

What are Containers?

The idea of containers as a way of running applications in isolation has been around for more than a decade. In practical terms, containers provide a layer of abstraction, allowing applications to be packaged with all their dependencies so they can run in any environment.

By taking advantage of OS virtualization primitives such as namespaces and control groups (cgroups), containers completely isolate application processes and protect them from interference and leaks.

In addition, a containerized environment offers control over the allocation of hardware resources (CPU, memory, and disk) available to a container.

Containers are faster than virtual machines because they don't require a guest operating system in every instance. Instead, they simply leverage the features and resources of the host OS to provide a working foundation for each container.

If you want to learn more about containers, you can read [Containerization vs Virtualization: Making the Right Choice for Your Infrastructure](#).

Benefits of Containers

Compared to other virtualization technologies, containers offer significant operational benefits, such as:

Lightweight

By sharing the machine OS kernel, containers eliminate the need for full OS instances for the applications. This reduces the size of container files and makes them more resource-efficient.

Portability and Platform Independence

Since all files, dependencies, and assets required by the application are included in the container, you can set up and initiate a container anywhere you need. This also means that you only need to write an app once and then run it on any compatible environment, including local devices, cloud, and on-premises environments, without re-configuring it.

What is Docker?

Docker is a great platform for packaging an application or a set of services in containers. A Docker container contains all the libraries and components required to run the application. From a practical point of view, a container is like a simplified virtual machine that functions independently of the underlying operating system.

In Docker, containers are generated from images that contain a complete operating system and/or pre-installed applications.

You can find a Docker image for all popular applications, including OS, DBMS, CRM, and programming languages. We strongly recommend checking out the official [Docker image repository](#) for an image. In case you can't find it, you can create your own image and deploy it to your Docker platform.

Let's see how you can install Docker on Ubuntu 22.04 in a couple of easy steps.

Installing Docker on Ubuntu

Ubuntu is the number one platform for managing Docker or [Kubernetes](#) containers. This is because Ubuntu runs the containers at scale, it is fast, secure, and open-source, powering millions of machines worldwide.

This article will cover two options for installing Docker on Ubuntu:

- **From the official Docker repository** - ensuring the latest Docker version.
- **From the default Ubuntu repositories** - providing a simpler installation.

Follow the steps below and install Docker on Ubuntu 20.04 or Ubuntu 22.04 using the method of your choice.

Important: Make sure to remove any older Docker installations before installing a new one. Removing previous Docker versions doesn't delete the images, containers, volumes, or networks you have created. Run the following command to uninstall previous versions:

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

Installing Docker from the Official Repository (Option 1)

Install Docker from the official Docker repository to ensure you get the latest stable program version. To access the official Docker repository, add the new package source to Ubuntu and then install Docker. Follow the steps below:

Step 1: Update the Package Repository

Run the following command to update the system's package repository and ensure the latest prerequisite packages are installed:

```
sudo apt update
```

When prompted, enter your root password and press **Enter** to proceed with the update.

Step 2: Install Prerequisite Packages

The apt package manager requires a few prerequisite packages on the system to use packages over HTTPS. Run the following command to allow Ubuntu to access the Docker repositories over HTTPS:

```
sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
```

```
bosko@pnap:~$ sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
ca-certificates is already the newest version (20211016~20.04.1).
ca-certificates set to manually installed.
software-properties-common is already the newest version (0.99.9.8).
software-properties-common set to manually installed.
The following additional packages will be installed:
  libcurl4
The following NEW packages will be installed:
  apt-transport-https curl libcurl4
0 upgraded, 3 newly installed, 0 to remove and 5 not upgraded.
Need to get 163 kB/398 kB of archives.
After this operation, 1,283 kB of additional disk space will be used.
```

The command above:

- Allows **apt** to transfer files and data over https.
- Allows the system to check security certificates.
- Installs curl, a data-transfer utility.
- Adds scripts for software management.

