



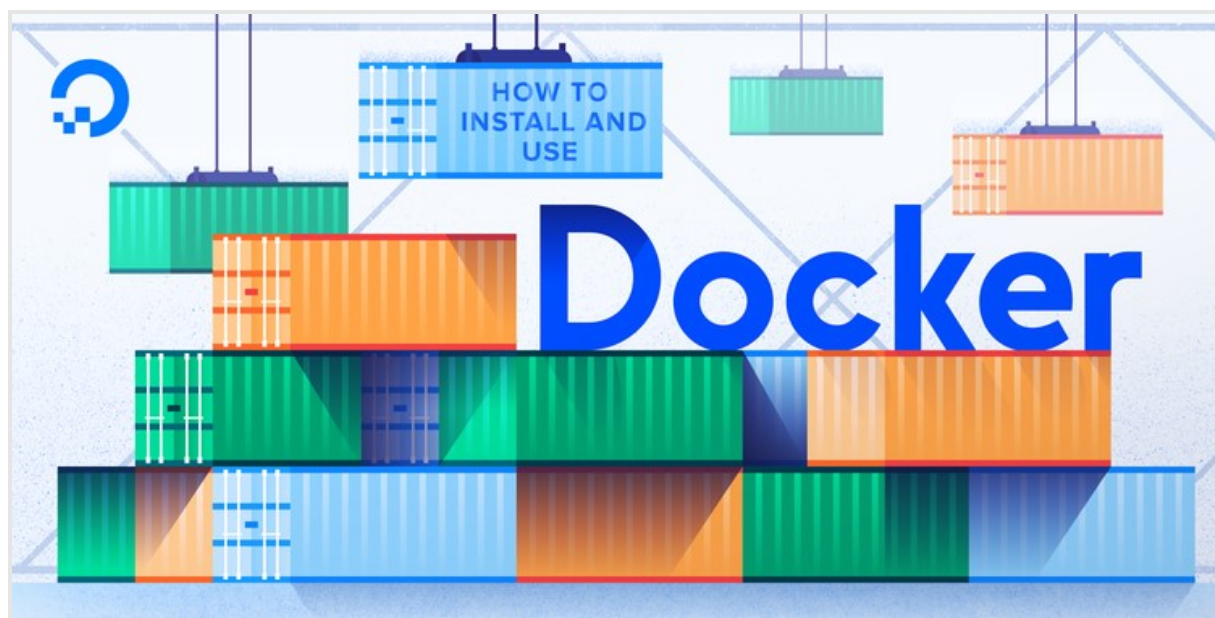
Subscribe



Share



Contents



How To Install and Use Docker on Ubuntu 16.04



125

Updated October 19, 2018

1.6m

DOCKER

UBUNTU

UBUNTU 16.04

By: finid

Not using **Ubuntu 16.04**? Choose a different version:

Introduction

Docker is an application that makes it simple and easy to run application processes in a container, which are like virtual machines, only more portable, more resource-friendly, and more dependent on the host operating system. For a detailed introduction to the

[SCROLL TO TOP](#)

different components of a Docker container, check out [The Docker Ecosystem: An Introduction to Common Components](#).

There are two methods for installing Docker on Ubuntu 16.04. One method involves installing it on an existing installation of the operating system. The other involves spinning up a server with a tool called [Docker Machine](#) that auto-installs Docker on it.

In this tutorial, you'll learn how to install and use it on an existing installation of Ubuntu 16.04.

Prerequisites

To follow this tutorial, you will need the following:

- One Ubuntu 16.04 server set up with a non-root user with sudo privileges and a basic firewall, as explained in the [Initial Setup Guide for Ubuntu 16.04](#)
- An account on [Docker Hub](#) if you wish to create your own images and push them to Docker Hub, as shown in Steps 7 and 8

Step 1 — Installing Docker

The Docker installation package available in the official Ubuntu 16.04 repository may not be the latest version. To get this latest version, install Docker from the official Docker repository. This section shows you how to do just that.

