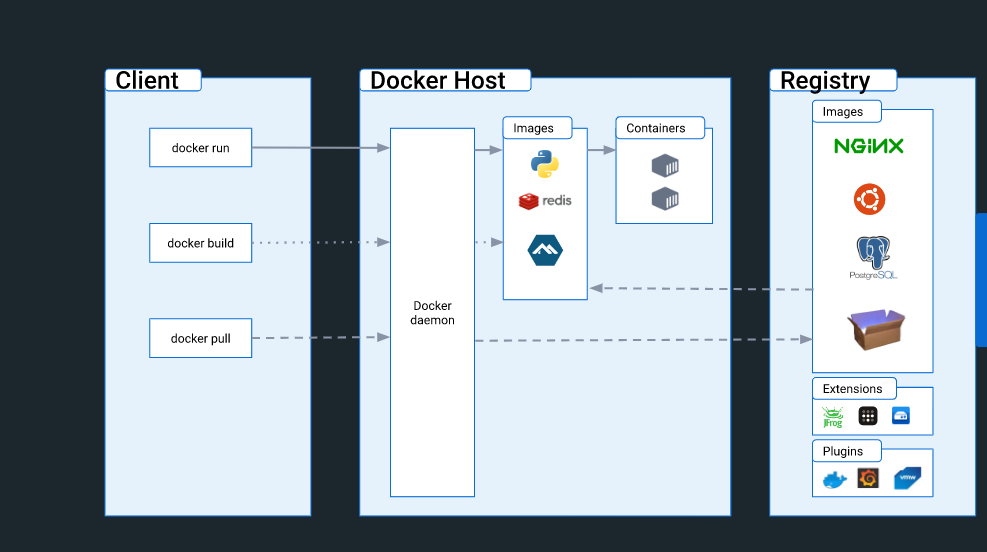
Docker architecture

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. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



### **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 Desktop**

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see [Docker Desktop](https://docs.docker.com/desktop/).

### **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.

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

Docker installation on linux distributions

Eg-linux 22.04

First, update your existing list of packages:

1. sudo apt update

Copy

Next, install a few prerequisite packages which let apt use packages over HTTPS:

1. sudo apt install apt-transport-https ca-certificates curl software-properties-common

Copy

Then add the GPG key for the official Docker repository to your system:

1. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

Copy

Add the Docker repository to APT sources:

1. echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

Copy

Update your existing list of packages again for the addition to be recognized:

1. sudo apt update

Copy

Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:

1. apt-cache policy docker-ce

Copy

You’ll see output like this, although the version number for Docker may be different:

Output of apt-cache policy docker-ce

docker-ce:

Installed: (none)

Candidate: 5:20.10.14~3-0~ubuntu-jammy

Version table:

5:20.10.14~3-0~ubuntu-jammy 500

500 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages

5:20.10.13~3-0~ubuntu-jammy 500

500 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages

Notice that docker-ce is not installed, but the candidate for installation is from the Docker repository for Ubuntu 22.04 (jammy).

Finally, install Docker:

1. sudo apt install docker-ce

Copy

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it’s running:

1. sudo systemctl status docker

Copy

The output should be similar to the following, showing that the service is active and running:

Output

● docker.service - Docker Application Container Engine

Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)

Active: active (running) since Fri 2022-04-01 21:30:25 UTC; 22s ago

TriggeredBy: ● docker.socket

Docs: https://docs.docker.com

Main PID: 7854 (dockerd)

Tasks: 7

Memory: 38.3M

CPU: 340ms

CGroup: /system.slice/docker.service

└─7854 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Installing Docker now gives you not just the Docker service (daemon) but also the docker command line utility, or the Docker client. We’ll explore how to use the docker command later in this tutorial.

[**Step 2 — Executing the Docker Command Without Sudo (Optional)**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04#step-2-executing-the-docker-command-without-sudo-optional)

By default, the docker command can only be run the **root** user or by a user in the **docker** group, which is automatically created during Docker’s installation process. If you attempt to run the docker command without prefixing it with sudo or without being in the **docker** group, you’ll get an output like this:

Output

docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.

See 'docker run --help'.

If you want to avoid typing sudo whenever you run the docker command, add your username to the docker group:

1. sudo usermod -aG docker ${USER}

Copy

To apply the new group membership, log out of the server and back in, or type the following:

1. su - ${USER}

Copy

You will be prompted to enter your user’s password to continue.

Confirm that your user is now added to the **docker** group by typing:

1. groups

Copy

Output

sammy sudo docker

If you need to add a user to the docker group that you’re not logged in as, declare that username explicitly using:

1. sudo usermod -aG docker username

Copy

The rest of this article assumes you are running the docker command as a user in the **docker** group. If you choose not to, please prepend the commands with sudo.

Let’s explore the docker command next.