Step 3: Add GPG Key

A GPG key verifies the authenticity of a software package. Add the Docker repository GPG key to your system by running:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
bosko@pnap:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
```

The output should state **OK**, verifying the authenticity.

Step 4: Add Docker Repository

Run the following command to add the Docker repository to **apt** sources:

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

```
bosko@pnep:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
Get:1 https://download.docker.com/linux/ubuntu focal InRelease [57.7 kB]
Hit:2 http://us.archive.ubuntu.com/ubuntu focal InRelease
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:4 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages [18.5 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main amd64 DEP-11 Metadata [40.7 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/universe amd64 DEP-11 Metadata [77.6 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 DEP-11 Metadata [278 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 DEP-11 Metadata [2,464 B]
```

The command adds the official Docker repository and updates the package database with the latest Docker packages.

Step 5: Specify Installation Source

Execute the **apt-cache** command to ensure the Docker installation source is the Docker repository, not the Ubuntu repository. The **apt-cache** command queries the package cache of the **apt** package manager for the Docker packages we have previously added.

Run the following command:

```
apt-cache policy docker-ce
```

```

bosko@nnap:~$ apt-cache policy docker-ce
docker-ce:
  Installed: (none)
  Candidate: 5:20.10.18~3-0~ubuntu-focal
  Version table:
   5:20.10.18~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
   5:20.10.17~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
   5:20.10.16~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
   5:20.10.15~3-0~ubuntu-focal 500
        500 https://download.docker.com/linux/ubuntu focal/stable amd64 Packages
   5:20.10.14~3-0~ubuntu-focal 500

```

The output states which version is the latest in the added source repository.

Step 6: Install Docker

Install Docker by running:

```
sudo apt install docker-ce -y
```

```

bosko@nnap:~$ sudo apt install docker-ce -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  bridge-utils ubuntu-fan
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  containerd.io docker-ce-cli docker-ce-rootless-extras docker-scan-plugin
  slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following packages will be REMOVED:
  containerd docker.io runc
The following NEW packages will be installed:
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras
  docker-scan-plugin slirp4netns
0 upgraded, 6 newly installed, 3 to remove and 5 not upgraded.
Need to get 102 MB of archives.
After this operation, 64.1 MB of additional disk space will be used.

```

Wait for the installation process to complete.

Step 7: Check Docker Status

Check if Docker is installed, the daemon started, and the process is enabled to start on boot. Run the following command:

```
sudo systemctl status docker
```

```
bosko@pnap:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
   Active: active (running) since Thu 2022-10-06 18:17:02 EDT; 3min 52s ago
   TriggeredBy: ● docker.socket
     Docs: https://docs.docker.com
    Main PID: 2735 (dockerd)
       Tasks: 9
      Memory: 28.2M
     CGroup: /system.slice/docker.service
             └─2735 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/cont>
```

The output states that the Docker daemon is up and running.

Installing Docker from the Default Repositories (Option 2)

Another way to install Docker on Ubuntu is to use the default Ubuntu repository. Although the installation process is more straightforward, the Docker package may be outdated. If you don't care about having the latest Docker version, follow the steps below and install Docker using the default repository.

Step 1: Update the Repository

Ensure that the local system package repository is updated by running:

```
sudo apt update
```

Enter the root password when prompted and wait for the process to finish.

Step 2: Install Docker

Run the following command to install Docker:

```
sudo apt install docker.io -y
```

```
bosko@pnap:~$ sudo apt install docker.io -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd git git-man liberror-perl pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools btrfs-progs cgroupfs-mount | cgroup-lite debootstrap
  docker-doc rinse zfs-fuse | zfsutils git-daemon-run | git-daemon-sysvinit
  git-doc git-el git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  bridge-utils containerd docker.io git git-man liberror-perl pigz runc
  ubuntu-fan
0 upgraded, 9 newly installed, 0 to remove and 5 not upgraded.
Need to get 74.3 MB of archives.
After this operation, 372 MB of additional disk space will be used.
```

Specifying the **-y** flag automatically answers **yes** to any prompt during the installation.

Step 3: Install Dependencies

Install all the Docker dependency packages by running the following command:

```
sudo snap install docker
```

```
bosko@pnap:~$ sudo snap install docker
docker 20.10.14 from Canonical ✓ installed
bosko@pnap:~$
```

The command installs all the dependencies using the Snap package manager.

Step 4: Check Installation

Check whether Docker was properly installed by running the **status** command or checking the program version. To see the Docker daemon status, run:

```
sudo systemctl status docker
```

Alternatively, check the program version by running:

```
docker --version
```

```
bosko@pnap:~$ docker --version
Docker version 20.10.12, build 20.10.12-0ubuntu2~20.04.1
```

How to Use Docker on Ubuntu

All Docker information, including the syntax, options, and commands, is available by running the **docker** command in the terminal:

docker

```
bosko@pnap:~$ docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
  --config string      Location of client config files (default
                        "/home/bosko/.docker")
  -c, --context string Name of the context to use to connect to the
                        daemon (overrides DOCKER_HOST env var and
                        default context set with "docker context use")
  -D, --debug          Enable debug mode
  -H, --host list      Daemon socket(s) to connect to
  -l, --log-level string Set the logging level
                        ("debug"|"info"|"warn"|"error"|"fatal")
                        (default "info")
```

Start using Docker by downloading Docker images, creating containers, and managing Docker volumes.

Important: Docker commands can only be run with the **sudo** prefix on Ubuntu.

Working With Docker Images

Docker images are the base for building Docker containers. The images are located on Docker Hub, which is a Docker repository. The repository allows all users to host their images on Docker hub, resulting in many images to choose from, including applications and Linux distributions.

The sections below show different ways of working with Docker images.

Note: See how to update a Docker image or container to the latest version available.

Search for Docker Images

Search for available images on Docker Hub using the **docker search** command. The syntax is:


```
sudo docker search [keyword]
```

For **[keyword]**, specify the keyword you want to search for. For example, to show all Ubuntu images, run:

```
sudo docker search ubuntu
```

```
bosko@pnep:~$ sudo docker search ubuntu
NAME                DESCRIPTION
STARS              OFFICIAL    AUTOMATED
ubuntu             Ubuntu is a Debian-based Linux operating sys...
15005              [OK]
websphere-liberty  WebSphere Liberty multi-architecture images ...
289                [OK]
ubuntu-upstart     DEPRECATED, as is Upstart (find other proces...
112                [OK]
neurodebian        NeuroDebian provides neuroscience research s...
93                 [OK]
ubuntu/nginx       Nginx, a high-performance reverse proxy & we...
61
open-liberty       Open Liberty multi-architecture images based...
55                 [OK]
```

The output is a list of all images containing the Ubuntu keyword. If the **OFFICIAL** column contains the **[OK]** parameter, the image was uploaded by the official company that developed the project.

Pull a Docker Image

After deciding which image you want, download it using the **pull** option. The syntax is:

```
sudo docker pull [image-name]
```

For example, download the official Ubuntu image by running:

```
sudo docker pull ubuntu
```

```
bosko@pnep:~$ sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
cf92e523b49e: Pull complete
Digest: sha256:35fb073f9e56eb84041b0745cb714eff0f7b225ea9e024f703cab56aaa5c7720
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

After downloading the image, use it to spin up a container. Alternatively, trying to create a container from an image that hasn't been downloaded causes Docker to download the image first and then create the container.

See Downloaded Images

Check which images you have downloaded by running:

```
sudo docker images
```

```
bosko@pnep:~$ sudo docker images
REPOSITORY    TAG       IMAGE ID      CREATED        SIZE
ubuntu        latest    216c552ea5ba  47 hours ago  77.8MB
mysql         latest    43fcfca0776d  3 weeks ago   449MB
```

The command outputs a list of all downloaded images on your machine. In our case, it is a Ubuntu and MySQL Docker image.