First, in order to ensure the downloads are valid, add the GPG key for the official Docker repository to your system:

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

Add the Docker repository to APT sources:

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



Next, update the package database with the Docker packages from the newly added repo:

SCROLL TO TOP

```
$ sudo apt-get update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu 16.04 repo:

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

You should see output similar to the follow:

Output of apt-cache policy docker-ce

```
docker-ce:
  Installed: (none)
  Candidate: 18.06.1~ce~3-0~ubuntu
  Version table:
     18.06.1~ce~3-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable amd64 Packages
```

Notice that docker-ce is not installed, but the candidate for installation is from the Docker repository for Ubuntu 16.04 (xenial).

Finally, install Docker:

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

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

```
$ sudo systemctl status docker
```

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: en
   Active: active (running) since Thu 2018-10-18 20:28:23 UTC; 355 ago
```

SCROLL TO TOP

```
Main PID: 13412 (dockerd)
  CGroup: /system.slice/docker.service
          └─13412 /usr/bin/dockerd -H fd://
              └─13421 docker-containerd --config /var/run/docker/containerd/containerd
```

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)

By default, running the `docker` command requires root privileges — that is, you have to prefix the command with `sudo`. It can also be run by a user in the **docker** group, which is automatically created during the installation of Docker. 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:

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

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

```
$ su - ${USER}
```

You will be prompted to enter your user's password to continue. Afterwards, you can confirm that your user is now added to the `docker` group by typing:

```
$ id -nG
```

SCROLL TO TOP

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:

```
$ sudo usermod -aG docker username
```

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

Step 3 — Using the Docker Command

With Docker installed and working, now's the time to become familiar with the command line utility. Using `docker` consists of passing it a chain of options and commands followed by arguments. The syntax takes this form:

```
$ docker [option] [command] [arguments]
```

To view all available subcommands, type:

```
$ docker
```

As of Docker 18.06.1, the complete list of available subcommands includes:

Output

<code>attach</code>	Attach local standard input, output, and error streams to a running container
<code>build</code>	Build an image from a Dockerfile
<code>commit</code>	Create a new image from a container's changes
<code>cp</code>	Copy files/folders between a container and the local filesystem
<code>create</code>	Create a new container
<code>diff</code>	Inspect changes to files or directories on a container's filesystem
<code>events</code>	Get real time events from the server
<code>exec</code>	Run a command in a running container
<code>export</code>	Export a container's filesystem as a tar archive
<code>history</code>	Show the history of an image

SCROLL TO TOP

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 switches available to a specific command, type:

```
$ docker docker-subcommand --help
```

To view system-wide information about Docker, use:

```
$ docker info
```

Step 4 — Working with Docker Images

Docker containers are run from Docker images. By default, it pulls these images from Docker Hub, a Docker registry managed by Docker, the company behind the Docker project. Anybody can build and host their Docker images on Docker Hub, so most applications and Linux distributions you'll need to run Docker containers have images that are hosted on Docker Hub.

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

```
$ docker run hello-world
```

In the output, you should see the following message, which indicates that Docker is working correctly:

Output

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

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:

```
$ docker search ubuntu
```

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
ubuntu	Ubuntu is a Debian-based Lin
dorowu/ubuntu-desktop-lxde-vnc	Ubuntu with openssh-server a
rastasheep/ubuntu-sshd	Dockerized SSH service, buil
consol/ubuntu-xfce-vnc	Ubuntu container with "headl
ansible/ubuntu14.04-ansible	Ubuntu 14.04 LTS with ansibl
ubuntu-upstart	Upstart is an event-based re

