INTRODUCTION

TO

Bobby Iliev

- **This version was published on October 27 2021**

This is an open-source introduction to Docker guide that will help you learn the basics of Docker and how to start using containers for your SysOps, DevOps, and Dev projects. No matter if you are a DevOps/SysOps engineer, developer, or just a Linux enthusiast, you will most likely have to use Docker at some point in your career.

The guide is suitable for anyone working as a developer, system administrator, or a DevOps engineer and wants to learn the basics of Docker.

My name is Bobby Iliev, and I have been working as a Linux DevOps Engineer since 2014. I am an avid Linux lover and supporter of the open-source movement philosophy. I am always doing that which I cannot do in order that I may learn how to do it, and I believe in sharing knowledge.

I think it's essential always to keep professional and surround yourself with good people, work hard, and be nice to everyone. You have to perform at a consistently higher level than others. That's the mark of a true professional.

For more information, please visit my blog at https://bobbyiliev.com, follow me on Twitter @bobbyiliev_ and YouTube.

This book is made possible thanks to these fantastic companies!

The Streaming Database for Real-time Analytics.

Materialize is a reactive database that delivers incremental view updates. Materialize helps developers easily build with streaming data using standard SQL.

DigitalOcean is a cloud services platform delivering the simplicity developers love and businesses trust to run production applications at scale.

It provides highly available, secure, and scalable compute, storage, and networking solutions that help developers build great software faster.

Founded in 2012 with offices in New York and Cambridge, MA, DigitalOcean offers transparent and affordable pricing, an elegant user interface, and one of the largest libraries of open source resources available.

For more information, please visit https://www.digitalocean.com or follow @digitalocean on Twitter.

If you are new to DigitalOcean, you can get a free $100 credit and spin up your own servers via this referral link here:

Free $100 Credit For DigitalOcean

The DevDojo is a resource to learn all things web development and web design. Learn on your lunch break or wake up and enjoy a cup of coffee with us to learn something new.

Join this developer community, and we can all learn together, build together, and grow together.

Join DevDojo

For more information, please visit https://www.devdojo.com or follow @thedevdojo on Twitter.

This ebook was generated by Ibis developed by Mohamed Said.

Ibis is a PHP tool that helps you write eBooks in markdown.

The cover for this ebook was created with Canva.com.

If you ever need to create a graphic, poster, invitation, logo, presentation – or anything that looks good — give Canva a go.

It is more likely than not that **Docker** and containers are going to be part of your IT career in one way or another.

After reading this eBook, you will have a good understanding of the following:

- What is Docker
- What are containers
- What are Docker Images
- What is Docker Hub
- How to installing Docker
- How to work with Docker containers
- How to work with Docker images
- What is a Dockerfile
- How to deploy a Dockerized app
- Docker networking
- What is Docker Swarm
- How to deploy and manage a Docker Swarm Cluster

I'll be using **DigitalOcean** for all of the demos, so I would strongly encourage you to create a **DigitalOcean** account follow along. You would learn more by doing!

To make things even better you can use my referral link to get a free $100 credit that you could use to deploy your virtual machines and test the guide yourself on a few **DigitalOcean servers**:

[DigitalOcean $100 Free Credit](#)

Once you have your account here's how to deploy your first Droplet/server:

https://www.digitalocean.com/docs/droplets/how-to/create/

I'll be using **Ubuntu 21.04** so I would recommend that you stick to the same so you could follow along.

However you can run Docker on almost any operating system including Linux, Windows, Mac, BSD and etc.

According to the official definition from the docker.com website, a container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

![x]

A **Docker Image** is just a template used to build a running Docker Container, similar to the ISO files and Virtual Machines. The containers are essentially the running instance of an image. Images are used to share a containerized applications. Collections of images are stored in registries like DockerHub or private registries.

DockerHub is the default **Docker image registry** where we can store our **Docker images**. You can think of it as GitHub for Git projects.

