and point to the new file.

Uninstall Docker EE

1. Uninstall the Docker EE package:

   ```
   $ sudo yum -y remove docker-ee
   ```

2. Delete all images, containers, and volumes (because these are not automatically removed from your host):

```
$ sudo rm -rf /var/lib/docker
```

3. Delete other Docker related resources:

```
$ sudo rm -rf /run/docker
$ sudo rm -rf /var/run/docker
$ sudo rm -rf /etc/docker
```

4. If desired, remove the `devicemapper` thin pool and reformat the block devices that were part of it.

You must delete any edited configuration files manually.

Next steps

- Continue to Post-installation steps for Linux (https://docs.docker.com/install/linux/linux-postinstall/)
- Continue with user guides on Universal Control Plane (UCP) (https://docs.docker.com/ee/ucp/) and Docker Trusted Registry (DTR) (https://docs.docker.com/ee/dtr/)

requirements (https://docs.docker.com/glossary/?term=requirements), apt (https://docs.docker.com/glossary/?term=apt), installation (https://docs.docker.com/glossary/?term=installation), centos (https://docs.docker.com/glossary/?term=centos), rpm (https://docs.docker.com/glossary/?term=rpm), install (https://docs.docker.com/glossary/?term=install), uninstall (https://docs.docker.com/glossary/?term=uninstall), upgrade (https://docs.docker.com/glossary/?term=upgrade), update (https://docs.docker.com/glossary/?term=update)

Get Docker EE for Oracle Linux

Estimated reading time: 8 minutes

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

- YUM repository (/install/linux/docker-ee/oracle/#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 (/install/linux/docker-ee/oracle/#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.

Docker Community Edition (Docker CE) is *not* supported on Oracle Linux.

Prerequisites

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

- Use OL 64-bit 7.3 or higher on RHCK 3.10.0-514 or higher.
- Use the `devicemapper` storage driver only (`direct-lvm` mode in production).
- Find the URL for your Docker EE repo at Docker Hub (https://hub.docker.com/my-content).
- Uninstall old versions of Docker.
- Remove old Docker repos from `/etc/yum.repos.d/` .
- Disable SELinux if installing or upgrading Docker EE 17.06.1 or newer.

Architectures and storage drivers

Docker EE supports Oracle Linux 64-bit, versions 7.3 and higher, running the Red Hat Compatible kernel (RHCK) 3.10.0-514 or higher. Older versions of Oracle Linux are not supported.

On Oracle Linux, Docker EE only supports the `devicemapper` storage driver. In production, you must use it in `direct-lvm` mode, which requires one or more dedicated block devices. Fast storage such as solid-state media (SSD) is recommended. Do not start Docker until properly configured per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

Find your Docker EE repo URL

To install Docker EE, you will need the URL of the Docker EE repository associated with your trial or subscription:

1. Go to https://hub.docker.com/my-content (https://hub.docker.com/my-content). All of your subscriptions and trials are listed.
2. Click the Setup button for Docker Enterprise Edition for Oracle Linux.
3. Copy the URL from Copy and paste this URL to download your Edition and save it for later use.

You will use this URL in a later step to create a variable called, `DOCKERURL` .

Uninstall old Docker versions

The Docker EE package is called `docker-ee` . Older versions were called `docker` or `docker-engine` . Uninstall all older versions and associated dependencies. The contents of `/var/lib/docker/` are preserved, including images, containers, volumes, and networks.

```
$ sudo yum remove docker \
                  docker-engine \
                  docker-engine-selinux
```

Repo install and upgrade

The advantage of using a repository from which to install Docker EE (or any software) is that it provides a certain level of automation. RPM-based distributions such as Oracle Linux, use a tool called YUM that work with your repositories to manage dependencies and provide automatic updates.

Set up the repository

You only need to set up the repository once, after which you can install Docker EE *from* the repo and repeatedly upgrade as necessary.

1. Remove existing Docker repositories from `/etc/yum.repos.d/` :

   ```
   $ sudo rm /etc/yum.repos.d/docker*.repo
   ```

2. Temporarily store the URL (that you copied above (/install/linux/docker-ee/oracle/#find-your-docker-ee-repo-url)) in an environment variable. Replace `<DOCKER-EE-URL>` with your URL in the following command. This variable assignment does not persist when the session ends:

   ```
   $ export DOCKERURL="<DOCKER-EE-URL>"
   ```

3. Store the value of the variable, `DOCKERURL` (from the previous step), in a `yum` variable in `/etc/yum/vars/` :

   ```
   $ sudo -E sh -c 'echo "$DOCKERURL/oraclelinux" > /etc/yum/vars/
   dockerurl'
   ```

4. 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
   ```

5. Enable the `ol7_addons` Oracle repository. This ensures access to the `container-selinux` package required by `docker-ee` .

   ```
   $ sudo yum-config-manager --enable ol7_addons
   ```

6. Add the Docker EE stable repository:

```
$ sudo -E yum-config-manager \
    --add-repo \
    "$DOCKERURL/oraclelinux/docker-ee.repo"
```

Install from the repository

NOTE: If you need to run Docker EE 2.0, please see the following instructions:

- 18.03 (https://docs.docker.com/v18.03/ee/supported-platforms/) - Older Docker EE Engine only release
- 17.06 (https://docs.docker.com/v17.06/engine/installation/) - Docker Enterprise Edition 2.0 (Docker Engine, UCP, and DTR).

1. Install the latest patch release, or go to the next step to install a specific version:

   ```
   $ sudo yum -y install docker-ee
   ```

   If prompted to accept the GPG key, verify that the fingerprint matches `77FE DA13 1A83 1D29 A418 D3E8 99E5 FF2E 7668 2BC9` , and if so, accept it.

2. To install a *specific version* of Docker EE (recommended in production), list versions and install:

   a. List and sort the versions available in your repo. This example sorts results by version number, highest to lowest, and is truncated:

   ```
   $ sudo yum list docker-ee  --showduplicates | sort -r

   docker-ee.x86_64      18.09.ee.2-1.el7.oraclelinux      docker-
   ee-stable-18.09
   ```

   The list returned depends on which repositories you enabled, and is specific to your version of Oracle Linux (indicated by `.el7` in this example).

b. Install a specific version by its fully qualified package name which is the package name (`docker-ee`) plus the version string (2nd column) up to the hyphen, for example: `docker-ee-18.09.0`

```
$ sudo yum -y install <FULLY-QUALIFIED-PACKAGE-NAME>
```

For example, if you want to install the 18.09 version run the following:

```
sudo yum-config-manager --enable docker-ee-stable-18.09
```

Docker is installed but not started. The `docker` group is created, but no users are added to the group.

3. Start Docker:

   > If using `devicemapper` , ensure it is properly configured before starting Docker, per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

   ```
   $ sudo systemctl start docker
   ```

4. Verify that Docker EE is installed correctly by running the `hello-world` image. This command downloads a test image, runs it in a container, prints an informational message, and exits:

   ```
   $ sudo docker run hello-world
   ```

   Docker EE is installed and running. Use `sudo` to run Docker commands. See Linux postinstall (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands.

Upgrade from the repository

1. Add the new repository (/install/linux/docker-ee/oracle/#set-up-the-

repository).

2. Follow the installation instructions (/install/linux/docker-ee/oracle/#install-from-the-repository) and install a new version.

Package install and upgrade

To manually install Docker EE, download the `.rpm` file for your release. You need to download a new file each time you want to upgrade Docker EE.

Install with a package

1. Go to the Docker EE repository URL associated with your trial or subscription in your browser. Go to `oraclelinux/`. Choose your Oracle Linux version, architecture, and Docker version. Download the `.rpm` file from the `Packages` directory.

2. Install Docker EE, changing the path below to the path where you downloaded the Docker package.

   ```
   $ sudo yum install /path/to/package.rpm
   ```

   Docker is installed but not started. The `docker` group is created, but no users are added to the group.

3. Start Docker:

   > If using `devicemapper`, ensure it is properly configured before starting Docker, per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

   ```
   $ sudo systemctl start docker
   ```

4. Verify that Docker EE is installed correctly by running the `hello-world` image. This command downloads a test image, runs it in a container, prints an informational message, and exits:

```
$ sudo docker run hello-world
```

Docker EE is installed and running. Use `sudo` to run Docker commands. See Linux postinstall (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands.

Upgrade with a package

1. Download the newer package file.
2. Repeat the installation procedure (/install/linux/docker-ee/oracle/#install-with-a-package), using `yum -y upgrade` instead of `yum -y install` , and point to the new file.

Uninstall Docker EE

1. Uninstall the Docker EE package:

   ```
   $ sudo yum -y remove docker-ee
   ```

2. Delete all images, containers, and volumes (because these are not automatically removed from your host):

   ```
   $ sudo rm -rf /var/lib/docker
   ```

3. Delete other Docker related resources:

   ```
   $ sudo rm -rf /run/docker
   $ sudo rm -rf /var/run/docker
   $ sudo rm -rf /etc/docker
   ```

4. If desired, remove the `devicemapper` thin pool and reformat the block devices that were part of it.

You must delete any edited configuration files manually.

Next steps

- Continue to Post-installation steps for Linux (https://docs.docker.com/install/linux/linux-postinstall/)
- Continue with user guides on Universal Control Plane (UCP) (https://docs.docker.com/ee/ucp/) and Docker Trusted Registry (DTR) (https://docs.docker.com/ee/dtr/)

requirements (https://docs.docker.com/glossary/?term=requirements), installation (https://docs.docker.com/glossary/?term=installation), oracle (https://docs.docker.com/glossary/?term=oracle), ol (https://docs.docker.com/glossary/?term=ol), rpm (https://docs.docker.com/glossary/?term=rpm), install (https://docs.docker.com/glossary/?term=install), uninstall (https://docs.docker.com/glossary/?term=uninstall), upgrade (https://docs.docker.com/glossary/?term=upgrade), update (https://docs.docker.com/glossary/?term=update)

Get Docker EE for Red Hat Enterprise Linux

Estimated reading time: 13 minutes

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

- YUM repository (/install/linux/docker-ee/rhel/#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 (/install/linux/docker-ee/rhel/#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.

Docker Community Edition (Docker CE) is *not* supported on Red Hat Enterprise Linux.

Prerequisites

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

- Use RHEL 64-bit 7.1 and higher on `x86_64` , `s390x` , or `ppc64le` (not ppc64).
- Use storage driver `overlay2` or `devicemapper` (`direct-lvm` mode in production).
- Find the URL for your Docker EE repo at Docker Hub (https://hub.docker.com/my-content).
- Uninstall old versions of Docker.
- Remove old Docker repos from `/etc/yum.repos.d/` .
- Disable SELinux on `s390x` (IBM Z) systems before install/upgrade.

Architectures and storage drivers

Docker EE supports Red Hat Enterprise Linux 64-bit, versions 7.1 and higher (7.1, 7.2, 7.3, 7.4, 7.5), running on one of the following architectures: `x86_64` , `s390x` (IBM Z), or `ppc64le` (IBM Power, little endian format). To ensure you have `ppc64le` (and not `ppc64`), run the command, `uname -m` .

✔ Little endian format only

On IBM Power systems, Docker EE only supports little endian format, `ppc64le` , even though RHEL 7 ships both big and little endian versions.

On Red Hat Enterprise Linux, Docker EE supports storage drivers, `overlay2` and `devicemapper` . In Docker EE 17.06.2-ee-5 and higher, `overlay2` is the recommended storage driver. The following limitations apply:

- OverlayFS (https://docs.docker.com/storage/storagedriver/overlayfs-driver): If `selinux` is enabled, the `overlay2` storage driver is supported on RHEL 7.4 or higher. If `selinux` is disabled, `overlay2` is supported on RHEL 7.2 or higher with kernel version 3.10.0-693 and higher.
- Device Mapper (https://docs.docker.com/storage/storagedriver/device-mapper-driver/): On production systems using `devicemapper` , you must use `direct-lvm` mode, which requires one or more dedicated block devices. Fast storage such as solid-state media (SSD) is recommended. Do not start Docker until properly configured per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

FIPS 140-2 cryptographic module support

Federal Information Processing Standards (FIPS) Publication 140-2 (https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf) is a United States Federal security requirement for cryptographic modules.

With Docker EE Basic license for versions 18.03 and later, Docker provides FIPS 140-2 support in RHEL 7.3, 7.4 and 7.5. This includes a FIPS supported cryptographic module. If the RHEL implementation already has FIPS support enabled, FIPS is automatically enabled in the Docker engine.

To verify the FIPS-140-2 module is enabled in the Linux kernel, confirm the file `/proc/sys/crypto/fips_enabled` contains `1` .

```
$ cat /proc/sys/crypto/fips_enabled
1
```

NOTE: FIPS is only supported in the Docker EE engine. UCP and DTR currently do not have support for FIPS-140-2.

To enable FIPS 140-2 compliance on a system that is not in FIPS 140-2 mode, do the following:

Create a file called `/etc/systemd/system/docker.service.d/fips-module.conf` . It needs to contain the following:

```
[Service]
Environment="DOCKER_FIPS=1"
```

Reload the Docker configuration to systemd.

```
$ sudo systemctl daemon-reload
```

Restart the Docker service as root.

```
$ sudo systemctl restart docker
```

To confirm Docker is running with FIPS-140-2 enabled, run the `docker info` command:

```
docker info --format {{.SecurityOptions}}
[name=selinux name=fips]
```

Disabling FIPS-140-2

If the system has the FIPS 140-2 cryptographic module installed on the operating system, it is possible to disable FIPS-140-2 compliance.

To disable FIPS 140-2 in Docker but not the operating system, set the value `DOCKER_FIPS=0` in the `/etc/systemd/system/docker.service.d/fips-module.conf` .

Reload the Docker configuration to systemd.

