



Sale ends in  
0d 10h 50m 20s



EN

TUTORIALS ▾

category ▾

[Home](#) > [Tutorials](#) > [Cloud](#)

# Install Docker on Ubuntu: From Setup to First Container

It only takes a few commands from a fresh OS installation to running your first containerized application. Learn how to do this with this ultimate Docker Ubuntu setup guide.

[Contents](#)

Feb 4, 2025 · 11 min read



Dario Radečić

Senior Data Scientist | Top Tech Writer (50K+ Medium followers, 10M+ views) | Book Author (Machine Learning Automation with TPOT).

## TOPICS

Cloud

Docker

Kubernetes

If you want to stop saying, "*it works on my machine*," containerization is the way to go.

Put simply, containerization allows you to package your applications into lightweight, isolated environments called **containers**. These containers include everything needed to run the application, such as code and dependencies, but nothing unnecessary. A containerized application will run consistently across different environments, whether on your laptop, a testing server, or in production.

**Docker** remains the go-to platform for containerizing applications in 2025. It can run on any operating system, but as with most development tools, UNIX-based systems are preferred.

In this article, you'll learn how to install Docker on Ubuntu, start your first container, and manage and monitor running containers.

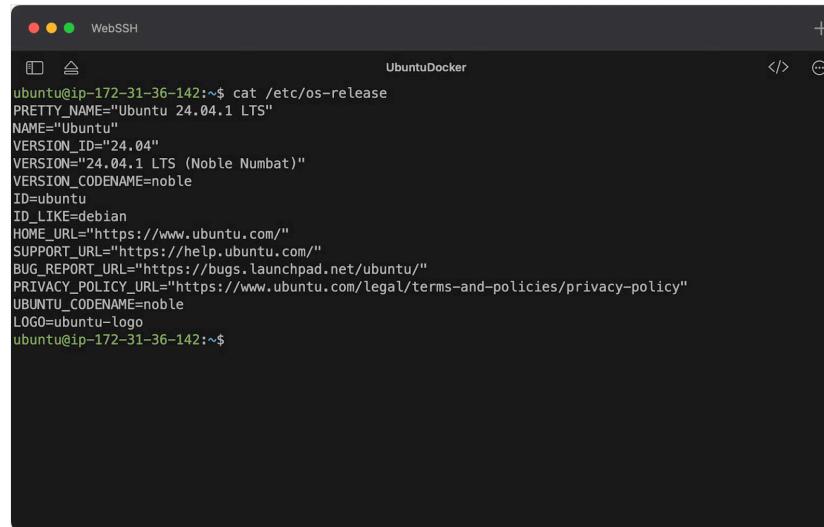
The setup is straightforward but involves a few steps. Let's dive in!

Is Docker the same as Kubernetes? [Here are all the differences every developer should know.](#)

## Preparing Your Ubuntu System

Before installing Docker on Ubuntu, there are a few housekeeping tasks to take care of. You'll handle all of them in this section.

To start, make sure you're running an Ubuntu system, either the Desktop or Server version. We're using *Ubuntu Server 24.04*:



```
ubuntu@ip-172-31-36-142:~$ cat /etc/os-release
PRETTY_NAME='Ubuntu 24.04.1 LTS'
NAME='Ubuntu'
VERSION_ID='24.04'
VERSION='24.04.1 LTS (Noble Numbat)'
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
ubuntu@ip-172-31-36-142:~$
```

Image 1 - Operating system details

If you see a similar output to the one shown in the image above, you're good to go.

### Check system requirements

Before installing Docker, ensure your system has enough resources to run it.

While Docker doesn't specify minimum requirements on its website (apart from compatible CPU architectures), we've found the following to be sufficient for running Docker itself:

- **Operating system:** Ubuntu Desktop/Server 20.04 or later
- **CPU architecture:** 64-bit (x86\_64 or arm64)
- **Memory:** 512 MB of RAM
- **Storage:** 5 GB

A machine with these specifications will allow you to run Docker, but not much else. Don't expect to run complex containers or multi-container applications.

## Update your Ubuntu system

We're running Docker on a fresh Ubuntu Server instance, so we'll provide all the necessary commands to set up Docker prerequisites.

First, update your system's package list and upgrade existing packages by running the following command:

```
sudo apt update && sudo apt upgrade -y
```



Explain code

POWERED BY datalab

This process may take a few minutes, but it ensures your package manager retrieves the latest package versions and security updates.

## Ensure prerequisites

Docker requires a few dependencies to install and function properly. Run this command to install them:

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



Explain code

POWERED BY datalab

In short, the above command enables package downloads over HTTPS and installs the tools necessary to fetch Docker GPG keys (which we'll discuss later).

And that's it—your system is now ready to install Docker!

## Installing Docker on Ubuntu

Once the system prerequisites are taken care of, installing Docker on Ubuntu is a four-step process.

### Step 1: Add Docker's official GPG key

Docker's GPG (GNU Privacy Guard) key is used to sign Docker packages and repositories, ensuring the integrity and authenticity of Docker images and packages.

Start by running the appropriate shell command to download the key and store a converted version compatible with APT in `/usr/share/keyrings`.

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



[Explain code](#)POWERED BY  datalab

## Step 2: Add Docker's official APT repository

Now, add the official Docker repository to your system.