Here's a link to the Docker Hub:

https://hub.docker.com

You can sign up for a free account. That way you could push your Docker images from your local machine to DockerHub.

Nowadays you can run Docker on Windows, Mac and of course Linux. I will only be going through the Docker installation for Linux as this is my operating system of choice.

I'll deploy an **Ubuntu server on DigitalOcean** so feel free to go ahead and do the same:

[Create a Droplet DigitalOcean](#)

Once your server is up and running, SSH to the Droplet and follow along!

If you are not sure how to SSH, you can follow the steps here:

https://www.digitalocean.com/docs/droplets/how-to/connect-with-ssh/

The installation is really straight forward, you could just run the following command, it should work on all major **Linux** distros:

```
wget -qO- https://get.docker.com | sh
```

It would do everything that's needed to install **Docker on your Linux machine**.

After that, set up Docker so that you could run it as a non-root user with the following command:

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

To test **Docker** run the following:

```
docker version
```

To get some more information about your Docker Engine, you can run the following command:

```
docker info
```

With the `docker info` command, we can see how many running containers that we've got and some server information.

The output that you would get from the docker version command should look something like this:

In case you would like to install Docker on your Windows PC or on your Mac, you could visit the official Docker documentation here:

https://docs.docker.com/docker-for-windows/install/

And:

https://docs.docker.com/docker-for-mac/install/

That is pretty much it! Now you have Docker running on your machine!

Now we are ready to start working with containers! We will pull a **Docker image** from the **DockerHub**, we will run a container, stop it, destroy it and more!

Once you have your **Ubuntu Droplet** ready, ssh to the server and follow along!

So let's run our first Docker container! To do that you just need to run the following command:

```
docker run hello-world
```

You will get the following output:

```
root@docker:~# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:6540fc08ee6e6b7b63468dc3317e3303aae178cb8a45ed3123180328bcc1d20f
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/

root@docker:~#
```

We just ran a container based on the **hello-world Docker Image**, as we did not have the image locally, docker pulled the image from the [DockerHub](#) and then used that image to run the container. All that happened was: the **container ran**, printed some text on the screen and then exited.

Then to see some information about the running and the stopped containers run:

```
docker ps -a
```

You will see the following information for your **hello-world container** that you just ran:

```
root@docker:~# docker ps -a
CONTAINER ID        IMAGE                   COMMAND
CREATED             STATUS                          PORTS
NAMES
62d360207d08        hello-world             "/hello"                5
minutes ago         Exited (0) 5 minutes ago
focused_cartwright
```

In order to list the locally available Docker images on your host run the following command:

```
docker images
```

Let's run a more useful container like an **Apache** container for example.

First, we can pull the image from the docker hub with the **docker pull command**:

```
docker pull webdevops/php-apache
```

You will see the following output:

```
[root@docker:~# docker pull webdevops/php-apache
Using default tag: latest
latest: Pulling from webdevops/php-apache
35c102085707: Pull complete
251f5509d51d: Pull complete
8e829fe70a46: Pull complete
6001e1789921: Pull complete
258b5eb418c1: Pull complete
631fa9e6fae0: Extracting [======>                                   ]  360.4kB/2.613MB
4b66173e7add: Download complete
f83bf8c8025c: Download complete
8a189afa963d: Download complete
820adbd2a2e9: Downloading [================================================>   ]  76.83MB/81.82MB
a6ed74e58632: Download complete
797a96241110: Download complete
953414d456f2: Download complete
7d4e5514ff1b: Download complete
```

Then we can get the image ID with the docker images command:

```
docker images
```

The output would look like this:

```
root@docker:~#
root@docker:~# docker images
REPOSITORY            TAG          IMAGE ID        CREATED         SIZE
webdevops/php-apache  latest       fd4f7e58ef4b    8 hours ago     531MB
hello-world           latest       fce289e99eb9    7 months ago    1.84kB
root@docker:~# █
```

Note, you do not necessarily need to pull the image, this is just for demo pourpouses. When running the `docker run` command, if the image is not available locally, it will automatically be pulled from Docker Hub.