```
$ sudo systemctl daemon-reload
```

Restart the Docker service as root.

```
$ sudo systemctl restart docker
```

Find your Docker EE repo URL

To install Docker EE, you will need the URL of the Docker EE repository associated with your trial or subscription:

1. Go to https://hub.docker.com/my-content (https://hub.docker.com/my-content). All of your subscriptions and trials are listed.
2. Click the Setup button for Docker Enterprise Edition for Red Hat Enterprise Linux.
3. Copy the URL from Copy and paste this URL to download your Edition and save it for later use.

You will use this URL in a later step to create a variable called, `DOCKERURL` .

Uninstall old Docker versions

The Docker EE package is called `docker-ee` . Older versions were called `docker` or `docker-engine` . Uninstall all older versions and associated dependencies. The contents of `/var/lib/docker/` are preserved, including images, containers, volumes, and networks.

```
$ 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 \
                  docker-ce
```

Repo install and upgrade

The advantage of using a repository from which to install Docker EE (or any software) is that it provides a certain level of automation. RPM-based distributions such as Red Hat Enterprise Linux, use a tool called YUM that work with your repositories to manage dependencies and provide automatic updates.

Disable SELinux before installing Docker EE on IBM Z systems

There is currently no support for `selinux` on IBM Z systems. If you attempt to install or upgrade Docker EE on an IBM Z system with `selinux` enabled, an error is thrown that the `container-selinux` package is not found. Disable `selinux` before installing or upgrading Docker on IBM Z.

Set up the repository

You only need to set up the repository once, after which you can install Docker EE *from* the repo and repeatedly upgrade as necessary.

1. Remove existing Docker repositories from `/etc/yum.repos.d/` :

   ```
   $ sudo rm /etc/yum.repos.d/docker*.repo
   ```

2. Temporarily store the URL (that you copied above (/install/linux/docker-ee/rhel/#find-your-docker-ee-repo-url)) in an environment variable. Replace `<DOCKER-EE-URL>` with your URL in the following command. This variable assignment does not persist when the session ends:

   ```
   $ export DOCKERURL="<DOCKER-EE-URL>"
   ```

3. Store the value of the variable, `DOCKERURL` (from the previous step), in a `yum` variable in `/etc/yum/vars/` :

   ```
   $ sudo -E sh -c 'echo "$DOCKERURL/rhel" > /etc/yum/vars/dockeru
   rl'
   ```

   Also, store your OS version string in `/etc/yum/vars/dockerosversion` . Most users should use `7` , but you can also use the more specific minor version, starting from `7.2` .

   ```
   $ sudo sh -c 'echo "7" > /etc/yum/vars/dockerosversion'
   ```

4. 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
   ```

5. Enable the `extras` RHEL repository. This ensures access to the `container-selinux` package required by `docker-ee` .

   The repository can differ per your architecture and cloud provider, so review the options in this step before running:

   For all architectures *except* IBM Power:

   ```
   $ sudo yum-config-manager --enable rhel-7-server-extras-rpms
   ```

   For IBM Power only (little endian):

   ```
   $ sudo yum-config-manager --enable extras
   $ sudo subscription-manager repos --enable=rhel-7-for-power-le-
   extras-rpms
   $ sudo yum makecache fast
   $ sudo yum -y install container-selinux
   ```

   Depending on cloud provider, you may also need to enable another repository:

   For AWS (where `REGION` is a literal, and does *not* represent the region your machine is running in):

   ```
   $ sudo yum-config-manager --enable rhui-REGION-rhel-server-extr
   as
   ```

   For Azure:

```
$ sudo yum-config-manager --enable rhui-rhel-7-server-rhui-extr
as-rpms
```

6. Add the Docker EE stable repository:

```
$ sudo -E yum-config-manager \
    --add-repo \
    "$DOCKERURL/rhel/docker-ee.repo"
```

Install from the repository

✔ *NOTE:* If you need to run Docker EE 2.0, please see the following instructions:

- 18.03 (https://docs.docker.com/v18.03/ee/supported-platforms/) - Older Docker EE Engine only release
- 17.06 (https://docs.docker.com/v17.06/engine/installation/) - Docker Enterprise Edition 2.0 (Docker Engine, UCP, and DTR).

1. Install the latest patch release, or go to the next step to install a specific version:

   ```
   $ sudo yum -y install docker-ee
   ```

   If prompted to accept the GPG key, verify that the fingerprint matches `77FE DA13 1A83 1D29 A418 D3E8 99E5 FF2E 7668 2BC9` , and if so, accept it.

2. To install a *specific version* of Docker EE (recommended in production), list versions and install:

   a. List and sort the versions available in your repo. This example sorts results by version number, highest to lowest, and is truncated:

   ```
   $ sudo yum list docker-ee  --showduplicates | sort -r

   docker-ee.x86_64      18.09.ee.2-1.el7.rhel      docker-ee-stab
   le-18.09
   ```

The list returned depends on which repositories you enabled, and is specific to your version of Red Hat Enterprise Linux (indicated by `.el7` in this example).

b. Install a specific version by its fully qualified package name which is the package name (`docker-ee`) plus the version string (2nd column) up to the hyphen, for example: `docker-ee-18.09.0`

```
$ sudo yum -y install <FULLY-QUALIFIED-PACKAGE-NAME>
```

For example, if you want to install the 18.09 version run the following:

```
sudo yum-config-manager --enable docker-ee-stable-18.09
```

Docker is installed but not started. The `docker` group is created, but no users are added to the group.

3. Start Docker:

   > If using `devicemapper` , ensure it is properly configured before starting Docker, per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

   ```
   $ sudo systemctl start docker
   ```

4. Verify that Docker EE is installed correctly by running the `hello-world` image. This command downloads a test image, runs it in a container, prints an informational message, and exits:

   ```
   $ sudo docker run hello-world
   ```

   Docker EE is installed and running. Use `sudo` to run Docker commands. See Linux postinstall (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands.

Upgrade from the repository

1. Add the new repository (/install/linux/docker-ee/rhel/#set-up-the-repository).
2. Follow the installation instructions (/install/linux/docker-ee/rhel/#install-from-the-repository) and install a new version.

Package install and upgrade

To manually install Docker EE, download the `.rpm` file for your release. You need to download a new file each time you want to upgrade Docker EE.

> Disable SELinux before installing Docker EE on IBM Z systems
>
> There is currently no support for `selinux` on IBM Z systems. If you attempt to install or upgrade Docker EE on an IBM Z system with `selinux` enabled, an error is thrown that the `container-selinux` package is not found. Disable `selinux` before installing or upgrading Docker on IBM Z.

Install with a package

1. Enable the `extras` RHEL repository. This ensures access to the `container-selinux` package which is required by `docker-ee` :

   ```
   $ sudo yum-config-manager --enable rhel-7-server-extras-rpms
   ```

   Alternately, obtain that package manually from Red Hat. There is no way to publicly browse this repository.