[**Step 3 — Using the Docker Command**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04#step-3-using-the-docker-command)

Using docker consists of passing it a chain of options and commands followed by arguments. The syntax takes this form:

1. docker [option] [command] [arguments]

Copy

To view all available subcommands, type:

1. docker

Copy

As of Docker version 20.10.14, the complete list of available subcommands includes:

Output

attach Attach local standard input, output, and error streams to a running container

build Build an image from a Dockerfile

commit Create a new image from a container's changes

cp Copy files/folders between a container and the local filesystem

create Create a new container

diff Inspect changes to files or directories on a container's filesystem

events Get real time events from the server

exec Run a command in a running container

export Export a container's filesystem as a tar archive

history Show the history of an image

images List images

import Import the contents from a tarball to create a filesystem image

info Display system-wide information

inspect Return low-level information on Docker objects

kill Kill one or more running containers

load Load an image from a tar archive or STDIN

login Log in to a Docker registry

logout Log out from a Docker registry

logs Fetch the logs of a container

pause Pause all processes within one or more containers

port List port mappings or a specific mapping for the container

ps List containers

pull Pull an image or a repository from a registry

push Push an image or a repository to a registry

rename Rename a container

restart Restart one or more containers

rm Remove one or more containers

rmi Remove one or more images

run Run a command in a new container

save Save one or more images to a tar archive (streamed to STDOUT by default)

search Search the Docker Hub for images

start Start one or more stopped containers

stats Display a live stream of container(s) resource usage statistics

stop Stop one or more running containers

tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

top Display the running processes of a container

unpause Unpause all processes within one or more containers

update Update configuration of one or more containers

version Show the Docker version information

wait Block until one or more containers stop, then print their exit codes

To view the options available to a specific command, type:

1. docker docker-subcommand --help

Copy

To view system-wide information about Docker, use:

1. docker info

Copy

Let’s explore some of these commands. We’ll start by working with images.

[**Step 4 — Working with Docker Images**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04#step-4-working-with-docker-images)

Docker containers are built from Docker images. By default, Docker pulls these images from [Docker Hub](https://hub.docker.com/), a Docker registry managed by Docker, the company behind the Docker project. Anyone can host their Docker images on Docker Hub, so most applications and Linux distributions you’ll need will have images hosted there.

To check whether you can access and download images from Docker Hub, type:

1. docker run hello-world

Copy

The output will indicate that Docker in working correctly:

Output

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

latest: Pulling from library/hello-world

2db29710123e: Pull complete

Digest: sha256:bfea6278a0a267fad2634554f4f0c6f31981eea41c553fdf5a83e95a41d40c38

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

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

...

Docker was initially unable to find the hello-world image locally, so it downloaded the image from Docker Hub, which is the default repository. Once the image downloaded, Docker created a container from the image and the application within the container executed, displaying the message.

You can search for images available on Docker Hub by using the docker command with the search subcommand. For example, to search for the Ubuntu image, type:

1. docker search ubuntu

Copy

The script will crawl Docker Hub and return a listing of all images whose name matches the search string. In this case, the output will be similar to this:

Output

NAME DESCRIPTION STARS OFFICIAL AUTOMATED

ubuntu Ubuntu is a Debian-based Linux operating sys… 14048 [OK]

websphere-liberty WebSphere Liberty multi-architecture images … 283 [OK]

ubuntu-upstart DEPRECATED, as is Upstart (find other proces… 112 [OK]

neurodebian NeuroDebian provides neuroscience research s… 88 [OK]

open-liberty Open Liberty multi-architecture images based… 51 [OK]

...

In the **OFFICIAL** column, **OK** indicates an image built and supported by the company behind the project. Once you’ve identified the image that you would like to use, you can download it to your computer using the pull subcommand.

Execute the following command to download the official ubuntu image to your computer:

1. docker pull ubuntu

Copy

You’ll see the following output:

Output

Using default tag: latest

latest: Pulling from library/ubuntu

e0b25ef51634: Pull complete

Digest: sha256:9101220a875cee98b016668342c489ff0674f247f6ca20dfc91b91c0f28581ae

Status: Downloaded newer image for ubuntu:latest

docker.io/library/ubuntu:latest

After an image has been downloaded, you can then run a container using the downloaded image with the run subcommand. As you saw with the hello-world example, if an image has not been downloaded when docker is executed with the run subcommand, the Docker client will first download the image, then run a container using it.

To see the images that have been downloaded to your computer, type:

1. docker images

Copy

The output will look similar to the following:

Output

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu latest 1d622ef86b13 3 weeks ago 73.9MB

hello-world latest bf756fb1ae65 4 months ago 13.3kB

As you’ll see later in this tutorial, images that you use to run containers can be modified and used to generate new images, which may then be uploaded (*pushed* is the technical term) to Docker Hub or other Docker registries.

