Arch Linux • Containers • Dev Ops

Arch Linux Docker Tutorial

2 years ago • by David Morelo

What Is Docker?

If you read technology news websites, you've most likely heard about Docker and all the wonderful things this open platform that allows developers and sysadmins to build, ship, and run distributed applications can do. But why just read about Docker when you can try it first-hand? In this tutorial, we'll teach you how to install and configure Docker on Arch Linux, and we'll also show you a few examples of what you can do with Docker.

On its official website, Docker is described as “world’s leading software container platform.” Okay, but what is a container? A container is a self-contained bundle of libraries and settings that guarantees that a piece of software will always run exactly the same, regardless of where it’s deployed.

In other words, containers, and Docker, solve the fragmentation issue that has been plaguing the Unix world for decades. Finally, developers can easily take software from development machines to remote servers and know with certainty that everything will run as expected.

Docker was initially released in 2013 by the company Docker, Inc. The person who started Docker is Solomon Hykes, who was the co-founder and CEO of dotCloud, a platform-as-a-service company. Several dotCloud engineers contributed to Docker, including Andrea Luzzardi and Francois-Xavier Bourlet.

Just three years after Docker’s initial release, an analysis revealed that major contributors to Docker include Red Hat, IBM, Microsoft, Huawei, Google, and Cisco. In a short time, Docker has caught the attention of some of the largest companies in the world and established itself as the leading software container platform.

Docker Versus Virtualization

Unlike virtual machines, which get virtual access to host resources through a hypervisor, Docker containers run natively on the host machine’s kernel, each running as a discrete process and taking no more memory than any other executable.

Docker containers don’t run any guest operating system. Instead, they contain only an executable and its package dependencies. This makes containers much less resource demanding and allows containerized applications to run anywhere.

How to Install Docker on Arch Linux

0. Before You Begin

Even though Arch Linux still allows i686 installations to receive upgraded packages, in accordance with the distribution’s plans to phase out the support of this architecture, Docker supports only 64-bit systems. That dusty old machine you may have in your closet may be great for some retro-gaming, but you won’t be able to run Docker on it.

1. Enable the Loop Module

Besides the 64-bit architecture, Docker also depends on the loop module, which is a block device that maps its data blocks not to a physical device such as a hard disk or optical disk drive, but to the blocks of a regular file in a filesystem or to another block device, according to Linux Programmer's Manual.

Docker should enable the loop module automatically during installation. Check if ”loop” has been loaded as a kernel module:

```
# lsmod | grep loop
```

If the loop module has been loaded, you may skip to the next step. Otherwise, run the following two commands:

```
# tee /etc/modules-load.d/loop.conf <<< "loop"
# modprobe loop
```

The first command passes the word “loop” to the standard input of the command on the left, which is the command tee. Then, tee writes the word “loop” to the file loop.conf. The modprobe command adds the loop module to the Linux kernel.

2. Install Docker

You can choose whether you want to install a stable version of Docker from the Community repository or a development version from AUR. The former is called simply docker, and the latter is called docker-git. If you’re new to using Docker in general or just using Docker on Arch Linux, we highly recommend you install the stable package:

```
# pacman -S docker
```

3. Start and Enable Docker

Before you can use Docker on Arch Linux, you have to start and enable the Docker daemon using system:

```
# systemctl start docker.service
# systemctl enable docker.service
```

The first command immediately starts the Docker daemon, and the second command ensures that the daemon will start automatically on bootup.

Optionally, use the following command to verify the installation and activation:

```
# docker info
```

Note that you can run Docker only as root. To run Docker as a regular user, add yourself to the docker group:

```
# groupadd docker

# gpasswd -a user docker [replace user with your
username]
```

The first command creates a new group called docker, and the second command adds a user to the group. Don't forget to re-login to apply the changes.

Post-Install Configuration

Provided that your host machine is properly configured to begin with, there's not much left to do after the installation before you can start using Docker on Arch Linux.

You may want to change the location of Docker images, however. Docker stores images by default in /var/lib/docker. To change their location, first stop the Docker daemon:

```
# systemctl stop docker.service
```

Then, move the images to the target destination. Finally, add the following parameter to the ExecStart in /etc/systemd/system/docker.service.d/docker-storage.conf:

```
ExecStart=/usr/bin/dockerd --data-
```

```
root=/path/to/new/location/docker -H fd://
```

For more post-install configuration options, see Docker's official Arch wiki page

Using Docker on Arch Linux

With Docker installed and configured, it's time to finally have some fun with it.

First Steps

To see what Docker can do, ask it to list all available commands:

```
# docker
```

You can also ask Docker to tell you its version or give you system-wide information:

```
# docker version
```

```
# docker info
```

Downloading Docker Images

When you're ready to try something more interesting, you may download an x86_64 Arch Linux image:

```
# docker pull base/archlinux
```

If you would like to download some other Docker image, search for it using the following command (make sure to replace [image name] with your preferred search query:

```
# docker search [image name]
```

As you experiment with Docker, your collection of Docker images will naturally increase, and the amount of available storage space will shrink. When Docker starts occupying too much space, you may want to change its default storage location and move it to a different hard drive or partition. By default, Docker stores images and containers in /var/lib/docker. To set a new storage location, stop the Docker daemon:

```
# systemctl stop docker.service
```

Next, create a drop-in file “docker.conf” at a new drop-in directory /etc/systemd/system/docker.service.d. All files with the suffix “.conf” from the new drop-in directory will be parsed after the original configuration file is parsed, allowing you to override its settings without having to modify it directly.

```
# mkdir /etc/systemd/system/docker.service.d
# touch
```

```
/etc/systemd/system/docker.service.d/docker.conf
```