2. Go to the Docker EE repository URL associated with your trial or subscription in your browser. Go to `rhel/` . Choose your Red Hat Enterprise Linux version, architecture, and Docker version. Download the `.rpm` file from the `Packages` directory.

   > If you have trouble with `selinux` using the packages under the `7` directory, try choosing the version-specific directory instead, such as `7.3` .

3. Install Docker EE, changing the path below to the path where you downloaded the Docker package.

   ```
   $ sudo yum install /path/to/package.rpm
   ```

   Docker is installed but not started. The `docker` group is created, but no users are added to the group.

4. Start Docker:

   > If using `devicemapper`, ensure it is properly configured before starting Docker, per the storage guide (https://docs.docker.com/storage/storagedriver/device-mapper-driver/).

   ```
   $ sudo systemctl start docker
   ```

5. Verify that Docker EE is installed correctly by running the `hello-world` image. This command downloads a test image, runs it in a container, prints an informational message, and exits:

   ```
   $ sudo docker run hello-world
   ```

   Docker EE is installed and running. Use `sudo` to run Docker commands. See Linux postinstall (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands.

Upgrade with a package

1. Download the newer package file.
2. Repeat the installation procedure (/install/linux/docker-ee/rhel/#install-with-a-package), using `yum -y upgrade` instead of `yum -y install`, and point to the new file.

Uninstall Docker EE

1. Uninstall the Docker EE package:

   ```
   $ sudo yum -y remove docker-ee
   ```

2. Delete all images, containers, and volumes (because these are not automatically removed from your host):

   ```
   $ sudo rm -rf /var/lib/docker
   ```

3. Delete other Docker related resources:

   ```
   $ sudo rm -rf /run/docker
   $ sudo rm -rf /var/run/docker
   $ sudo rm -rf /etc/docker
   ```

4. If desired, remove the `devicemapper` thin pool and reformat the block devices that were part of it.

You must delete any edited configuration files manually.

Next steps

- Continue to Post-installation steps for Linux (https://docs.docker.com/install/linux/linux-postinstall/)
- Continue with user guides on Universal Control Plane (UCP) (https://docs.docker.com/ee/ucp/) and Docker Trusted Registry (DTR) (https://docs.docker.com/ee/dtr/)

requirements (https://docs.docker.com/glossary/?term=requirements), installation (https://docs.docker.com/glossary/?term=installation), rhel (https://docs.docker.com/glossary/?term=rhel), rpm (https://docs.docker.com/glossary/?term=rpm), install (https://docs.docker.com/glossary/?term=install), uninstall (https://docs.docker.com/glossary/?term=uninstall), upgrade (https://docs.docker.com/glossary/?term=upgrade), update (https://docs.docker.com/glossary/?term=update)

Get Docker EE for SLES

Estimated reading time: 12 minutes

To get started with Docker on SUSE Linux Enterprise Server (SLES), make sure you meet the prerequisites (/install/linux/docker-ee/suse/#prerequisites), then install Docker (/install/linux/docker-ee/suse/#install-docker-ee).

Prerequisites

Docker EE URL

To install Docker Enterprise Edition (Docker EE), you need to know the Docker EE repository URL associated with your trial or subscription. These instructions work for Docker EE for SLES and for Docker EE for Linux, which includes access to Docker EE for all Linux distributions. To get this information:

- Go to https://hub.docker.com/my-content (https://hub.docker.com/my-content).
- Each subscription or trial you have access to is listed. Click the Setup button for Docker Enterprise Edition for SUSE Linux Enterprise Server.
- Copy the URL from the field labeled Copy and paste this URL to download your Edition.

Use this URL when you see the placeholder text `<DOCKER-EE-URL>` .

To learn more about Docker EE, see Docker Enterprise Edition (https://www.docker.com/enterprise-edition/).

Docker Community Edition (Docker CE) is not supported on SLES.

OS requirements

To install Docker EE, you need the 64-bit version of SLES 12.x, running on `x86_64` , `s390x` (IBM Z), or `ppc64le` (IBM Power) architectures. Docker EE is not supported on OpenSUSE.

The only supported storage driver for Docker EE on SLES is Btrfs, which is used by default if the underlying filesystem hosting `/var/lib/docker/` is a BTRFS filesystem.

FIREWALL CONFIGURATION

Docker creates a `DOCKER` iptables chain when it starts. The SUSE firewall may block access to this chain, which can prevent you from running containers with published ports. You may see errors such as the following:

```
WARNING: IPv4 forwarding is disabled. Networking will not work.
docker: Error response from daemon: driver failed programming extern
al
        connectivity on endpoint adoring_ptolemy
        (0bb5fa80bc476f8a0d343973929bb3b7c039fc6d7cd30817e837bc2a511
fce97):
        (iptables failed: iptables --wait -t nat -A DOCKER -p tcp -d
 0/0 --dport 80 -j DNAT --to-destination 172.17.0.2:80 ! -i docker0:
 iptables: No chain/target/match by that name.
 (exit status 1)).
```

If you see errors like this, adjust the start-up script order so that the firewall is started before Docker, and Docker stops before the firewall stops. See the SLES documentation on init script order (https://www.suse.com/documentation/sled11/book_sle_admin/data/sec_boot_init.html).

Uninstall old versions

Older versions of Docker were called `docker` or `docker-engine`. If you use OS images from a cloud provider, you may need to remove the `runc` package, which conflicts with Docker EE. If these are installed, uninstall them, along with associated dependencies.

```
$ sudo zypper rm docker docker-engine runc
```

If removal of the `docker-engine` package fails, use the following command instead:

```
$ sudo rpm -e docker-engine
```

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

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

Configure the Btrfs filesystem

By default, SLES formats the `/` filesystem using Btrfs, so most people do not not need to do the steps in this section. If you use OS images from a cloud provider, you may need to do this step. If the filesystem that hosts `/var/lib/docker/` is not a BTRFS filesystem, you must configure a BTRFS filesystem and mount it on `/var/lib/docker/`.

1. Check whether `/` (or `/var/` or `/var/lib/` or `/var/lib/docker/` if they are separate mount points) are formatted using Btrfs. If you do not have separate mount points for any of these, a duplicate result for `/` is returned.

   ```
   $ df -T / /var /var/lib /var/lib/docker
   ```

   You need to complete the rest of these steps only if one of the following is true:

   - You have a separate `/var/` filesystem that is not formatted with Btrfs
   - You do not have a separate `/var/` or `/var/lib/` or `/var/lib/docker/` filesystem and `/` is not formatted with Btrfs

   If `/var/lib/docker` is already a separate mount point and is not formatted with Btrfs, back up its contents so that you can restore them after step 3.

2. Format your dedicated block device or devices as a Btrfs filesystem. This example assumes that you are using two block devices called `/dev/xvdf` and `/dev/xvdg`. Make sure you are using the right device names.