The next command uses the previously stored GPG key to verify the repository's authenticity and then adds it to the system package sources:

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://mirrors.docker.com/api嘴角右下角" > /etc/apt/sources.list.d/docker.list
```

[Explain code](#)POWERED BY  datalab

After that, update your system's package list to include the new repository:

```
sudo apt update
```

[Explain code](#)POWERED BY  datalab

## Step 3: Install Docker

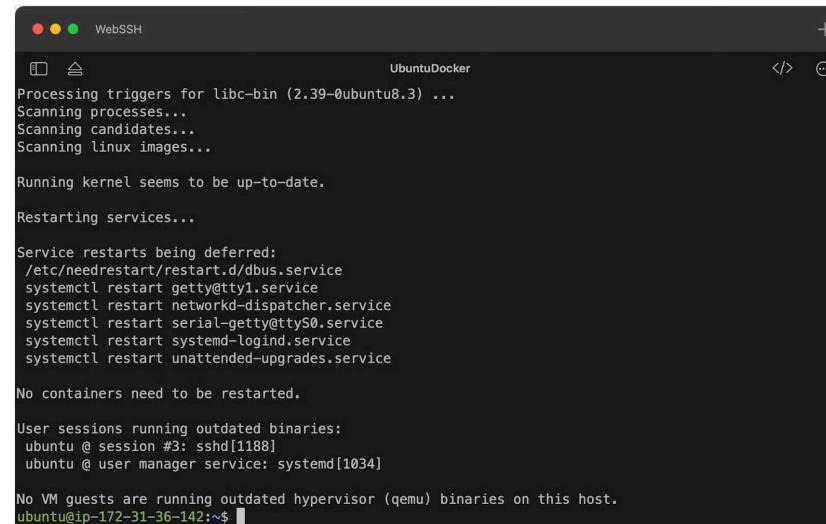
Now that the repository is configured, you're ready to install Docker on Ubuntu.

The installation command will install the core Docker engine, its command-line interface, and the runtime for managing container execution:

```
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

[Explain code](#)POWERED BY  datalab

Once the installation is complete, you should see an output confirming the successful installation:



```
Processing triggers for libc-bin (2.39-0ubuntu8.3) ...
Scanning processes...
Scanning candidates...
Scanning linux images...

Running kernel seems to be up-to-date.

Restarting services...

Service restarts being deferred:
/etc/needrestart/restart.d/dbus.service
systemctl restart getty@tty1.service
systemctl restart networkd-dispatcher.service
systemctl restart serial-getty@ttyS0.service
systemctl restart systemd-logind.service
systemctl restart unattended-upgrades.service

No containers need to be restarted.

User sessions running outdated binaries:
ubuntu @ session #3: sshd[1188]
ubuntu @ user manager service: systemd[1034]

No VM guests are running outdated hypervisor (qemu) binaries on this host.

ubuntu@ip-172-31-36-142:~$
```

*Image 2 - Docker installation results*

#### Step 4: Verify Docker installation

Docker should now be installed on your system.

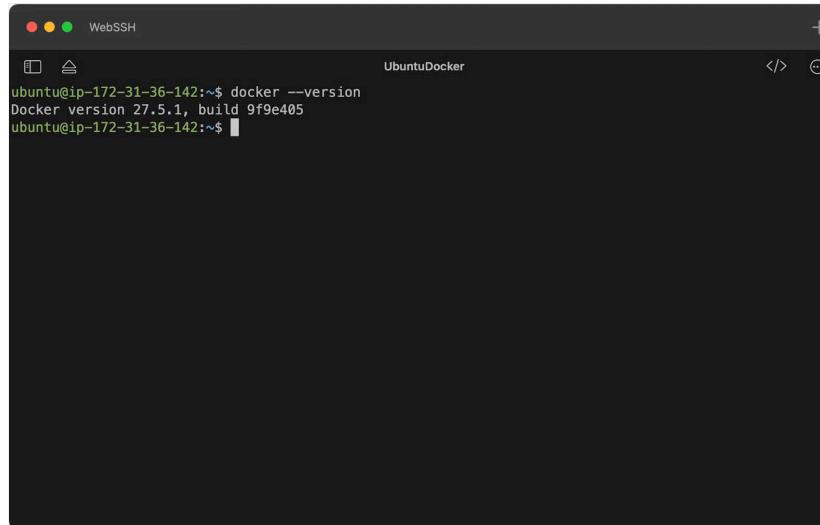
To verify, print the Docker version to the console. If the installation was successful, you should see a version number displayed:

```
docker --version
```



[Explain code](#)

POWERED BY  **databricks**



```
ubuntu@ip-172-31-36-142:~$ docker --version
Docker version 27.5.1, build 9f9e405
ubuntu@ip-172-31-36-142:~$
```

*Image 3 - Installed Docker version*

As of late January 2025, Docker version 27.5.1 is the latest, and we have it successfully running on Ubuntu Server.

## Post-Installation Steps

You now have Docker installed and are ready to start containerizing your applications.

But what is containerization? Our course in [Containerization and Virtualization concepts](#) will teach you the basics.

Before doing so, we recommend going through three additional configuration steps to get the most out of it. This section is for you if you want to [run Docker on system boot](#) or [manage images and containers as a non-root user](#).

### Step 1: Start and enable the Docker service

The Docker service is currently running on your system, but it won't start automatically after a reboot.