Let’s look at how to run containers in more detail.

[**Step 5 — Running a Docker Container**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04#step-5-running-a-docker-container)

The hello-world container you ran in the previous step is an example of a container that runs and exits after emitting a test message. Containers can be much more useful than that, and they can be interactive. After all, they are similar to virtual machines, only more resource-friendly.

As an example, let’s run a container using the latest image of Ubuntu. The combination of the **-i** and **-t** switches gives you interactive shell access into the container:

1. docker run -it ubuntu

Copy

Your command prompt should change to reflect the fact that you’re now working inside the container and should take this form:

Output

root@d9b100f2f636:/#

Note the container id in the command prompt. In this example, it is d9b100f2f636. You’ll need that container ID later to identify the container when you want to remove it.

Now you can run any command inside the container. For example, let’s update the package database inside the container. You don’t need to prefix any command with sudo, because you’re operating inside the container as the **root** user:

1. apt update

Copy

Then install any application in it. Let’s install Node.js:

1. apt install nodejs

Copy

This installs Node.js in the container from the official Ubuntu repository. When the installation finishes, verify that Node.js is installed:

1. node -v

Copy

You’ll see the version number displayed in your terminal:

Output

v12.22.9

Any changes you make inside the container only apply to that container.

To exit the container, type exit at the prompt.

Let’s look at managing the containers on our system next.

[**Step 6 — Managing Docker Containers**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04#step-6-managing-docker-containers)

After using Docker for a while, you’ll have many active (running) and inactive containers on your computer. To view the **active ones**, use:

1. docker ps

Copy

You will see output similar to the following:

Output

CONTAINER ID IMAGE COMMAND CREATED

In this tutorial, you started two containers; one from the hello-world image and another from the ubuntu image. Both containers are no longer running, but they still exist on your system.

To view all containers — active and inactive, run docker ps with the -a switch:

1. docker ps -a

Copy

You’ll see output similar to this:

Output

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1c08a7a0d0e4 ubuntu "bash" About a minute ago Exited (0) 7 seconds ago dazzling\_taussig

587000e49d53 hello-world "/hello" 5 minutes ago Exited (0) 5 minutes ago adoring\_kowalevski

To view the latest container you created, pass it the -l switch:

1. docker ps -l

Copy

Output

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1c08a7a0d0e4 ubuntu "bash" 3 minutes ago Exited (0) 2 minutes ago dazzling\_taussig

To start a stopped container, use docker start, followed by the container ID or the container’s name. Let’s start the Ubuntu-based container with the ID of 1c08a7a0d0e4:

1. docker start 1c08a7a0d0e4

Copy

The container will start, and you can use docker ps to see its status:

Output

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1c08a7a0d0e4 ubuntu "bash" 6 minutes ago Up 8 seconds dazzling\_taussig

To stop a running container, use docker stop, followed by the container ID or name. This time, we’ll use the name that Docker assigned the container, which is dazzling\_taussig:

1. docker stop dazzling\_taussig

Copy

Once you’ve decided you no longer need a container anymore, remove it with the docker rm command, again using either the container ID or the name. Use the docker ps -a command to find the container ID or name for the container associated with the hello-world image and remove it.

1. docker rm adoring\_kowalevski

Copy

You can start a new container and give it a name using the --name switch. You can also use the --rm switch to create a container that removes itself when it’s stopped. See the docker run help command for more information on these options and others.

Containers can be turned into images which you can use to build new containers. Let’s look at how that works.

[**Step 7 — Committing Changes in a Container to a Docker Image**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04#step-7-committing-changes-in-a-container-to-a-docker-image)

When you start up a Docker image, you can create, modify, and delete files just like you can with a virtual machine. The changes that you make will only apply to that container. You can start and stop it, but once you destroy it with the docker rm command, the changes will be lost for good.

This section shows you how to save the state of a container as a new Docker image.

After installing Node.js inside the Ubuntu container, you now have a container running off an image, but the container is different from the image you used to create it. But you might want to reuse this Node.js container as the basis for new images later.

Then commit the changes to a new Docker image instance using the following command.

1. docker commit -m "What you did to the image" -a "Author Name" container\_id repository/new\_image\_name

Copy

The **-m** switch is for the commit message that helps you and others know what changes you made, while **-a** is used to specify the author. The container\_id is the one you noted earlier in the tutorial when you started the interactive Docker session. Unless you created additional repositories on Docker Hub, the repository is usually your Docker Hub username.

For example, for the user **sammy**, with the container ID of d9b100f2f636, the command would be:

1. docker commit -m "added Node.js" -a "sammy" d9b100f2f636 sammy/ubuntu-nodejs

Copy

When you *commit* an image, the new image is saved locally on your computer. Later in this tutorial, you’ll learn how to push an image to a Docker registry like Docker Hub so others can access it.