Next, open the newly created drop-in file in your favorite text editor and add the following lines:

```
[Service]

ExecStart=

ExecStart=/usr/bin/dockerd --graph="/mnt/new_volume"
--storage-driver=devicemapper
```

Don’t forget to change “new_volume” to your preferred new storage location and “devicemapper” to your current storage driver, which controls how images and containers are stored and managed on your Docker host. You can find out what storage driver is currently used by Docker using the following command, which you should be already familiar with:

```
# docker info
```

The only thing remaining is to reload service daemon to scan for new or changed units and start Docker again:

```
# systemctl daemon-reload

# systemctl start docker.service
```

Creating New Containers

Once you've downloaded your first Docker image, you can use it to create a new container by specifying a command to run using the image:

```
# docker run [image name] [command to run]
```

If the container suddenly stops, you can start it again:

```
# docker run [container ID]
```

And if you want it to stop, you can do that as well:

```
# docker stop [container ID]
```

From time to time, you may want to commit a container's file changes or settings into a new image. List all running Docker containers to find the container that you would like to commit into a new image:

```
# docker ps
```

Issue the following command to commit the changes and create a new image:

```
# docker commit [container ID] [image name]
```

Just keep in mind that when you commit a container's file changes or settings into a new image, the newly created image will not include any data contained in volumes mounted inside the container.

Finally, you can easily delete a container and start from scratch:

```
# docker rm [container ID]
```

Monitoring Docker Containers

There are several available options how to collect useful metrics from Docker containers. One readily available option is the docker stats command, which gives access to CPU, memory, network and disk utilization for all of the containers running on your host.

```
# docker stats
```

If you run multiple Docker containers at the same time, you may want to restrict the output of the command to only one or more containers by specifying container ids, separated by a space:

```
# docker stats [container ID] [container ID]
[container ID]
```

To get a one-time snapshot of current container resource usage, add the –no-stream option:

```
# docker stats --no-steam
```

You can also use the –all option, which displays stopped containers:

```
# docker stats --all
```

Apart from docker stats, you can also use cAdvisor (a container monitoring tool from Google), Prometheus (an open source monitoring system and time series database), or Agentless System Crawler (ASC) (a cloud monitoring tool from IBM with support for containers), among other services.

Networking Configuration

By default, Docker creates three networks automatically, and you can list them using the following command:

```
# docker network ls
```

You should see something like this:

```
NETWORK ID NAME DRIVER
7fca4eb8c647 bridge bridge
9f904ee27bf5 none null
cf03ee007fb4 host host
```

The bridge network corresponds to the docker0 network, which is present in all Docker installations. The none network doesn’t have any access to the external network, but it can be used for running batch jobs. Finally, the host network adds a container on the host’s network stack without any isolation between the host machine and the container.

Use the following command to see information about the default bridge network:

```
# docker network inspect bridge
```

Docker recommends using user-defined bridge networks to control which containers can communicate with each other. Docker doesn’t limit how many new networks users can create using the default networks as templates, and containers can be connected to multiple networks at the same time. Create a new bridge network:

```
# docker network create --driver bridge bridge_new
```

And inspect it:

```
# docker network inspect bridge_new
```

Launch a busybox (or any other) container connected to the newly created network:

```
# docker run --network= bridge_new -itd --name=
[container ID] busybox
```

SSH Into a Container

To SSH into Docker containers, you could install an SSH server in the images you wish to ssh-into and run each container mapping the ssh port to one of the host's ports. However, this is not the right approach. "In order to reduce complexity, dependencies, file sizes, and build times, you should avoid installing extra or unnecessary packages just because they might be 'nice to have,'" states the Docker user guide.

Instead, it's a better idea to use a containerized SSH server and stick it to any running container. The only requirement is that the container has bash. User Jeroen Peeters provides the following example on Stack Exchange and encourages readers to visit his GitHub for more information:

```
$ docker run -d -p 2222:22 \
 -v /var/run/docker.sock:/var/run/docker.sock \
 -e CONTAINER=my-container -e AUTH_MECHANISM=noAuth
\
 jeroenpeeters/docker-ssh
$ ssh -p 2222 localhost
```

Alternatively, you can use the docker exec command to run a command in a running container. For example:

```
# docker exec -it <mycontainer> bash
```

Sharing Data Between a Docker Container and the Host

You can use Docker volumes to share files between a host system and the Docker container. This can be handy, for example, when you want to create a permanent copy of a log file to analyze it later.

First, create a directory on the host in a location that a Docker user will have access to:

```
# mkdir ~/container-share
```

Then, attach the host directory to the container volume located in the /data directory within the container:

```
#docker run -d -P --name test-container -v
/home/user/container-share:/data archlinux
```

You will see the ID of the newly created container. Gain shell access to the container:

```
docker attach [container ID]
```

Once you've entered the command above, you will be the data directory we added at container run-time. Any file you add to this directory will be available from the host folder.

Conclusion

Docker is an immensely powerful software technology, and this tutorial is only an introduction for those who have never used it before. You can learn much more about Docker from the official documentation, which is always kept up to date. If you would like to learn how to define and deploy applications with Docker, check the Get started with Docker guide. If you run into any problems with Docker, the Troubleshooting page is the best place where to look for a solution.

ABOUT THE AUTHOR

David Morelo

Content writer and copywriter, researcher, wannabe linguistic, part-time marketer, gym rat, sometimes annoying but always loving boyfriend.

I was born and raised in the Czech Republic, where I studied English and Japanese philology at the Palacký University in Olomouc, the second oldest university in the Czech Republic and the largest university in Moravia, one of the historical Czech lands.

View all posts