To ensure Docker starts automatically on boot, use `systemctl` to enable the service:

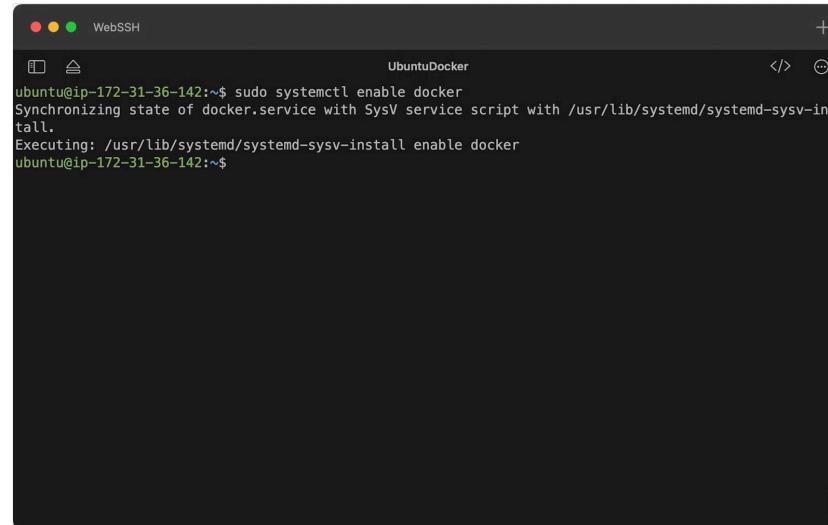
```
sudo systemctl enable docker
```



[Explain code](#)

POWERED BY datalab

You should see an output similar to ours:



```
ubuntu@ip-172-31-36-142:~$ sudo systemctl enable docker
Synchronizing state of docker.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable docker
ubuntu@ip-172-31-36-142:~$
```

Image 4 - Enabling Docker service on boot

In the future, if you ever see a message indicating that Docker isn't running (which shouldn't happen), you can **start Docker manually** using the following command:

```
sudo systemctl start docker
```



[Explain code](#)

POWERED BY  datalab

Docker will now remain up and running as long as your system is powered on.

## Step 2: Check Docker status

There are a few additional `systemctl` commands you can use to manage the Docker service.

For example, the `status` command shows the state of a service, which is most commonly *active*, *inactive*, or *failed*. Although there are more possible states, these three are the most frequent:

```
sudo systemctl status docker
```



[Explain code](#)

POWERED BY  datalab

```
ubuntu@ip-172-31-36-142:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
    Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
      Active: active (running) since Thu 2025-01-30 07:59:38 UTC; 2min 48s ago
        TriggeredBy: ● docker.socket
          Docs: https://docs.docker.com
        Main PID: 8003 (dockerd)
          Tasks: 8
        Memory: 30.9M (peak: 32.0M)
          CPU: 281ms
        CGroup: /system.slice/docker.service
                └─8003 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Jan 30 07:59:37 ip-172-31-36-142 systemd[1]: Starting docker.service - Docker Application Container Engine
Jan 30 07:59:37 ip-172-31-36-142 dockerd[8003]: time="2025-01-30T07:59:37.819315201Z" level=info msg="[graphdriver] using native storage driver"
Jan 30 07:59:37 ip-172-31-36-142 dockerd[8003]: time="2025-01-30T07:59:37.820213127Z" level=info msg="dockerd version 20.10.16"
Jan 30 07:59:37 ip-172-31-36-142 dockerd[8003]: time="2025-01-30T07:59:37.820286154Z" level=info msg="[graphdriver] using native storage driver"
Jan 30 07:59:38 ip-172-31-36-142 dockerd[8003]: time="2025-01-30T07:59:38.018032644Z" level=info msg="[graphdriver] using native storage driver"
Jan 30 07:59:38 ip-172-31-36-142 dockerd[8003]: time="2025-01-30T07:59:38.368826341Z" level=info msg="[graphdriver] using native storage driver"
Jan 30 07:59:38 ip-172-31-36-142 dockerd[8003]: time="2025-01-30T07:59:38.392598927Z" level=info msg="[graphdriver] using native storage driver"
Jan 30 07:59:38 ip-172-31-36-142 dockerd[8003]: time="2025-01-30T07:59:38.392679182Z" level=info msg="[graphdriver] using native storage driver"
Jan 30 07:59:38 ip-172-31-36-142 dockerd[8003]: time="2025-01-30T07:59:38.442921268Z" level=info msg="[graphdriver] using native storage driver"
Jan 30 07:59:38 ip-172-31-36-142 systemd[1]: Started docker.service - Docker Application Container Engine
lines 1-22/22 (END)
```

*Image 5 - Docker service status*

If your Docker service is active, all systems are running correctly.

If you see a different state for Docker or any other service, restarting it should resolve the issue:

```
sudo systemctl restart docker
```



[Explain code](#)

POWERED BY datalab

Once restarted, you can check the status again to confirm that the service is running.

### Step 3: Manage Docker as a non-root user

By default, running Docker commands requires `sudo`, which can be inconvenient. This means you must type `sudo` before any `docker` command.

Luckily, there's an easy way to allow non-root users to run Docker commands—by adding the current user to the `docker` group:

```
sudo usermod -aG docker $USER
```



[Explain code](#)

POWERED BY datalab

After running the above command, log out and log back in for the change to take effect. You should now be able to run Docker commands without superuser privileges.

We'll verify that in the next section.

## Testing Docker Installation

Docker is now installed and configured. It also runs automatically on boot and allows your user to run commands directly.

In this section, we'll wrap things up by running two containers—one for a simple "Hello, World!" example and another for a Python script.

### Step 1: Run a test Docker container

Docker provides a simple test image called `hello-world`, which is used solely to confirm that Docker is working properly.