   Double-check the block device names because this is a destructive operation.

```
$ sudo mkfs.btrfs -f /dev/xvdf /dev/xvdg
```

There are many more options for Btrfs, including striping and RAID. See the Btrfs documentation (https://btrfs.wiki.kernel.org/index.php/Using_Btrfs_with_Multiple_Devices).

3. Mount the new Btrfs filesystem on the `/var/lib/docker/` mount point. You can specify any of the block devices used to create the Btrfs filesystem.

   ```
   $ sudo mount -t btrfs /dev/xvdf /var/lib/docker
   ```

   Don't forget to make the change permanent across reboots by adding an entry to `/etc/fstab` .

4. If `/var/lib/docker` previously existed and you backed up its contents during step 1, restore them onto `/var/lib/docker` .

Install Docker EE

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

- Most users set up Docker's repositories (/install/linux/docker-ee/suse/#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 and manage upgrades completely manually. This is useful in situations such as installing Docker on air-gapped systems with no access to the internet.

Install using the repository

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

NOTE: If you need to run Docker EE 2.0, please see the following instructions:

- 18.03 (https://docs.docker.com/v18.03/ee/supported-platforms/) - Older Docker EE Engine only release
- 17.06 (https://docs.docker.com/v17.06/engine/installation/) - Docker Enterprise Edition 2.0 (Docker Engine, UCP, and DTR).

SET UP THE REPOSITORY

1. Temporarily add a `$DOCKER_EE_URL` variable into your environment. This only persists until you log out of the session. Replace `<DOCKER-EE-URL>` with the URL you noted down in the prerequisites (/install/linux/docker-ee/suse/#prerequisites).

   ```
   $ DOCKER_EE_URL="<DOCKER-EE-URL>/sles/12.3/<ARCHITECTURE>/stabl
   e-<VERSION>"
   ```

   Where:

   - `DOCKER-EE-URL` is the URL from your Docker Hub subscription.
   - `ARCHITECTURE` is `x86_64` , `s390x` , or `ppc64le` .
   - `VERSION` is `18.09`

   As an example your command should look like:

   ```
   DOCKER_EE_URL="https://storebits.docker.com/ee/sles/sub-555-55-
   555/sles/12.3/x86_64/stable-18.09"
   ```

2. Use the following command to set up the stable repository. Use the command as-is. It works because of the variable you set in the previous step.

   ```
   $ sudo zypper addrepo $DOCKER_EE_URL docker-ee-stable
   ```

3. Import the GPG key from the repository. Replace `<DOCKER-EE-URL>` with the URL you noted down in the prerequisites (/install/linux/docker-ee/suse/#prerequisites).

```
$ sudo rpm --import "<DOCKER-EE-URL>/sles/gpg"
```

INSTALL DOCKER EE

1. Update the `zypper` package index.

   ```
   $ sudo zypper refresh
   ```

   If this is the first time you have refreshed the package index since adding the Docker repositories, you are prompted to accept the GPG key, and the key's fingerprint is shown. Verify that the fingerprint matches `77FE DA13 1A83 1D29 A418 D3E8 99E5 FF2E 7668 2BC9` and if so, accept the key.

2. Install the latest version of Docker EE, or go to the next step to install a specific version.

   ```
   $ sudo zypper install docker-ee
   ```

   Start Docker:

   ```
   $ sudo service docker start
   ```

3. On production systems, you should install a specific version of Docker EE instead of always using the latest. List the available versions. The following example only lists binary packages and is truncated. To also list source packages, omit the `-t package` flag from the command.

```
$ zypper search -s --match-exact -t package docker-ee

  Loading repository data...
  Reading installed packages...

  S | Name           | Type    | Version
       | Arch    | Repository
  --+---------------+---------+-------------------------------
-------+--------+---------------
    | docker-ee     | package | 18.09-1                 | x86_6
4 | docker-ee-stable
```

The contents of the list depend upon which repositories you have enabled. Choose a specific version to install. The third column is the version string. The fifth column is the repository name, which indicates which repository the package is from and by extension its stability level. To install a specific version, append the version string to the package name and separate them by a hyphen (`-`):

```
$ sudo zypper install docker-ee-<VERSION_STRING>
```

Docker is installed but not started. The `docker` group is created, but no users are added to the group.

4. Configure Docker EE to use the Btrfs filesystem. This is only required if the `/` filesystem is not using BTRFS. However, explicitly specifying the `storage-driver` has no harmful side effects.

   Edit the file `/etc/docker/daemon.json` (create it if it does not exist) and add the following contents:

   ```
   {
     "storage-driver": "btrfs"
   }
   ```

   Save and close the file.

5. Start Docker:

```
$ sudo service docker start
```

6. Verify that Docker EE is installed correctly by running the `hello-world` image.

   ```
   $ sudo docker run hello-world
   ```

   This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

Docker EE is installed and running. You need to use `sudo` to run Docker commands. Continue to Linux postinstall (https://docs.docker.com/install/linux/linux-postinstall/) to configure the graph storage driver, allow non-privileged users to run Docker commands, and for other optional configuration steps.

> Important: Be sure Docker is configured to start after the system firewall. See Firewall configuration (/install/linux/docker-ee/suse/#firewall-configuration).

UPGRADE DOCKER EE

To upgrade Docker EE:

1. If upgrading to a new major Docker EE version (such as when going from Docker 18.03.x to Docker 18.09.x), add the new repository (/install/linux/docker-ee/suse/#set-up-the-repository).
2. Run `sudo zypper refresh` .
3. Follow the installation instructions (/install/linux/docker-ee/suse/#install-docker-ee), choosing the new version you want to install.

Install from a package

If you cannot use the official Docker repository to install Docker EE, you can download the `.rpm` file for your release and install it manually. You need to download a new file each time you want to upgrade Docker EE.

1. Go to the Docker EE repository URL associated with your trial or subscription in your browser. Go to `sles/12.3/` choose the directory corresponding to your architecture and desired Docker EE version. Download the `.rpm` file from the `Packages` directory.
2. Import Docker's official GPG key:

   ```
   $ sudo rpm --import <DOCKER-EE-URL>/sles/gpg
   ```

3. Install Docker EE, changing the path below to the path where you downloaded the Docker package.

   ```
   $ sudo zypper install /path/to/package.rpm
   ```

   Docker EE is installed but not started. The `docker` group is created, but no users are added to the group.
4. Configure Docker EE to use the Btrfs filesystem. This is only required if the `/` filesystem is not using Btrfs. However, explicitly specifying the `storage-driver` has no harmful side effects.