After that we can use the **docker run** command to spin up a new container:

```
docker run -d -p 80:80 IMAGE_ID
```

Quick rundown of the arguments that I've used:

- `-d`: it specifies that I want to run the container in the background. That way when you close your terminal the container would remain running.

- `-p 80:80`: this means that the traffic from the host on port 80 would be forwarded to the container. That way you could access the Apache instance which is running inside your docker container directly via your browser.

With the docker info command now we can see that we have 1 running container.

And with the `docker ps` command we could see some useful information about the container like the container ID, when the container was started and etc.:

```
root@docker:~# docker ps
CONTAINER ID         IMAGE                    COMMAND
CREATED              STATUS                   PORTS
NAMES
7dd1d512b50e         fd4f7e58ef4b             "/entrypoint supervi…"
About a minute ago   Up About a minute   443/tcp,
0.0.0.0:80->80/tcp, 9000/tcp   pedantic_murdock
```

Then you can stop the running container with the docker stop command followed by the container ID:

```
docker stop CONTAINER_ID
```

If you need to you can start the container again:

```
docker start CONTAINER_ID
```

In order to restart the container you can use the following:

```
docker restart CONTAINER_ID
```

If you need to attach to the container and run some commands inside the container use the `docker exec` command:

```
docker exec -it CONTAINER_ID /bin/bash
```

That way you will get to a **bash shell** in the container and execute some commands inside the container itself.

Then to detach from the interactive shell press CTRL+PQ that way you will not stop the container but just detached from the interactive shell.

```
root@docker:~#
root@docker:~# docker exec -it 7dd1d512b50e /bin/bash
root@7dd1d512b50e:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.3 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.3 LTS"
VERSION_ID="18.04"
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"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
root@7dd1d512b50e:/# read escape sequence
root@docker:~# docker stop 7dd1d512b50e
7dd1d512b50e
root@docker:~#
```

To delete the container run first make sure that the container is not running and then run:

```
docker rm CONTAINER_ID
```

If you would like to delete the container and the image all together, just run:

```
docker rmi IMAGE_ID
```

With that you now know how to pull Docker images from the **Docker Hub**, run, stop, start and even attach to Docker containers!

We are ready to learn how to work with **Docker images!**

A Docker Image is just a template used to build a running Docker Container, similar to the ISO files and Virtual Machines. The containers are essentially the running instance of an image. Images are used to share a containerized applications. Collections of images are stored in registries like DockerHub or private registries.

The `docker run` command downloads and runs images at the same time. But we could also only download images if we wanted to wit the docker pull command. For example:

```
docker pull ubuntu
```

Or if you want to get a specific version you could also do that with:

```
docker pull ubuntu:14.04
```

Then to list all of your images use the docker images command:

```
docker images
```

You would get a similar output to:

```
root@docker:~# docker images
REPOSITORY              TAG            IMAGE ID        CREATED          SIZE
webdevops/php-apache    latest         fd4f7e58ef4b    8 hours ago      531MB
ubuntu                  latest         a2a15febcdf3    2 days ago       64.2MB
ubuntu                  14.04          2c5e00d77a67    3 months ago     188MB
hello-world             latest         fce289e99eb9    7 months ago     1.84kB
root@docker:~#
```

The images are stored locally on your docker host machine.

To take a look a the docker hub, go to: https://hub.docker.com and you would be able to see where the images were just downloaded from.

For example, here's a link to the **Ubuntu image** that we've just downloaded:

https://hub.docker.com/_/ubuntu

There you could find some useful information.

As Ubuntu 14.04 is really outdated, to delete the image use the `docker rmi`

command:

```
docker rmi ubuntu:14.04
```

One of the ways of modifying images is with ad-hoc commands. For example just start your ubuntu container.

```
docker run -d -p 80:80 IMAGE_ID
```

After that to attach to your running container you can run:

```
docker exec -it container_name /bin/bash
```

Install whatever packages needed then exit the container just press CTRL+P+Q.

