DigitalOcean

Subscribe

How to Install and Use Docker on Debian 9

3

Posted September 4, 2018 13.5k DOCKER DEBIAN 9

By: Brian Hogan By: Hanif Jetha

Not using **Debian 9**? Choose a different version:

A previous version of this tutorial was written by finid.

Introduction

Docker is an application that simplifies the process of managing application processes in *containers*. Containers let you run your applications in resource-isolated processes. They're similar to virtual machines, but containers are more portable, more resource-friendly, and more dependent on the host operating system.

For a detailed introduction to the different components of a Docker container, check out The Docker Ecosystem: An Introduction to Common Components.

In this tutorial, you'll install and use Docker Community Edition (CE) on Debian 9. You'll install Docker itself, work with containers and images, and push an image to a Docker Repository.

Prerequisites

To follow this tutorial, you will need the following:

- One Debian 9 server set up by following the Debian 9 initial server setup guide, including a sudo non-root user and a firewall.
- 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 Debian repository may not be the latest version. To ensure we get the latest version, we'll install Docker from the official Docker repository. To do that, we'll add a new package source, add the GPG key from Docker to ensure the downloads are valid, and then install the package.

First, update your existing list of packages:

```
$ sudo apt update
```

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

```
$ sudo apt install apt-transport-https ca-certificates curl gnupg2 software-properties-comm
```

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

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

Add the Docker repository to APT sources:

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

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

```
$ sudo apt update
```

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

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

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

Output of apt-cache policy docker-ce

```
docker-ce:
  Installed: (none)
  Candidate: 18.06.1~ce~3-0~debian
```

```
  Version table:
     18.06.1~ce~3-0~debian 500
        500 https://download.docker.com/linux/debian stretch/stable amd64 Packages
```

Notice that `docker-ce` is not installed, but the candidate for installation is from the Docker repository for Debian 9 (`stretch`).

Finally, install Docker:

```
$ sudo apt install 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: enabled)
   Active: active (running) since Thu 2018-07-05 15:08:39 UTC; 2min 55s ago
     Docs: https://docs.docker.com
 Main PID: 21319 (dockerd)
   CGroup: /system.slice/docker.service
           ├─21319 /usr/bin/dockerd -H fd://
           └─21326 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

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

```
Output
docker: Cannot connect to the Docker daemon. Is the docker daemon running on this host?.
See 'docker run --help'.
```

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

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

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

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

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

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

```
$ id -nG
```

```
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** group. If you choose not to, please prepend the commands with `sudo`.

Let's explore the `docker` command next.

Step 3 — Using the Docker Command

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, the complete list of available subcommands includes:

Output

```
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
```

```
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes
```

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

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

To view system-wide information about Docker, use:

```
$ docker info
```

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

Step 4 — Working with Docker Images

Docker containers are built from Docker images. By default, Docker pulls these images from Docker Hub, a Docker registry managed by Docker, the company behind the Docker project. Anyone can host their Docker images on Docker Hub, so most applications and Linux distributions you'll need will have images hosted there.

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

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

The output will indicate that Docker in working correctly:

```
Output
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
9db2ca6ccae0: Pull complete
Digest: sha256:4b8ff392a12ed9ea17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc
Status: Downloaded newer image for hello-world:latest

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

Docker was initially unable to find the `hello-world` image locally, so it downloaded the image from Docker Hub, which is the default repository. Once the image downloaded, Docker created a

container from the image and the application within the container executed, displaying the message.

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

```
$ docker search ubuntu
```

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

```
Output
NAME                                                      DESCRIPTION
ubuntu                                                    Ubuntu is a Debian-based Linux opera
dorowu/ubuntu-desktop-lxde-vnc                            Ubuntu with openssh-server and NoVNC
rastasheep/ubuntu-sshd                                    Dockerized SSH service, built on top
consol/ubuntu-xfce-vnc                                    Ubuntu container with "headless" VNC
ansible/ubuntu14.04-ansible                               Ubuntu 14.04 LTS with ansible
ubuntu-upstart                                            Upstart is an event-based replacemer
neurodebian                                               NeuroDebian provides neuroscience re
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5      ubuntu-16-nginx-php-phpmyadmin-mysql
ubuntu-debootstrap                                        debootstrap --variant=minbase --comp
nuagebec/ubuntu                                           Simple always updated Ubuntu docker
tutum/ubuntu                                              Simple Ubuntu docker images with SSH
i386/ubuntu                                               Ubuntu is a Debian-based Linux opera
1and1internet/ubuntu-16-apache-php-7.0                    ubuntu-16-apache-php-7.0
ppc64le/ubuntu                                            Ubuntu is a Debian-based Linux opera
eclipse/ubuntu_jdk8                                       Ubuntu, JDK8, Maven 3, git, curl, nm
darksheer/ubuntu                                          Base Ubuntu Image -- Updated hourly
codenvy/ubuntu_jdk8                                       Ubuntu, JDK8, Maven 3, git, curl, nm
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4         ubuntu-16-nginx-php-5.6-wordpress-4
pivotaldata/ubuntu                                        A quick freshening-up of the base Ub
1and1internet/ubuntu-16-sshd                              ubuntu-16-sshd
ossobv/ubuntu                                             Custom ubuntu image from scratch (ba
smartentry/ubuntu                                         ubuntu with smartentry
1and1internet/ubuntu-16-healthcheck                       ubuntu-16-healthcheck
pivotaldata/ubuntu-gpdb-dev                               Ubuntu images for GPDB development
paasmule/bosh-tools-ubuntu                                Ubuntu based bosh-cli
...
```

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

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

```
$ docker pull ubuntu
```

You'll see the following output:

```
Output
Using default tag: latest
latest: Pulling from library/ubuntu
6b98dfc16071: Pull complete

4001a1209541: Pull complete
6319fc68c576: Pull complete
b24603670dc3: Pull complete
97f170c87c6f: Pull complete
Digest: sha256:5f4bdc3467537cbbe563e80db2c3ec95d548a9145d64453b06939c4592d67b6d
Status: Downloaded newer image for ubuntu:latest
```

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

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

```
$ docker images
```

The output should look similar to the following:

```
Output
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              16508e5c265d        13 days ago         84.1MB
hello-world         latest              2cb0d9787c4d        7 weeks ago         1.85kB
```

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

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

Step 5 — Running a Docker Container

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
```

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

```
Output
root@d9b100f2f636:/#
```

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

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

```
root@d9b100f2f636:/# apt update
```

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

```
root@d9b100f2f636:/# apt install nodejs
```

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

```
root@d9b100f2f636:/# 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:

```
Output
CONTAINER ID        IMAGE               COMMAND             CREATED
```

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

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

```
$ docker ps -a
```

You'll see output similar to this:

```
d9b100f2f636        ubuntu              "/bin/bash"         About an hour ago   Exited (0)
01c950718166        hello-world         "/hello"            About an hour ago   Exited (0)
```

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

```
$ docker ps -l
```

```
$ CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
$ d9b100f2f636        ubuntu              "/bin/bash"         About an hour 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 `d9b100f2f636`:

```
$ docker start d9b100f2f636
```

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

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
d9b100f2f636        ubuntu              "/bin/bash"         About an hour ago   Up 8 second
```

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 `sharp_volhard`:

```
$ docker stop sharp_volhard
```

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 festive_williams
```

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.

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

```
$ docker commit -m "What you did to the image" -a "Author Name" container_id repository/ne
```

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
```

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

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

```
$ docker images
```

You'll see output like this:

```
Output
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
sammy/ubuntu-nodejs   latest              7c1f35226ca6        7 seconds ago       179MB
ubuntu                  latest              113a43faa138        4 weeks ago         81.2MB
hello-world             latest              e38bc07ac18e        2 months ago        1.85kB
```

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

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

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

Step 8 — Pushing Docker Images to a Docker Repository

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

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 may push your own image using:

```
$ docker push docker-registry-username/docker-image-name
```

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

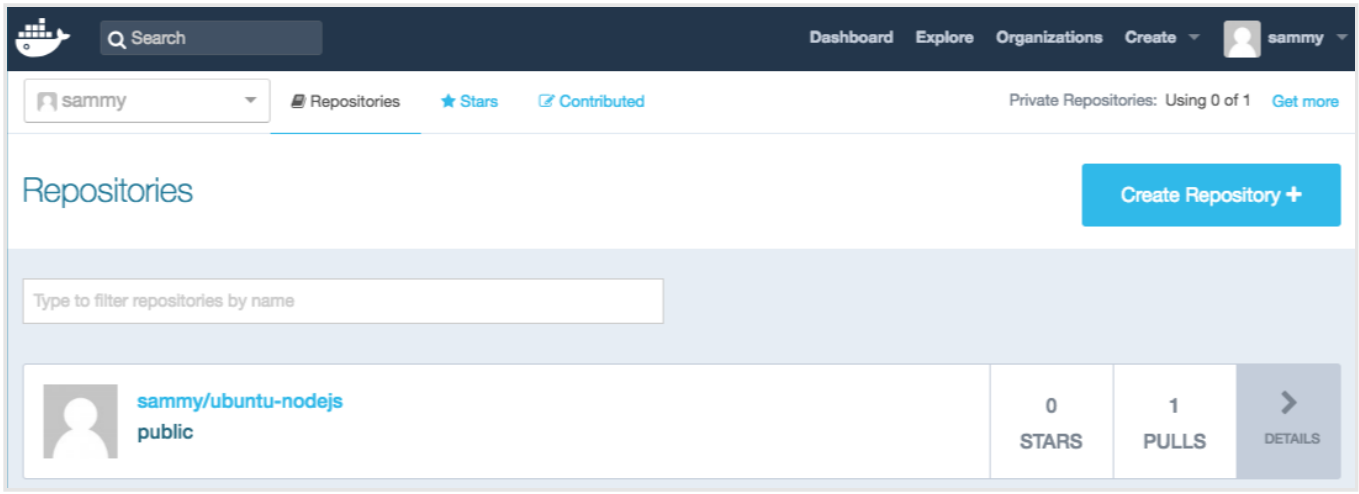
```
$ docker push sammy/ubuntu-nodejs
```

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 a repository [docker.io/sammy/ubuntu-nodejs]
e3fbbfb44187: Pushed
5f70bf18a086: Pushed
a3b5c80a4eba: Pushed
7f18b442972b: Pushed
3ce512daaf78: Pushed
7aae4540b42d: Pushed

...
```

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



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

```
Output
The push refers to a repository [docker.io/sammy/ubuntu-nodejs]
e3fbbfb44187: Preparing
5f70bf18a086: Preparing
a3b5c80a4eba: Preparing
7f18b442972b: Preparing
3ce512daaf78: Preparing
```

```
7aae4540b42d: Waiting
unauthorized: authentication required
```

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

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

Conclusion

In this tutorial you installed Docker, worked with images and containers, and pushed a modified image to Docker Hub. Now that you know the basics, explore the other Docker tutorials in the DigitalOcean Community.

By: Brian Hogan By: Hanif Jetha

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 Use Traefik as a Reverse Proxy for Docker Containers on Ubuntu 18.04

How To Provision and Manage Remote Docker Hosts with Docker Machine on Ubuntu 18.04

0 Comments

Leave a comment...

Log In to Comment