   Edit the file `/etc/docker/daemon.json` (create it if it does not exist) and add the following contents:

   ```
   {
     "storage-driver": "btrfs"
   }
   ```

   Save and close the file.
5. Start Docker:

   ```
   $ sudo service docker start
   ```

6. Verify that Docker EE is installed correctly by running the `hello-world` image.

```
$ sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

Docker EE is installed and running. You need to use `sudo` to run Docker commands. Continue to Post-installation steps for Linux (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands and for other optional configuration steps.

> Important: Be sure Docker is configured to start after the system firewall. See Firewall configuration (/install/linux/docker-ee/suse/#firewall-configuration).

UPGRADE DOCKER EE

To upgrade Docker EE, download the newer package file and repeat the installation procedure (/install/linux/docker-ee/suse/#install-from-a-package), using `zypper update` instead of `zypper install`, and pointing to the new file.

Uninstall Docker EE

1. Uninstall the Docker EE package using the following command.

   ```
   $ sudo zypper rm docker-ee
   ```

2. Images, containers, volumes, or customized configuration files on your host are not automatically removed. To delete all images, containers, and volumes:

   ```
   $ sudo rm -rf /var/lib/docker/*
   ```

   If you used a separate BTRFS filesystem to host the contents of `/var/lib/docker/`, you can unmount and format the Btrfs filesystem.

You must delete any edited configuration files manually.

Next steps

- Continue to Post-installation steps for Linux (https://docs.docker.com/install/linux/linux-postinstall/)
- Continue with the User Guide (https://docs.docker.com/engine/userguide/).

requirements (https://docs.docker.com/glossary/?term=requirements), apt (https://docs.docker.com/glossary/?term=apt), installation (https://docs.docker.com/glossary/?term=installation), suse (https://docs.docker.com/glossary/?term=suse), opensuse (https://docs.docker.com/glossary/?term=opensuse), sles (https://docs.docker.com/glossary/?term=sles), rpm (https://docs.docker.com/glossary/?term=rpm), install (https://docs.docker.com/glossary/?term=install), uninstall (https://docs.docker.com/glossary/?term=uninstall), upgrade (https://docs.docker.com/glossary/?term=upgrade), update (https://docs.docker.com/glossary/?term=update)

Get Docker EE for Ubuntu

Estimated reading time: 9 minutes

To get started with Docker EE on Ubuntu, make sure you meet the prerequisites (/install/linux/docker-ee/ubuntu/#prerequisites), then install Docker (/install/linux/docker-ee/ubuntu/#install-docker-ee).

Prerequisites

Docker CE users should go to Get Docker CE for Ubuntu (https://docs.docker.com/install/linux/docker-ce/ubuntu/) instead of this topic.

To install Docker Enterprise Edition (Docker EE), you need to know the Docker EE repository URL associated with your trial or subscription. These instructions work for Docker EE for Ubuntu and for Docker EE for Linux, which includes access to Docker EE for all Linux distributions. To get this information:

- Go to https://hub.docker.com/my-content (https://hub.docker.com/my-content).
- Each subscription or trial you have access to is listed. Click the Setup button for Docker Enterprise Edition for Ubuntu.
- Copy the URL from the field labeled Copy and paste this URL to download your Edition.

Use this URL when you see the placeholder text `<DOCKER-EE-URL>` .

To learn more about Docker EE, see Docker Enterprise Edition (https://www.docker.com/enterprise-edition/).

System requirements

To learn more about software requirements and supported storage drivers, check the compatibility matrix (https://success.docker.com/article/compatibility-matrix).

Uninstall old versions

Older versions of Docker were called `docker` or `docker-engine` . In addition, if you are upgrading from Docker CE to Docker EE, remove the Docker CE package.

```
$ sudo apt-get remove docker docker-engine docker-ce docker.io
```

It’s OK if `apt-get` reports that none of these packages are installed.

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

EXTRA STEPS FOR AUFS

If your version supports the `aufs` storage driver, you need some preparation before installing Docker.

Xenial 16.04 or higher	Trusty 14.04

For Ubuntu 16.04 and higher, the Linux kernel includes support for overlay2, and Docker EE uses it as the default storage driver. If you need to use `aufs` instead, you need to configure it manually. See aufs (https://docs.docker.com/engine/userguide/storagedriver/aufs-driver/)

Install Docker EE

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

- Most users set up Docker’s repositories (/install/linux/docker-ee/ubuntu/#install-using-the-repository) and install from them, for ease of installation and upgrade tasks. This is the recommended approach.
- Some users download the DEB package and install it manually and manage upgrades completely manually. This is useful in situations such as installing Docker on air-gapped systems with no access to the internet.

Install using the repository

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

SET UP THE REPOSITORY

1. Update the `apt` package index:

   ```
   $ sudo apt-get update
   ```

2. Install packages to allow `apt` to use a repository over HTTPS:

   ```
   $ sudo apt-get install \
       apt-transport-https \
       ca-certificates \
       curl \
       software-properties-common
   ```

3. Temporarily add a `$DOCKER_EE_URL` variable into your environment. This only persists until you log out of the session. Replace `<DOCKER-EE-URL>` with the URL you noted down in the prerequisites (/install/linux/docker-ee/ubuntu/#prerequisites).

   ```
   $ DOCKER_EE_URL="<DOCKER-EE-URL>"
   ```

4. Temporarily add a `$DOCKER_EE_VERSION` variable into your environment.

   ✓ *NOTE:* If you need to run Docker EE 2.0, please see the following instructions:

   - 18.03 (https://docs.docker.com/v18.03/ee/supported-platforms/) - Older Docker EE Engine only release
   - 17.06 (https://docs.docker.com/v17.06/engine/installation/) - Docker Enterprise Edition 2.0 (Docker Engine, UCP, and DTR).

   ```
   $ DOCKER_EE_VERSION=<YOUR_VERSION>
   ```

5. Add Docker's official GPG key using your customer Docker EE repository URL:

   ```
   $ curl -fsSL "${DOCKER_EE_URL}/ubuntu/gpg" | sudo apt-key add -
   ```

Verify that you now have the key with the fingerprint `DD91 1E99 5A64 A202 E859 07D6 BC14 F10B 6D08 5F96`, by searching for the last eight characters of the fingerprint. Use the command as-is. It works because of the variable you set earlier.

```
$ sudo apt-key fingerprint 6D085F96

pub   4096R/0EBFCD88 2017-02-22
      Key fingerprint = DD91 1E99 5A64 A202 E859  07D6 BC14 F10
B 6D08 5F96
uid                  Docker Release (EE deb) <docker@docker.com
>
sub   4096R/6D085F96 2017-02-22
```

6. Use the following command to set up the stable repository. Use the command as-is. It works because of the variable you set earlier.

> Note: The `lsb_release -cs` sub-command below returns the name of your Ubuntu distribution, such as `xenial`.

x86_64 / amd64	IBM Z (s390x)	IBM Power (ppc64el)

```
$ sudo add-apt-repository \
   "deb [arch=amd64] $DOCKER_EE_URL/ubuntu \
   $(lsb_release -cs) \
   stable-18.09"
```

INSTALL DOCKER EE

1. Update the `apt` package index.