Run the following command to start the image:

```
docker run hello-world
```



Explain code

POWERED BY datalab

Since the image is not found locally, Docker will automatically download it from the web. After a few moments, you should see an output similar to this:

```
● ● ● WebSSH
ubuntu@ip-172-31-36-142:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:d715f14f9eca81473d9112df50457893aa4d099adeb4729f679006bf5ea12407
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.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

ubuntu@ip-172-31-36-142:~$
```

Image 6 - Docker hello world example

In short, this confirms that Docker is installed correctly, the Docker daemon is running, and your system can pull images from Docker Hub.

## Step 2: Run your first "real" container

If you want to build images from scratch, this section is for you.

We'll build a simple image running Python 3.9 that, when launched, executes a custom Python script.

Start by creating a dedicated folder and an `app.py` file that will be executed:

```
mkdir python-docker-test && cd python-docker-test  
nano app.py
```



Explain code

POWERED BY datalab

The file contains only a simple `print()` command, but of course, you can expand it to include more complex logic:

```
print("Hello from Docker running on Ubuntu!")
```



Explain code

POWERED BY datalab

In the same folder, create a `Dockerfile`, which tells Docker how to build the image step by step:

```
nano Dockerfile
```



Explain code

POWERED BY datalab

These three lines are all you need:

```
FROM python:3.9-slim  
COPY app.py /app.py  
CMD ["python", "/app.py"]
```

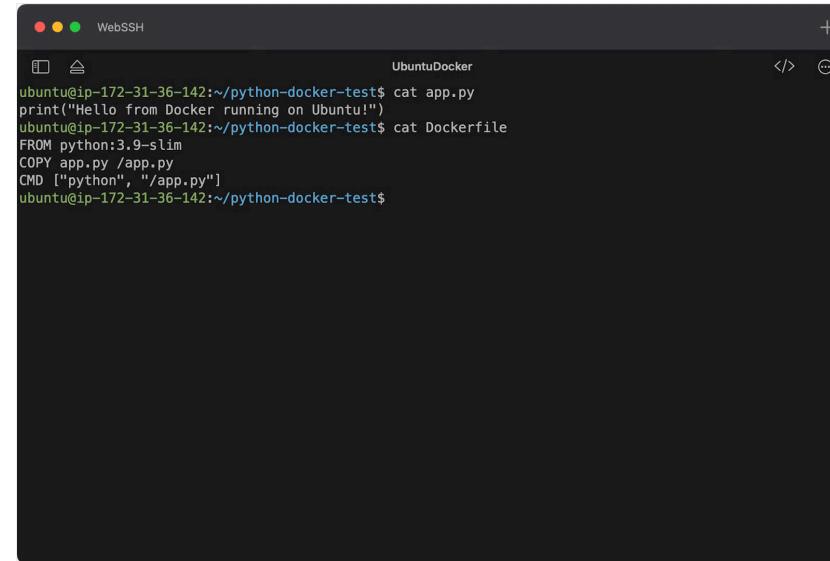


Explain code

POWERED BY datalab

In essence, Docker is instructed to fetch the `python:3.9-slim` image from Docker Hub, copy the `app.py` script to the container, and then run the script.

By now, you should have the following files in your working directory:

A screenshot of a terminal window titled "WebSSH". The window shows the command "cat Dockerfile" followed by the Dockerfile content:

```
FROM python:3.9-slim
COPY app.py /app.py
CMD ["python", "/app.py"]
```

Then it shows the command "cat app.py" followed by the Python script content:

```
print("Hello from Docker running on Ubuntu!")
```

Image 7 - Demo Python application contents

To build the image, run the following command:

```
docker build -t python-hello .
```



 Explain code

POWERED BY  dataLab

The command above creates a new image in the current directory (.) and assigns it the tag python-hello .

```

ubuntu@ip-172-31-36-142:~/python-docker-test$ docker build -t python-hello .
[+] Building 4.9s (7/7) FINISHED
=> [internal] load build definition from Dockerfile               docker:default
=> => transferring dockerfile: 104B                                0.1s
=> [internal] load metadata for docker.io/library/python:3.9-slim   0.0s
=> [internal] load .dockerrcignore                                1.2s
=> => transferring context: 2B                                  0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 79B                                0.0s
=> [1/2] FROM docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fc  3.4s
=> => resolve docker.io/library/python:3.9-slim@sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fc  0.0s
=> => sha256:bb8009c87ab69e751a1dd2c6c7f8abaae3d9fc@e072802d4a23c95594d16d 10.41kB / 10.41kB  0.0s
=> => sha256:ddb5f2e39ec00c79c1207f182c60e03ddff417525b56ef467e1519706792cd 1.75kB / 1.75kB  0.0s
=> => sha256:453d3342b002fa5f904ba0cd72a07acc5121641d20776f3e64339842f275d38 5.28kB / 5.28kB  0.0s
=> => sha256:af302e5c37e9dc1be2eadc8f5059d82a914066b541b0d1a6daa91d0cc5505 28.21MB / 28.21MB  0.6s
=> => sha256:1da0723265ec311debcf6bec17d4fae5f1e5f7809fc4378aac265cef238f1c 3.51MB / 3.51MB  0.4s
=> => sha256:4f4cb1a24c6f1a92f204ba0bb6d2a7c941a853c83161ffa38bbfa1214488 14.93MB / 14.93MB  0.7s
=> => sha256:c876ae22765eda125855eb121718c3f8f07bd800dae0ad4e68e716571961f37 249B / 249B  0.6s
=> => extracting sha256:af302e5c37e9dc1dbe2eadc8f5059d82a914066b541b0d1a6daa91d0cc55057d 1.5s
=> => extracting sha256:1da0723265ec311debcf6bec17d4fae5f1e5f7809fc4378aac265cef238f1c 0.2s
=> => extracting sha256:4f4cb1a24c6f1a92f204ba0bb6d2a7c941a853c83161ffa38bbfa121448861 0.8s
=> => extracting sha256:c876ae22765e4a125855eb121718c3f8f07bd800dae0ad4e68e716571961f37 0.0s
=> [2/2] COPY app.py /app.py                                     0.1s
=> exporting to image                                         0.1s
=> => exporting layers                                       0.0s
=> => writing image sha256:1f144165a1d7039f2b2df6b4220fd4263697d47d029538b90aa4c86da36416df 0.0s
=> => naming to docker.io/library/python-hello                0.0s
ubuntu@ip-172-31-36-142:~/python-docker-test$