SCROLL TO TOP

neurodebian	NeuroDebian provides neurosc
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5	ubuntu-16-nginx-php-phpmyadm
ubuntu-debootstrap	debootstrap --variant=minbas
nuagebec/ubuntu	Simple always updated Ubuntu
tutum/ubuntu	Simple Ubuntu docker images
i386/ubuntu	Ubuntu is a Debian-based Lin
1and1internet/ubuntu-16-apache-php-7.0	ubuntu-16-apache-php-7.0
ppc64le/ubuntu	Ubuntu is a Debian-based Lin
eclipse/ubuntu_jdk8	Ubuntu, JDK8, Maven 3, git,
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4	ubuntu-16-nginx-php-5.6-worc
codenvy/ubuntu_jdk8	Ubuntu, JDK8, Maven 3, git,
darksheer/ubuntu	Base Ubuntu Image -- Updatec
pivotaldata/ubuntu	A quick freshening-up of the
1and1internet/ubuntu-16-sshd	ubuntu-16-sshd
smartentry/ubuntu	ubuntu with smartentry
ossobv/ubuntu	Custom ubuntu image from scr
paasmule/bosh-tools-ubuntu	Ubuntu based bosh-cli
1and1internet/ubuntu-16-healthcheck	ubuntu-16-healthcheck
pivotaldata/ubuntu-gpdb-dev	Ubuntu images for GPDB devel



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. Try this with the `ubuntu` image, like so:

```
$ docker pull ubuntu
```

After an image has been downloaded, you may then run a container using the downloaded image with the `run` subcommand. 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:

```
$ docker run ubuntu
```

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

```
$ docker images
```

SCROLL TO TOP

The output should look similar to the following:

Output

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	ea4c82dcd15a	16 hours ago	85.1 MB
hello-world	latest	4ab4c602aa5e	5 weeks ago	1.84 MB



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.

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:

```
$ docker run -it ubuntu
```

Note: The default behavior for the `run` command is to start a new container. Once you run the preceding the command, you will open up the shell interface of a second `ubuntu` container.

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

Output

```
root@9b0db8a30ad1:/#
```

Note: Remember the container id in the command prompt. In the preceding example, it is `9b0db8a30ad1`. 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:

```
root@9b0db8a30ad1:/# apt-get update
```

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

```
root@9b0db8a30ad1:/# apt-get install -y nodejs
```

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

```
root@9b0db8a30ad1:/# node -v
```

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

```
Output  
v8.10.0
```

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

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

```
$ docker ps
```

You will see output similar to the following:

[SCROLL TO TOP](#)

Output

CONTAINER ID	IMAGE	COMMAND	CREATED
--------------	-------	---------	---------

In this tutorial, you started three containers; one from the `hello-world` image and two from the `ubuntu` image. These 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:

```
$ docker ps -a
```

You'll see output similar to this:

Output

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
9b0db8a30ad1	ubuntu	"/bin/bash"	21 minutes ago	Exited
d7851eb12e23	ubuntu	"/bin/bash"	24 minutes ago	Exited
d54945b6510b	hello-world	"/hello"	32 minutes ago	Exited



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

```
$ docker ps -l
```

Output

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
9b0db8a30ad1	ubuntu	"/bin/bash"	22 minutes ago	Exited



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 `9b0db8a30ad1`:

```
$ docker start 9b0db8a30ad1
```

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

[SCROLL TO TOP](#)

Output

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
9b0db8a30ad1	ubuntu	"/bin/bash"	23 minutes ago	Up

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 `xenodochial_neumann`:

```
$ docker stop xenodochial_neumann
```

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.

```
$ docker rm youthful_roentgen
```

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

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.

[SCROLL TO TOP](#)

To do this, commit the changes to a new Docker image instance using the following command structure:

```
$ docker commit -m "What did you do to the image" -a "Author Name" container-id re
```

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:

```
$ docker commit -m "added node.js" -a "sammy" d9b100f2f636 sammy/ubuntu-nodejs
```

Note: When you *commit* an image, the new image is saved locally, that is, on your computer. Later in this tutorial, you'll learn how to push an image to a Docker registry like Docker Hub so that it can be assessed and used by you and others.

After that operation is completed, listing the Docker images now on your computer should show the new image, as well as the old one that it was derived from:

```
$ docker images
```

The output should be similar to this:

Output

REPOSITORY	TAG	IMAGE ID	CREATED
sammy/ubuntu-nodejs	latest	6a1784a63edf	2 minutes ago
ubuntu	latest	ea4c82dcd15a	17 hours ago
hello-world	latest	4ab4c602aa5e	5 weeks ago

In the above 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

SCROLL TO TOP

were made. In this example, the change was that Node.js was installed. Next time you need to run a container using Ubuntu with Node.js 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

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 another 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.