Working With Docker Containers

A Docker container is an isolated virtual environment created from a Docker image. Use an image you previously downloaded or specify its name in the **docker run** command to automatically download the image and create a container.

For example, use the **hello-world** image to download a test image and spin up a container. Run the following command:

```
sudo docker run hello-world
```

```
bosko@pnep:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:62af9efd515a25f84961b70f973a798d2eca956b1b2b026d0a4a63a3b0b6a3f2
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.
```

The command instructs Docker to download the image from Docker Hub and spin up a container. After creation, the *"Hello from Docker"* message appears, along with the explanation of how the container works, and then Docker shuts it down.

Note: Check out the best container practices in [How to Manage Docker Containers](#), or see how to [list, start, and stop Docker containers](#).

Run a Docker Container

Like in the test container created in the previous section, [running a Docker container](#) utilizes the **run** subcommand. The syntax for creating a new Docker container is as follows:

```
sudo docker run [image-name]
```

For **[image-name]**, specify the name of the image to use as a base for the container. When creating a new container, Docker assigns it a unique name. Alternatively, start a new container and give it a name of your choice by passing the **--name** switch:

```
sudo docker run --name [container-name] [image-name]
```

Note: The **docker run** command is an alias for **docker container run**. Before Docker version 1.13, only the **docker run** command was used, but later it was refactored to have the form **docker [COMMAND] [SUBCOMMAND]**, where the **[COMMAND]** is **container**, and the **[SUBCOMMAND]** is **run**.

For example, run the following command to create a container using the Ubuntu image we pulled earlier:

```
sudo docker run ubuntu
```

```
bosko@pnap:~$ sudo docker run ubuntu
bosko@pnap:~$
```

The command creates a container from the specified image, but it is in non-interactive mode. To obtain interactive shell access to the container, specify the **-i** and **-t** switches. For example:

```
sudo docker run -it ubuntu
```

```
bosko@pnap:~$ sudo docker run -it ubuntu
root@c9395c7f03b0:/#
```

The terminal changes, stating that it is operating within the container, identifying it via the container ID. Use the container ID to remove, start, or stop the container.

After running the container in interactive mode, pass any command normally to interact with the system. Also, the access is root, so there is no need for **sudo**. Any changes made inside the container apply only to that container.

Exit the container by running the **exit** command in the prompt.

Note: For additional security, learn how to [SSH into a running Docker container](#).

View Docker Containers

An active Docker container is a container that is currently running. Listing containers is useful as it outputs the unique ID and name required when starting, stopping, or removing a container.

To see only the active Docker containers, run:

```
sudo docker ps
```

```
bosko@pnap:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
bosko@pnap:~$
```

The command outputs a list of active containers. If there are no containers in the list, they have all been shut down, but they still exist in the system.

To list all containers, including the inactive ones, add the **-a** flag:

```
sudo docker ps -a
```

```
bosko@pnap:~$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
c9395c7f03b0	ubuntu	"bash"	About a minute ago	Exited (0) 46 seconds ago
5f9478691970	ubuntu	"bash"	3 minutes ago	Exited (0) 3 minutes ago
64f715dc77e4	hello-world	"/hello"	5 minutes ago	Exited (0) 5 minutes ago

The command outputs all existing containers and their details, including the container ID, image, command, creation time, the status, ports, and the unique name.

Alternatively, show only the latest container by passing the **-l** flag:

```
sudo docker ps -l
```

```
bosko@pnap:~$ sudo docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
c9395c7f03b0	ubuntu	"bash"	2 minutes ago	Exited (0) About a minute ago

Start a Docker Container

Start a stopped container using the **docker start** command. The syntax is:

```
sudo docker start [container-ID | container-name]
```

Provide either the **container ID** or the **container name**. The container name is a unique name that Docker assigns to the container during creation. Obtain the ID and name by listing all containers.