```

*Image 8 - Building a Docker image*

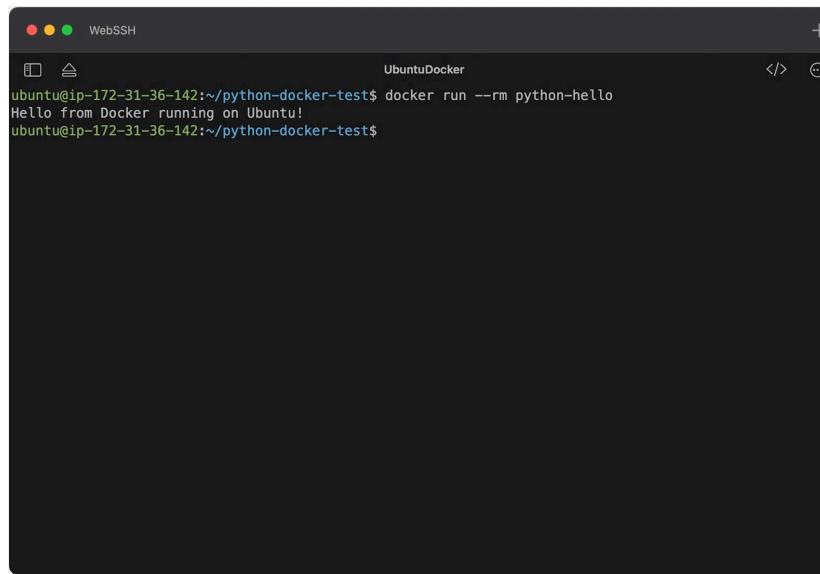
To run the image, use the `docker run` command followed by the image tag. The optional `--rm` flag ensures the container is removed after it exits, which happens immediately after the Python script finishes execution:

```
docker run --rm python-hello
```



Explain code

POWERED BY datalab



The screenshot shows a terminal window titled "WebSSH". The terminal session is titled "UbuntuDocker". The command run was "docker run --rm python-hello". The output of the command is "Hello from Docker running on Ubuntu!". The terminal prompt is "ubuntu@ip-172-31-36-142:~/python-docker-test\$".

*Image 9 - Running a Python script in Docker*

And just like that, you've successfully containerized your first Python application with Docker running on Ubuntu!

If you're looking for a machine learning application of Docker, DataCamp's [Containerization: Docker and Kubernetes for Machine Learning](#) is a recommended follow-up read.

## Updating and Uninstalling Docker on Ubuntu

Running Docker in your own environment gives you full control but also makes you responsible for keeping up with the latest updates, new features, and security patches.

This section explains how to perform basic tasks such as updating Docker and completely removing it from your system.

### Updating Docker

Updating Docker comes down to two main steps:

- Updating the package list to fetch the latest Docker version.
- Upgrading Docker (core, CLI, and container management runtime) to the newest version.

Run the following commands in the terminal to update Docker on Ubuntu:

```
sudo apt update  
sudo apt upgrade -y docker-ce docker-ce-cli containerd.io  
docker --version
```



[Explain code](#)POWERED BY  datalab

The `docker -- version` command displays the currently installed version. This is the easiest way to verify whether Docker has been updated successfully.

## Uninstalling Docker

If you need to remove Docker from your system, you should start by uninstalling Docker and related packages and removing any remaining Docker-related files and configurations.

Additionally, you can optionally remove the Docker repository from APT sources.

The following four commands will uninstall Docker from your Ubuntu system, but keep in mind that only the first three are required:

```
sudo apt remove -y docker-ce docker-ce-cli containerd.io
sudo rm -rf /var/lib/docker
sudo rm -rf /var/lib/containerd

# Optional
sudo rm /etc/apt/sources.list.d/docker.list
```

[Explain code](#)POWERED BY  datalab

After running these commands, issuing `docker --version` should return an error indicating that Docker is no longer installed.

## Best Practices for Docker on Ubuntu

Managing Docker requires ongoing maintenance, as it's your responsibility to remove unused resources and monitor performance.

In this section, we'll cover two essential housekeeping tasks you should perform regularly while running Docker on Ubuntu.