This section shows you how to push a Docker image to Docker Hub. To learn how to create your own private Docker registry, check out [How To Set Up a Private Docker Registry on Ubuntu 14.04](#).

To push your image, first log into Docker Hub:

```
$ docker login -u docker-registry-username
```

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:

```
$ docker tag sammy/ubuntu-nodejs docker-registry-username/ubuntu-nodejs
```

Then you can push your own image using:

SCROLL TO TOP

```
$ docker push docker-registry-username/ubuntu-nodejs
```

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

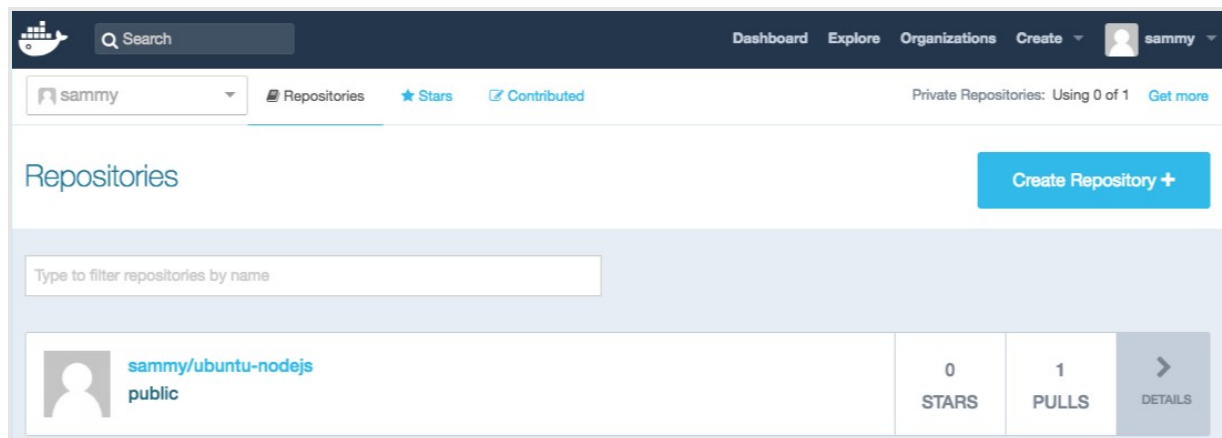
```
$ docker push sammy/ubuntu-nodejs
```

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 repository [docker.io/sammy/ubuntu-nodejs]
1aa927602b6a: Pushed
76c033092e10: Pushed
2146d867acf3: Pushed
ae1f631f14b7: Pushed
102645f1cf72: Pushed
latest: digest: sha256:2be90a210910f60f74f433350185feadbbdaca0d050d97181bf593dd8519
```

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



If a push attempt results in the following error, it is likely that you are not logged in:

Output

```
The push refers to a repository [docker.io/sammy/ubuntu-nodejs]
e3fbbfb44187: Preparing
5f70bf18a086: Preparing
a3b5c80a4eba: Preparing
```

[SCROLL TO TOP](#)

```
7f18b442972b: Preparing
3ce512daaf78: Preparing
7aae4540b42d: Waiting
unauthorized: authentication required
```

Log in, then repeat the push attempt.

Conclusion

In this tutorial, you've learned the basics to get you started working with Docker on Ubuntu 16.04. Like most open source projects, Docker is built from a fast-developing codebase, so make a habit of visiting the project's [blog page](#) for the latest information.

For further exploration, check out the [other Docker tutorials](#) in the DigitalOcean Community.

By: finid

♡ Upvote (125)

📄 Subscribe

🔗 Share



Editor:
Tammy Fox



We just made it easier for you to deploy faster.

[TRY FREE](#)

[SCROLL TO TOP](#)

Related Tutorials

How To Install Docker Compose on Ubuntu 18.04

How To Remove Docker Images, Containers, and Volumes

Naming Docker Containers: 3 Tips for Beginners

How to Manually Set Up a Prisma Server on Ubuntu 18.04

How To Use Traefik as a Reverse Proxy for Docker Containers on Debian 9

39 Comments

Leave a comment...

Log In to Comment

^ [zetadisseny](#) May 23, 2016



0 I only can say that: I get at your feet. Excellent tutorial

^ [panousis](#) May 29, 2016



0 Thank you very much for your article. It's really excellent!. Please keep going

^ [oded](#) June 2, 2016



4 Its probably better to use the Ubuntu APT tools to update the APT configuration. Instead of `echo` , use `apt-add-repository` to manipulate the apt sources:

```
sudo apt-add-repository 'deb https://apt.dockerproject.org/repo ubuntu-xenial
```



^ [MelissaAnderson](#) MOD November 3, 2016



0 Thanks for taking time to point that out. I've made the update.

^ [iambobby](#) January 13, 2017



0 If you choose to use apt-add-repository you need to make sure software-properties-common is installed...

^ [davemanginelli](#) June 17, 2016



2 I had some trouble with docker failing to run after some updates to a 16.04 droplet. The solution was to install a dependency not mentioned above but recommended in Docker's instructions for 16.04:

```
sudo apt-get install linux-image-extra-$(uname -r)
```

After which I was able to get Docker running again.

^ [dekkermichelle1](#) June 22, 2016



0 Hi,

I have followed the tutorial, but I get this message

```
_dockermachine_ps1: command not found
```

^ [tapeason](#) July 22, 2016



0 Great tutorial, easy to follow, and no problems along the way. Thank you!

^ [florianfalk](#) August 1, 2016



0 Well, it's not working for me. After I try running a ubuntu container, I get the following Error:

SCROLL TO TOP

docker: failed to register layer: devmapper: Error mounting '/dev/mapper/docker-8:2-526216-d9cc121a2c92ab980cbf1218ba4f467d6cc27248993b1a79165e7602113d88b0' on
'/var/lib/docker/devicemapper/mnt/d9cc121a2c92ab980cbf1218ba4f467d6cc27248993b1a79165e7602113d88b0': invalid argument.

Have anybody a solution?



^ [florianfalk](#) August 1, 2016



0 Well it doesn't work for me. It get this error when I try to run a container:

```
docker: failed to register layer: devmapper: Error mounting
'/dev/mapper/docker-8:2-526216-
d9cc121a2c92ab980cbf1218ba4f467d6cc27248993b1a79165e7602113d88b0' on
'/var/lib/docker/devicemapper/mnt/d9cc121a2c92ab980cbf1218ba4f467d6cc27248993b1
invalid argument.
```

Does anybody has any solution?



^ [zx1986](#) August 24, 2016



0 I failed on Linode ...

```
root@ubuntu:/var/log# service docker status
```

- docker.service - Docker Application Container Engine

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

Active: failed (Result: signal) since Wed 2016-08-24 15:09:17 UTC; 2min 11s ago

Docs: <https://docs.docker.com>

Main PID: 3881 (code=killed, signal=KILL)

```
Aug 24 13:07:30 ubuntu systemd[1]: Starting Docker Application Container Engine...
```

```
Aug 24 13:07:30 ubuntu dockerd[3881]: time="2016-08-24T13:07:30.731767656Z"
```

```
level=info msg="libcontainerd: new containerd process, pid: 3984"
```

```
Aug 24 15:07:47 ubuntu dockerd[3881]: time="2016-08-24T15:07:47.334383528Z"
```

```
level=info msg="Processing signal 'terminated'"
```

```
Aug 24 15:09:17 ubuntu systemd[1]: docker.service: State 'stop-sigterm' timed out. Killing.
```

```
Aug 24 15:09:17 ubuntu systemd[1]: docker.service: Main process exited, code=killed,
status=9/KILL
```

```
Aug 24 15:09:17 ubuntu systemd[1]: Stopped Docker Application Container Engine.
```

```
Aug 24 15:09:17 ubuntu systemd[1]: docker.service: Unit entered failed state.
```