Listing the Docker images again will show the new image, as well as the old one that it was derived from:

1. docker images

Copy

You’ll see output like this:

Output

REPOSITORY TAG IMAGE ID CREATED SIZE

sammy/ubuntu-nodejs latest 7c1f35226ca6 7 seconds ago 179MB

...

In this example, ubuntu-nodejs is the new image, which was derived from the existing ubuntu image from Docker Hub. The size difference reflects the changes that were made. And in this example, the change was that NodeJS was installed. So next time you need to run a container using Ubuntu with NodeJS pre-installed, you can just use the new image.

You can also build Images from a Dockerfile, which lets you automate the installation of software in a new image. However, that’s outside the scope of this tutorial.

Now let’s share the new image with others so they can create containers from it.

[**Step 8 — Pushing Docker Images to a Docker Repository**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04#step-8-pushing-docker-images-to-a-docker-repository)

The next logical step after creating a new image from an existing image is to share it with a select few of your friends, the whole world on Docker Hub, or other Docker registry that you have access to. To push an image to Docker Hub or any other Docker registry, you must have an account there.

To push your image, first log into Docker Hub.

1. docker login -u docker-registry-username

Copy

You’ll be prompted to authenticate using your Docker Hub password. If you specified the correct password, authentication should succeed.

**Note:** If your Docker registry username is different from the local username you used to create the image, you will have to tag your image with your registry username. For the example given in the last step, you would type:

1. docker tag sammy/ubuntu-nodejs docker-registry-username/ubuntu-nodejs

Copy

Then you may push your own image using:

1. docker push docker-registry-username/docker-image-name

Copy

To push the ubuntu-nodejs image to the **sammy** repository, the command would be:

1. docker push sammy/ubuntu-nodejs

Copy

The process may take some time to complete as it uploads the images, but when completed, the output will look like this:

Output

The push refers to a repository [docker.io/sammy/ubuntu-nodejs]

e3fbbfb44187: Pushed

5f70bf18a086: Pushed

a3b5c80a4eba: Pushed

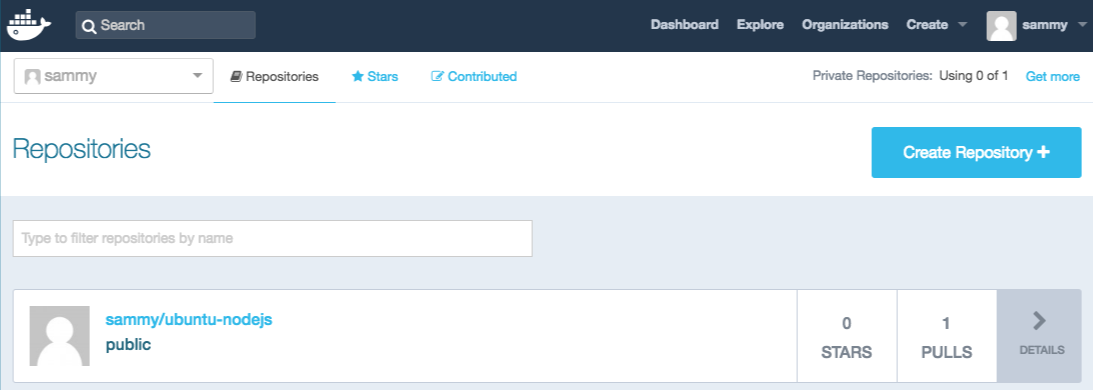
7f18b442972b: Pushed

3ce512daaf78: Pushed

7aae4540b42d: Pushed

...

After pushing an image to a registry, it should be listed on your account’s dashboard, like that show in the image below.



If a push attempt results in an error of this sort, then you likely did not log in:

Output

The push refers to a repository [docker.io/sammy/ubuntu-nodejs]

e3fbbfb44187: Preparing

5f70bf18a086: Preparing

a3b5c80a4eba: Preparing

7f18b442972b: Preparing

3ce512daaf78: Preparing

7aae4540b42d: Waiting

unauthorized: authentication required

Log in with docker login and repeat the push attempt. Then verify that it exists on your Docker Hub repository page.

You can now use docker pull sammy/ubuntu-nodejs to pull the image to a new machine and use it to run a new container.

[**Conclusion**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04#conclusion)

In this tutorial you installed Docker, worked with images and containers, and pushed a modified image to Docker Hub. Now that you know the basics, explore the [other Docker tutorials](https://www.digitalocean.com/community/tags/docker?type=tutorials) in the DigitalOcean Community.

Thanks for learning with the DigitalOcean Community. Check out our offerings for compute, storage, networking, and managed databases.

[**Learn more about us**](https://www.digitalocean.com/)

[**Next in series: How To Install and Use Docker Compose on Ubuntu 22.04 ->**](https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-22-04)