### Regularly clean up Docker resources

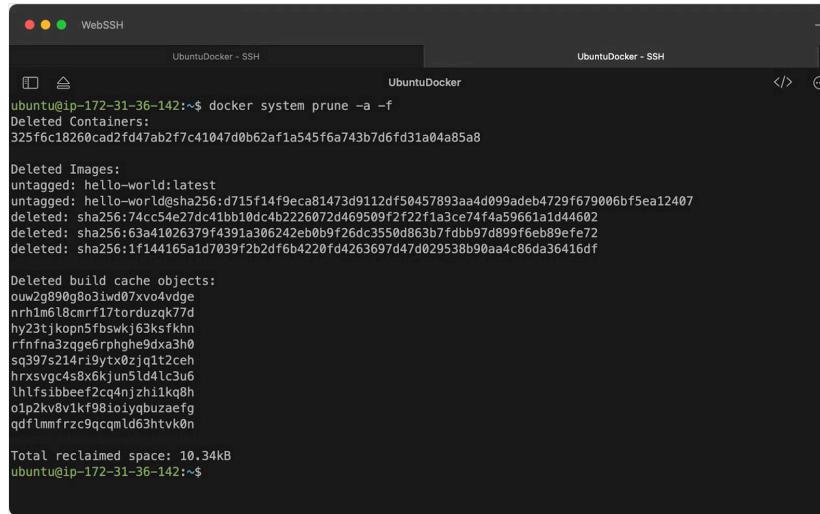
Over time, unused Docker images, containers, networks, and volumes can accumulate, consuming significant disk space.

Cleaning up these resources is known as **pruning**, though a detailed explanation is outside the scope of this article.

Just remember this command — it will remove all unused containers, images, volumes, and networks at once:

```
docker system prune -a -f
```



[Explain code](#)POWERED BY  datalab


```

ubuntu@ip-172-31-36-142:~$ docker system prune -a -f
Deleted Containers:
325f6c18260cad2fd47ab2f7c41047d0b62af1a545f6a743b7d6fd31a04a85a8

Deleted Images:
untagged: hello-world:latest
untagged: hello-worldsha256:d715f14f9eca81473d9112df50457893aa4d099adeb4729f679006bf5ea12407
deleted: sha256:74cc54e27dc41bb10dc4b226072d469509f2f22f1a3ce74f4a59661a1d44602
deleted: sha256:74cc54e27dc41bb10dc4b226072d469509f2f22f1a3ce74f4a59661a1d44602
deleted: sha256:1f144165a1d7039f2b2df6b4220fd4263697d47d029538b90aa4c86da36416df

Deleted build cache objects:
ouw2g890g8631wd07xv04vdge
nrh1m6l8cmrf17torduzqk77d
hy23tjkopn5fbwkj63ksfkhn
rfmna3zqge6rphghe90xa3h0
sq397s214ri9ytx0zjqltzceh
hrxsvgc4s8x6kjun51d4lcl3u6
1hlfs1bbeef2cq4njzh1kq8h
01p2kv8v1kf981oiyabuzaefg
qdflumfrzc9qcqmld63rtvk0n

Total reclaimed space: 10.34kB
ubuntu@ip-172-31-36-142:~$
```

*Image 10 - Resource pruning*

We've only reclaimed 10 kB, but in a more "serious" environment, this command can often free up tens of gigabytes of disk space.

## Secure your Docker installation

We've already covered two important security topics — keeping Docker updated and managing superuser permissions.

In addition to these, here are a few more steps to enhance Docker security on Ubuntu:

- **Limit container privileges:** Avoid running containers with the `--privileged` flag whenever possible, as it disables many security checks that isolate containers from the host system.
- **Use only official and trusted images:** Don't pull images from unverified third-party registries. Stick to trusted sources like Docker Hub and private registries.
- **Enable Docker Content Trust (DCT):** Setting this environment variable ensures only signed and verified images are used.

## Monitor Docker performance

The Docker CLI includes powerful tools to help you monitor container performance and identify potential issues.

Before diving into them, let's modify the `app.py` file to keep it running after the container starts:

```
import time  
  
time.sleep(10000)  
print("Hello from Docker running on Ubuntu!")
```

[Explain code](#)

POWERED BY  datalab

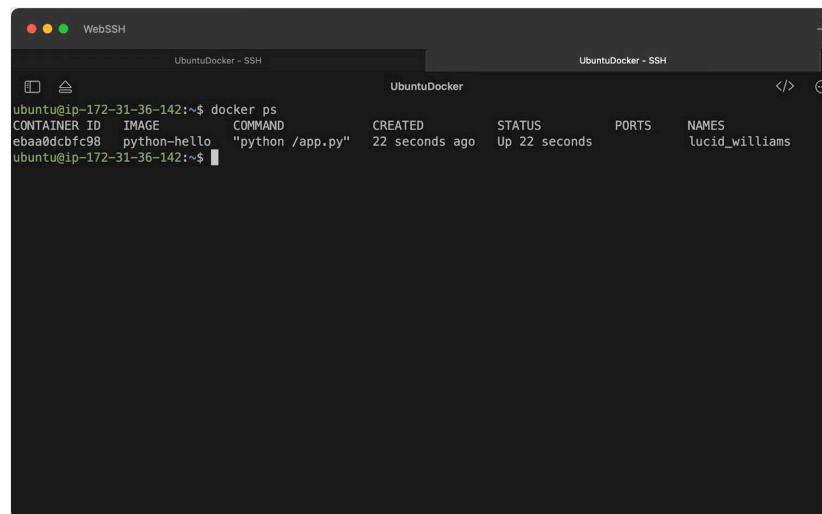
The reason is simple — we need a running container to monitor its performance.