```
$ sudo apt-get update
```

2. Install the latest version of Docker EE, or go to the next step to install a specific version. Any existing installation of Docker EE is replaced.

   Use this command to install the latest version of Docker EE:

```
$ sudo apt-get install docker-ee
```

> Warning: If you have multiple Docker repositories enabled, installing or updating without specifying a version in the `apt-get install` or `apt-get update` command always installs the highest possible version, which may not be appropriate for your stability needs.

3. On production systems, you should install a specific version of Docker EE instead of always using the latest. This output is truncated. List the available versions.

   ```
   $ apt-cache madison docker-ee

   docker-ee | 18.09.0~ee-0~ubuntu-xenial | <DOCKER-EE-URL>/ubuntu
    xenial/stable amd64 Packages
   ```

   The contents of the list depend upon which repositories are enabled, and are specific to your version of Ubuntu (indicated by the `xenial` suffix on the version, in this example). Choose a specific version to install. The second column is the version string. The third column is the repository name, which indicates which repository the package is from and by extension its stability level. To install a specific version, append the version string to the package name and separate them by an equals sign (`=`):

   ```
   $ sudo apt-get install docker-ee=<VERSION>
   ```

   The Docker daemon starts automatically.

4. Verify that Docker EE is installed correctly by running the `hello-world` image.

   ```
   $ sudo docker run hello-world
   ```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

Docker EE is installed and running. The `docker` group is created but no users are added to it. You need to use `sudo` to run Docker commands. Continue to Linux postinstall (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands and for other optional configuration steps.

UPGRADE DOCKER EE

To upgrade Docker EE:

1. If upgrading to a new major Docker EE version (such as when going from Docker 18.03.x to Docker 18.09.x), add the new repository (/install/linux/docker-ee/ubuntu/#set-up-the-repository).
2. Run `sudo apt-get update` .
3. Follow the installation instructions (/install/linux/docker-ee/ubuntu/#install-docker-ee), choosing the new version you want to install.

Install from a package

If you cannot use Docker's repository to install Docker EE, you can download the `.deb` file for your release and install it manually. You need to download a new file each time you want to upgrade Docker EE.

1. Go to the Docker EE repository URL associated with your trial or subscription in your browser. Go to `ubuntu/x86_64/stable-<VERSION>` and download the `.deb` file for the Docker EE version and architecture you want to install.
2. Install Docker EE, changing the path below to the path where you downloaded the Docker EE package.

   ```
   $ sudo dpkg -i /path/to/package.deb
   ```

   The Docker daemon starts automatically.
3. Verify that Docker EE is installed correctly by running the `hello-world` image.

```
$ sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.

Docker EE is installed and running. The `docker` group is created but no users are added to it. You need to use `sudo` to run Docker commands. Continue to Post-installation steps for Linux (https://docs.docker.com/install/linux/linux-postinstall/) to allow non-privileged users to run Docker commands and for other optional configuration steps.

UPGRADE DOCKER EE

To upgrade Docker EE, download the newer package file and repeat the installation procedure (/install/linux/docker-ee/ubuntu/#install-from-a-package), pointing to the new file.

Uninstall Docker EE

1. Uninstall the Docker EE package:

   ```
   $ sudo apt-get purge docker-ee
   ```

2. Images, containers, volumes, or customized configuration files on your host are not automatically removed. To delete all images, containers, and volumes:

   ```
   $ sudo rm -rf /var/lib/docker
   ```

You must delete any edited configuration files manually.

Next steps

- Continue to Post-installation steps for Linux (https://docs.docker.com/install/linux/linux-postinstall/).
- Continue with the User Guide (https://docs.docker.com/engine/userguide/).

requirements (https://docs.docker.com/glossary/?term=requirements), apt (https://docs.docker.com/glossary/?term=apt), installation (https://docs.docker.com/glossary/?term=installation), ubuntu (https://docs.docker.com/glossary/?term=ubuntu), install (https://docs.docker.com/glossary/?term=install), uninstall (https://docs.docker.com/glossary/?term=uninstall), upgrade (https://docs.docker.com/glossary/?term=upgrade), update (https://docs.docker.com/glossary/?term=update)

Install Docker Engine - Enterprise on Windows Servers

Estimated reading time: 6 minutes

Docker Engine - Enterprise enables native Docker containers on Windows Server. Windows Server 2016 and later versions are supported. The Docker Engine - Enterprise installation package includes everything you need to run Docker on Windows Server. This topic describes pre-install considerations, and how to download and install Docker Engine - Enterprise.

Release notes

Release notes for all versions (https://docs.docker.com/release-notes/)

System requirements

Windows OS requirements around specific CPU and RAM requirements also need to be met as specified in the Windows Server Requirements (https://docs.microsoft.com/en-us/windows-server/get-started/system-requirements). This provides information for specific CPU and memory specs and capabilities (instruction sets like CMPXCHG16b, LAHF/SAHF, and PrefetchW, security: DEP/NX, etc.).

- OS Versions: Server 2016 (Core and GUI), 1709 and 1803
- RAM: 4GB
- Disk space: 32 GB minimum recommendation for Windows (https://docs.microsoft.com/en-us/windows-server/get-started/system requirements). An additional 32 GB of Space is recommended for base images for ServerCore and NanoServer along with buffer space for workload containers running IIS, SQL Server and .Net apps.

Install Docker Engine - Enterprise

Docker Engine - Enterprise requires Windows Server 2016, 1703, or 1803. See What to know before you install (/install/windows/docker-ee/#what-to-know-before-you-install) for a full list of prerequisites.

1. Open a PowerShell command prompt, and type the following commands.

   ```
   Install-Module DockerMsftProvider -Force
   Install-Package Docker -ProviderName DockerMsftProvider -Force
   ```

2. Check if a reboot is required, and if yes, restart your instance:

   ```
   (Install-WindowsFeature Containers).RestartNeeded
   ```

   If the output of this command is Yes, then restart the server with:

   ```
   Restart-Computer
   ```

3. Test your Docker Engine - Enterprise installation by running the `hello-world` container.

   ```
   docker run hello-world:nanoserver

   Unable to find image 'hello-world:nanoserver' locally
   nanoserver: Pulling from library/hello-world
   bce2fbc256ea: Pull complete
   3ac17e2e6106: Pull complete
   8cac44e17f16: Pull complete
   5e160e4d8db3: Pull complete
   Digest: sha256:25eac12ba40f7591969085ab3fb9772e8a4307553c14ea72
   d0e6f98b2c8ced9d
   Status: Downloaded newer image for hello-world:nanoserver

   Hello from Docker!
   This message shows that your installation appears to be working
    correctly.
   ```

(optional) Make sure you have all required updates

Some advanced Docker features, such as swarm mode, require the fixes included in KB4015217 (https://support.microsoft.com/en-us/help/4015217/windows-10-update-kb4015217) (or a later cumulative patch).

```
sconfig
```

Select option `6) Download and Install Updates` .