For example, the following command starts the Ubuntu container we previously created:

```
sudo docker start 5f9478691970
```

Stop a Docker Container

Stop a running Docker container using the **docker stop** command. The syntax is:

```
sudo docker stop [container-ID | container-name]
```

For example, to stop the running Ubuntu container, run:

```
sudo docker stop 5f9478691970
```

Remove a Docker Container

Remove an unnecessary Docker container using the **docker rm** command. The syntax is:

```
sudo docker rm [container-ID | container-name]
```

For example, the following command removes the **hello-world** test container we previously created:

```
sudo docker rm silly_hamilton
```

Note: Pass the **--rm** switch when creating a container to remove the container automatically when stopped.

Working With Docker Volumes

A Docker volume is a filesystem mechanism allowing users to preserve data generated and used by Docker containers. The volumes ensure data persistence and provide a better alternative to persisting data in a container's writable layer because they don't increase the Docker container size.

Note: See how to [keep the Docker image size small](#).

Use Docker volumes for databases or other stateful applications. Since the volumes are stored on the host, they don't depend on the container but allow for easy data backups and [sharing between multiple containers](#).

Create a Docker Volume

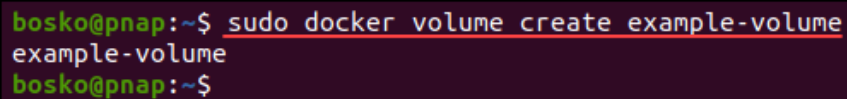
Create a Docker volume using the following syntax:

```
sudo docker volume create [volume_name]
```

For **[volume_name]**, specify a unique name for your volume.

For example:

```
sudo docker volume create example-volume
```

A terminal window with a dark purple background. The prompt is 'bosko@pnap:~\$'. The command 'sudo docker volume create example-volume' is entered and executed. The output is 'example-volume'. The prompt changes to 'bosko@pnap:~\$'.

Remove a Docker Volume

Remove a Docker volume using the following syntax:

```
sudo docker volume rm [volume_name]
```

For example:

```
sudo docker volume rm example-volume
```

Check out our tutorial for detailed information and instructions on [using Docker volumes](#).

Docker Network Commands

Networking is very tightly integrated with Docker and running Docker containers. Docker networking commands allow users to create and manage their networks within a container. The following table shows the basic subcommands used for Docker network management:

Command	Description
docker network connect [network-name]	Connects a container to a network.
docker network create [network-name]	Creates a new network.
docker network disconnect [network-name]	Disconnects a container from the specified network.
docker network inspect [network-name]	Outputs detailed information on the specified network or networks.
docker network ls	Lists all networks in a container.
docker network prune	Removes all unused networks in a container.
docker network rm [network-name]	Removes the specified network or networks.

For example, run the following command to list all existing networks:

```
sudo docker network ls
```

```
bosko@pnap:~$ sudo docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
9d0b512f31b7    bridge    bridge       local
413f51568712    host      host         local
5d4e67e4a38b    none      null         local
```

Since this is a fresh installation, the output shows only a few automatically created networks. The information includes the unique network ID, an arbitrary name for each network, the network driver, and the network scope.

To obtain more detail on a particular network, use the **inspect** subcommand. For example, to get detailed information on the **bridge** network, run:


```
sudo docker network inspect bridge
```

```
bosko@pnap:~$ sudo docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "9d0b512f31b7a8739155bc37653b42b5f954da544e8dc8a37abedebd7cb979c6"
  },
  {
    "Created": "2022-10-07T03:13:34.530770415-04:00",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    }
  }
]
```

The output includes details such as all the containers attached to the network, the particular bridge it is connected to, and much more.

For more detail on Docker commands, refer to our [Docker commands tutorial](#), which includes a downloadable cheat sheet with all the important commands in one place.