```
Aug 24 15:09:17 ubuntu systemd[1]: docker.service: Failed with result 'signal'.
```

SCROLL TO TOP

^ [gurukumara](#) August 29, 2016

0 I like the Clear and Simple explanation. Excellent!

^ [waryak](#) September 16, 2016

0 in September 2016, problems with 1st step - docker installation

^ [ganeshb](#) October 12, 2016

0 The push refers to a repository [docker.io/ganesh08/grasp/ubuntu-python]
An image does not exist locally with the tag: ganesh08/grasp/ubuntu-python

--- this was an error while iam pushing an image into the docker.
is there any solution?

^ [couchpotatoe](#) October 23, 2016

0 I run into this issue, complaining about libsystemd-journal0 being required. when I run the "sudo apt-get install -y docker-engine" step. I haven't been able to install libsystemd-journal0 with apt-get. Is there another recommended way to install this? Or did I miss a step?

amato@amato-VirtualBox:~\$ sudo apt-get install -y docker-engine
Reading package lists... Done
Building dependency tree

Reading state information... Done

Some packages could not be installed. This may mean that you have requested an impossible situation or if you are using the unstable distribution that some required packages have not yet been created or been moved out of Incoming.

The following information may help to resolve the situation:

The following packages have unmet dependencies:

docker-engine : Depends: libsystemd-journal0 (>= 201) but it is not installable

Recommends: aufs-tools but it is not going to be installed

Recommends: cgroupfs-mount but it is not going to be installed or
cgroup-lite but it is not going to be installed

Recommends: git

E: Unable to correct problems, you have held broken packages.

^ [mleewise](#) November 18, 2016

0 I created a user specifically for docker. Would someone concerned about security frown upon adding that user to the docker group to avoid having to use sudo? I already did the standard stuff for restricting ssh access (no login, ssh required). Thanks !

^ [Squonk42](#) December 9, 2016

0 Extremely useful tutorial, thank you!

^ [billyallen](#) January 6, 2017

0 excellent

^ [Allwo](#) March 5, 2017

0 Thank you very much for this tutorial!

However, I get an error that is, after of a couple of attempts, is really frustrating me. I have been installing docker a lot of times already, also on this OS, but this never happened.


I am stuck at the end of Step 1, when docker-engine cannot be installed:

```
~# systemctl status docker.service
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor pre
   Active: failed (Result: exit-code) since So 2017-03-05 17:47:20 CET; 32
   Docs: https://docs.docker.com
   Main PID: 18194 (code=exited, status=1/FAILURE)
```

```
dockerd[18194]: time="2017-03-05T17:47:20.567753592+01:00" level=error msg
dockerd[18194]: time="2017-03-05T17:47:20.569299675+01:00" level=error msg
dockerd[18194]: time="2017-03-05T17:47:20.591796895+01:00" level=info msg=
dockerd[18194]: time="2017-03-05T17:47:20.592394882+01:00" level=warning r
dockerd[18194]: time="2017-03-05T17:47:20.592410368+01:00" level=warning r
dockerd[18194]: time="2017-03-05T17:47:20.592421460+01:00" level=warning r
dockerd[18194]: time="2017-03-05T17:47:20.592427398+01:00" level=warning r
dockerd[18194]: time="2017-03-05T17:47:20.592458649+01:00" level=warning r
dockerd[18194]: time="2017-03-05T17:47:20.592490516+01:00" level=warning r
dockerd[18194]: Error starting daemon: Devices cgroup isn't mounted
```

I added root to the group, also I found the advice to add
`GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"` to the file
`/etc/default/grub`, but that file does not exist!

Thank you for your help!

 [jayb7c00ad853d0b1e62f6bfce](#) April 14, 2017

0 p80.pool.sks-keyservers.net does not resolve, so you can't get keys this way anymore.

Load More Comments



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



Copyright © 2019 DigitalOcean™ Inc.

[Community](#) [Tutorials](#) [Questions](#) [Projects](#) [Tags](#) [Newsletter](#) [RSS](#) 

[Distros & One-Click Apps](#) [Terms, Privacy, & Copyright](#) [Security](#) [Report a Bug](#)
[Write for DONations](#) [Shop](#)

[SCROLL TO TOP](#)