FIPS 140-2 cryptographic module support

Federal Information Processing Standards (FIPS) Publication 140-2 (https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf) is a United States Federal security requirement for cryptographic modules.

With Docker EE Basic license for versions 18.09 and later, Docker provides FIPS 140-2 support in Windows Server 2016. This includes a FIPS supported cryptographic module. If the Windows implementation already has FIPS support enabled, FIPS is automatically enabled in the Docker engine.

NOTE: FIPS 140-2 is only supported in the Docker EE engine. UCP and DTR currently do not have support for FIPS 140-2.

To enable FIPS 140-2 compliance on a system that is not in FIPS 140-2 mode, do the following in PowerShell:

```
[System.Environment]::SetEnvironmentVariable("DOCKER_FIPS", "1", "Ma
chine")
```

Restart the Docker service by running the following command.

```
net stop docker
net start docker
```

To confirm Docker is running with FIPS-140-2 enabled, run the `docker info` command:

```
Labels:
 com.docker.security.fips=enabled
```

NOTE: If the system has the FIPS-140-2 cryptographic module installed on the operating system, it is possible to disable FIPS-140-2 compliance. To disable FIPS-140-2 in Docker but not the operating system, set the value `"DOCKER_FIPS","0"` in the `[System.Environment]` .`

Use a script to install Docker EE

Use the following steps when you want to install manually, script automated installs, or install on air-gapped systems.

1. In a PowerShell command prompt, download the installer archive on a machine that has a connection.

   ```
   # On an online machine, download the zip file.
   Invoke-WebRequest -UseBasicParsing -OutFile docker-18.09.0.zip
   https://download.docker.com/components/engine/windows-server/18
   .09/docker-18.09.0.zip
   ```

   If you need to download a specific Docker EE Engine release, all URLs can be found on this JSON index (https://download.docker.com/components/engine/windows-server/index.json)

2. Copy the zip file to the machine where you want to install Docker. In a PowerShell command prompt, use the following commands to extract the archive, register, and start the Docker service.

```
#Stop Docker service
Stop-Service docker

# Extract the archive.
Expand-Archive docker-18.09.0.zip -DestinationPath $Env:Program
Files -Force

# Clean up the zip file.
Remove-Item -Force docker-18.09.0.zip

# Install Docker. This requires rebooting.
$null = Install-WindowsFeature containers

# Add Docker to the path for the current session.
$env:path += ";$env:ProgramFiles\docker"

# Optionally, modify PATH to persist across sessions.
$newPath = "$env:ProgramFiles\docker;" +
[Environment]::GetEnvironmentVariable("PATH",
[EnvironmentVariableTarget]::Machine)

[Environment]::SetEnvironmentVariable("PATH", $newPath,
[EnvironmentVariableTarget]::Machine)

# Register the Docker daemon as a service.
dockerd --register-service

# Start the Docker service.
Start-Service docker
```

3. Test your Docker EE installation by running the `hello-world` container.

```
docker container run hello-world:nanoserver
```

Install a specific version

To install a specific version, use the `RequiredVersion` flag:

```
Install-Package -Name docker -ProviderName DockerMsftProvider -Force
 -RequiredVersion 18.09
...
Name                      Version                 Source          Sum
mary
----                      -------                 ------          ---
----
Docker                    18.09                   Docker          Con
tains Docker Engine - Enterprise for use with Windows Server...
```

Updating the DockerMsftProvider

Installing specific Docker EE versions may require an update to previously installed DockerMsftProvider modules. To update:

```
Update-Module DockerMsftProvider
```

Then open a new Powershell session for the update to take effect.

Update Docker Engine - Enterprise

To update Docker Engine - Enterprise to the most recent release, specify the `-RequiredVersion` and `-Update` flags:

```
Install-Package -Name docker -ProviderName DockerMsftProvider -Requi
redVersion 18.09 -Update -Force
```

The required version must match any of the versions available in this json file: https://dockermsft.blob.core.windows.net/dockercontainer/DockerMsftIndex.json

Uninstall Docker EE

Use the following commands to completely remove the Docker Engine - Enterprise from a Windows Server:

1. Leave any active Docker Swarm

   ```
   docker swarm leave --force
   ```

2. Remove all running and stopped containers

   ```
   docker rm -f $(docker ps --all --quiet)
   ```

3. Prune container data

   ```
   docker system prune --all --volumes
   ```

4. Uninstall Docker PowerShell Package and Module

   ```
   Uninstall-Package -Name docker -ProviderName DockerMsftProvide
   r
   Uninstall-Module -Name DockerMsftProvider
   ```

5. Clean up Windows Networking and file system

   ```
   Get-HNSNetwork | Remove-HNSNetwork
   Remove-Item -Path "C:\ProgramData\Docker" -Recurse -Force
   ```

Preparing a Docker EE Engine for use with UCP

Run the UCP installation script for Windows (https://docs.docker.com/datacenter/ucp/3.0/guides/admin/configure/join-windows-worker-nodes/#run-the-windows-node-setup-script).

Start the Docker service:

```
Start-Service Docker
```

- What the Docker Engine - Enterprise install includes: The installation provides Docker Engine (https://docs.docker.com/engine/userguide/intro/) and the Docker CLI client (https://docs.docker.com/engine/reference/commandline/cli/).

About Docker Engine - Enterprise containers and Windows Server

Looking for information on using Docker Engine - Enterprise containers?

- Getting Started with Windows Containers (Lab) (https://github.com/docker/labs/blob/master/windows/windows-containers/README.md) provides a tutorial on how to set up and run Windows containers on Windows 10 or Windows Server 2016. It shows you how to use a MusicStore application with Windows containers.
- Setup - Windows Server 2016 (Lab) (https://github.com/docker/labs/blob/master/windows/windows-containers/Setup-Server2016.md) describes environment setup in detail.
- Docker Container Platform for Windows Server articles and blog posts (https://www.docker.com/microsoft/) on the Docker website.

Where to go next

- Getting started (https://docs.docker.com/docker-for-windows/) provides an overview of Docker for Windows, basic Docker command examples, how to get help or give feedback, and links to all topics in the Docker for Windows guide.
- FAQs (https://docs.docker.com/docker-for-windows/faqs/) provides answers to frequently asked questions.
- Release Notes (https://docs.docker.com/docker-for-windows/release-notes/) lists component updates, new features, and improvements associated with Stable and Edge releases.
- Learn Docker (https://docs.docker.com/learn/) provides general Docker tutorials.
- Windows Containers on Windows Server (https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/quick-start-windows-server) is the official Microsoft documentation.

Windows (https://docs.docker.com/glossary/?term=Windows), Windows Server (https://docs.docker.com/glossary/?term=Windows Server), install (https://docs.docker.com/glossary/?term=install), download (https://docs.docker.com/glossary/?term=download), ucp (https://docs.docker.com/glossary/?term=ucp), Docker Engine - Enterprise (https://docs.docker.com/glossary/?term=Docker Engine - Enterprise)