Let's start with the `ps` command. Simply put, it lists all currently running containers:

```
docker ps
```

[Explain code](#)

POWERED BY  datalab



A screenshot of a terminal window titled "UbuntuDocker - SSH". The window shows the command `docker ps` being run, which lists one container named "lucid\_williams". The container was created 22 seconds ago and is currently up for 22 seconds. The command used was "python /app.py".

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
ebaadcbfc98	python-hello	"python /app.py"	22 seconds ago	Up 22 seconds		lucid_williams

Image 11 - Running Docker containers

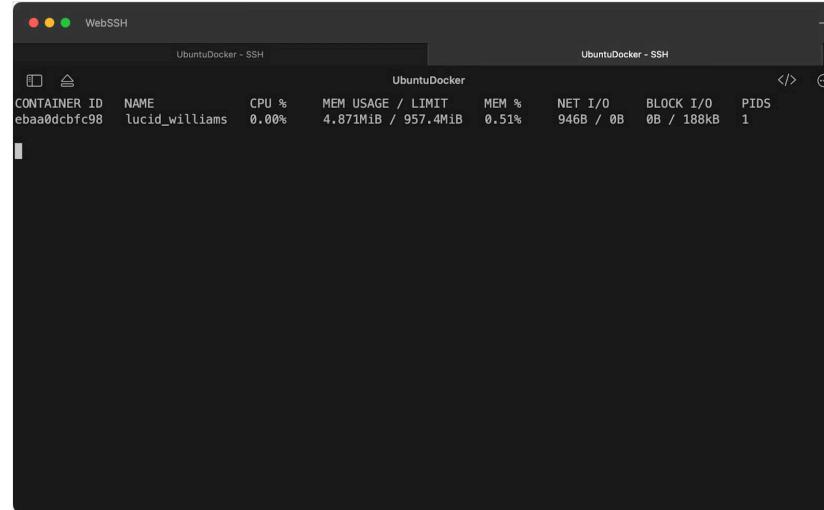
While the output doesn't reveal much, the `CONTAINER ID` field is crucial for further inspection.

Now, let's explore the `stats` command, which displays real-time resource usage, including CPU, memory, and network consumption for all running containers:

```
docker stats
```

[Explain code](#)

POWERED BY  datalab

*Image 12 - Container stats*

If you need a detailed breakdown of a container's resource usage and configuration, use the `inspect` command.

Just provide the container ID:

```
docker inspect <container_id>
```



[Explain code](#)

POWERED BY databricks

```
ubuntu@ip-172-31-36-142:~$ docker inspect ebaa0dcfc98
[
    {
        "Id": "ebaa0dcfc98b1562e67d86f4fc45eb42c426f066afcd2506f8f8ea52bb4f9e16",
        "Created": "2025-01-30T08:20:10.316094678Z",
        "Path": "python",
        "Args": [
            "/app.py"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 8967,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2025-01-30T08:20:10.4388636Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:05cd45fa64e45af53e19109fdff4918c18185ebffbbb442b0d8673a3a724f87",
        "ResolvConfPath": "/var/lib/docker/containers/ebaa0dcfc98b1562e67d86f4fc45eb42c426f066afcd2506f8f8ea52bb4f9e16/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/ebaa0dcfc98b1562e67d86f4fc45eb42c426f066afcd2506f8f8ea52bb4f9e16/hostname"
    }
]
```

*Image 13 - Container details*

If you're experiencing issues with a Docker container, the `logs` command can help diagnose the problem and potentially identify its cause.

It also requires the container ID:

```
docker logs <container_id>
```



Explain code

POWERED BY datalab

The screenshot shows a terminal window titled "UbuntuDocker - SSH" with two tabs. The left tab is titled "UbuntuDocker - SSH" and the right tab is also titled "UbuntuDocker". The terminal displays the command "ubuntu@ip-172-31-36-142:~\$ docker logs ebba0dcbfc98" followed by a blank line. The terminal has a dark background and light-colored text. There are standard terminal icons at the top and bottom.

*Image 14 - Container logs*

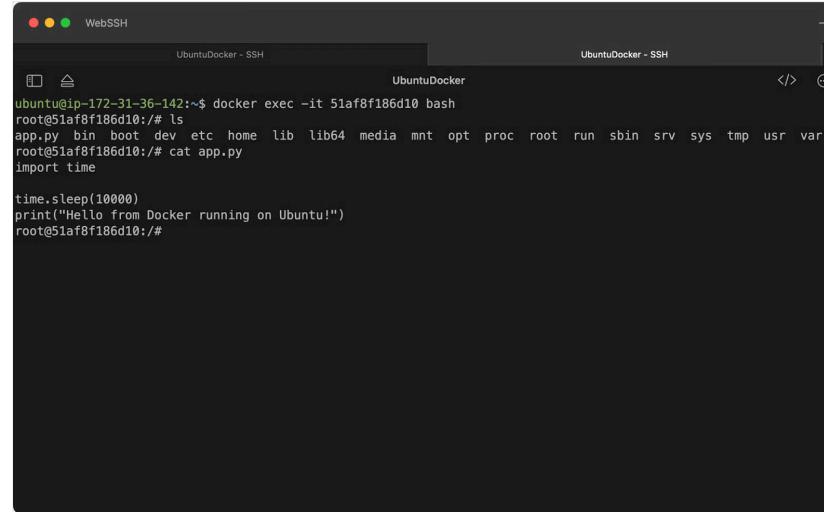
Sometimes, you may need to connect directly to a running container. If you know the container ID, this is straightforward:

```
docker exec -it <container_id> bash
```



Explain code

POWERED BY datalab



```
UbuntuDocker - SSH          UbuntuDocker - SSH
ubuntu@ip-172-31-36-142:~$ docker exec -it 51af8f186d10 bash
root@51af8f186d10:/# ls
app.py bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
root@51af8f186d10:/# cat app.py
import time

time.sleep(10000)
print("Hello from Docker running on Ubuntu!")
root@51af8f186d10:/#
```

*Image 15 - Container logs*

The previous command opens a shell inside the specified Docker container, allowing you to access or modify files directly.

## Summing up Docker on Ubuntu

Today, you've learned how to install, configure, and manage Docker on Ubuntu.

If you've followed every step, you will have Docker running on your system, which starts automatically on boot and doesn't require superuser permissions to manage containers. You've also learned how to verify Docker installation, run test containers, and even build your first custom Python containerized application.

It's a lot, but it's only just **the beginning**. There's much more to this whole containerization story, and we highly recommend checking out our official resources to get proficient in Docker:

- [Introduction to Docker](#)
- [Intermediate Docker](#)
- [10 Docker Project Ideas: From Beginner to Advanced](#)
- [How to Learn Docker from Scratch: A Guide for Data Professionals](#)

## Master Docker and Kubernetes

Learn the power of Docker and Kubernetes with an interactive track to build and deploy applications in modern environments.

[Start Track for Free](#)

## FAQs

### What is Docker used for?

Docker is used to automate the deployment of applications into lightweight containers, which package an app and its dependencies into an isolated environment that's reproducible and independent of the host environment.

### Is Ubuntu good for Docker?

### Why should I use Docker?

### What is the difference between Docker and virtual machines?

### How do I know if Docker is working properly?



AUTHOR

**Dario Radečić**

in

Senior Data Scientist based in Croatia. Top Tech Writer with over 700 articles published, generating more than 10M views. Book Author of Machine Learning Automation with TPOT.

**TOPICS**[Cloud](#)   [Docker](#)   [Kubernetes](#)

### Training more people?

Get your team access to the full DataCamp for business platform.

[For Business](#)

For a bespoke solution [book a demo](#).

## Learn more about Docker with these courses!

➡ TRACK

### Containerization and Virtualization with Docker and Kubernetes

⌚ 0 min

Learn the power of Docker and Kubernetes, this interactive track will allow you to build and deploy applications in modern environments.

[See Details →](#)

[Start Course](#)

➡ COURSE

### Introductic

⌚ 4 hr ⚡ 38.

Gain an intrc

[See Details →](#)

[See More →](#)

## Related



BLOG

How to Learn Docker from Scratch: A Guide for Data...

CHEAT-SHEET

Docker for Data Science Cheat Sheet

TUTORIAL

Docker for Data Science: An Introduction

[See More →](#)

## Grow your data skills with DataCamp for Mobile

Make progress on the go with our mobile courses and daily 5-minute coding challenges.



## LEARN

[Learn Python](#)

[Learn AI](#)

[Learn Power BI](#)

[Learn Data Engineering](#)

[Assessments](#)

[Career Tracks](#)

[Skill Tracks](#)

[Courses](#)

[Data Science Roadmap](#)

## DATA COURSES

[Python Courses](#)

[R Courses](#)

[SQL Courses](#)

[Power BI Courses](#)

[Tableau Courses](#)

[Alteryx Courses](#)

[Azure Courses](#)

[AWS Courses](#)

[Google Sheets Courses](#)

[Excel Courses](#)

[AI Courses](#)

[Data Analysis Courses](#)[Data Visualization Courses](#)[Machine Learning Courses](#)[Data Engineering Courses](#)[Probability & Statistics Courses](#)

## DATALAB

[Get Started](#)[Pricing](#)[Security](#)[Documentation](#)

## CERTIFICATION

[Certifications](#)[Data Scientist](#)[Data Analyst](#)[Data Engineer](#)[SQL Associate](#)[Power BI Data Analyst](#)[Tableau Certified Data Analyst](#)[Azure Fundamentals](#)[AI Fundamentals](#)

## RESOURCES

[Resource Center](#)[Upcoming Events](#)[Blog](#)[Code-Alongs](#)

[Tutorials](#)[Docs](#)[Open Source](#)[RDocumentation](#)[Book a Demo with DataCamp for Business](#)[Data Portfolio](#)

## PLANS

[Pricing](#)[For Students](#)[For Business](#)[For Universities](#)[Discounts, Promos & Sales](#)[Expense DataCamp](#)[DataCamp Donates](#)

## FOR BUSINESS

[Business Pricing](#)[Teams Plan](#)[Data & AI Unlimited Plan](#)[Customer Stories](#)[Partner Program](#)

## ABOUT

[About Us](#)[Learner Stories](#)[Careers](#)[Become an Instructor](#)

[Press](#)[Leadership](#)[Contact Us](#)[DataCamp Español](#)[DataCamp Português](#)[DataCamp Deutsch](#)[DataCamp Français](#)

## SUPPORT

[Help Center](#)[Become an Affiliate](#)[Privacy Policy](#)   [Cookie Notice](#)   [Do Not Sell My Personal Information](#)   [Accessibility](#)   [Security](#)   [Terms of Use](#)

© 2025 DataCamp, Inc. All Rights Reserved.