To then save your changes run the following:

```
docker container commit ID_HERE
```

Then list your images and note your image ID:

```
docker images ls
```

The process would look as follows:

```
root@docker:~# docker images          List our images and get the ID for the php-apache image
REPOSITORY              TAG            IMAGE ID            CREATED             SIZE
webdevops/php-apache    latest         fd4f7e58ef4b        8 hours ago         531MB
ubuntu                  latest         a2a15febcdf3        2 days ago          64.2MB
ubuntu                  14.04          2c5e00d77a67        3 months ago        188MB
hello-world             latest         fce289e99eb9        7 months ago        1.84kB
root@docker:~# docker run -d -p 80:80 fd4f7e58ef4b   Run the container
17e6f50543c576d9fb48e7a29a5cbb243ba14a1a829de545406cdd687b214989
root@docker:~# docker exec -it 17e6f50543c576d /bin/bash   Attach to the container and make some changes
root@17e6f50543c5:/# echo "<h1>Bobby Iliev Introduction to Docker</h1>" > /var/www/html/index.html
root@17e6f50543c5:/# read escape sequence
root@docker:~# docker container commit 17e6f50543c576d   Commit your changes
sha256:9f812ad10cbb055775b9c28994e7b1de40c8b84e44ca5e1fe76f7387086dd00f
root@docker:~# docker images          List our images
REPOSITORY              TAG            IMAGE ID            CREATED             SIZE
<none>                  <none>         9f812ad10cbb        14 seconds ago      532MB
webdevops/php-apache    latest         fd4f7e58ef4b        8 hours ago         531MB
ubuntu                  latest         a2a15febcdf3        2 days ago          64.2MB
ubuntu                  14.04          2c5e00d77a67        3 months ago        188MB
hello-world             latest         fce289e99eb9        7 months ago        1.84kB
```

As you would notice your newly created image would not have a name nor a tag, so in order tag your image run:

```
docker tag IMAGE_ID YOUR_TAG
```

Now if you list your images you would see the following output:

```
root@docker:~# docker tag c05721b2fab6 bobbyiliev/php-apache:latest
root@docker:~# docker images
REPOSITORY              TAG            IMAGE ID            CREATED
bobbyiliev/php-apache   latest         c05721b2fab6        23 seconds ago
webdevops/php-apache    latest         fd4f7e58ef4b        9 hours ago
ubuntu                  latest         a2a15febcdf3        2 days ago
ubuntu                  14.04          2c5e00d77a67        3 months ago
hello-world             latest         fce289e99eb9        7 months ago
root@docker:~#
```

Now that we have our new image locally, let's see how we could push that new image to DockerHub.

For that you would need a Docker Hub account first. Then once you have your account ready, in order to authenticate, run the following command:

```
docker login
```

Then push your image to the **Docker Hub**:

```
docker push your-docker-user/name-of-image-here
```

The output would look like this:

```
Login Succeeded
root@docker:~# docker images
REPOSITORY              TAG            IMAGE ID
bobbyiliev/php-apache   latest         c05721b2fab6
webdevops/php-apache    latest         fd4f7e58ef4b
ubuntu                  latest         a2a15febcdf3
ubuntu                  14.04          2c5e00d77a67
hello-world             latest         fce289e99eb9
root@docker:~# docker push bobbyiliev/php-apache
The push refers to repository [docker.io/bobbyiliev/php-apache]
868169b4b463: Pushed
7932a6d7fb8b: Mounted from webdevops/php-apache
dd8c9ac68361: Mounted from webdevops/php-apache
192e1c6f533d: Mounted from webdevops/php-apache
7366b208287e: Mounted from webdevops/php-apache
d8e58967ec00: Mounted from webdevops/php-apache
0fb9f5e3bca8: Mounted from webdevops/php-apache
4bce5b11e428: Mounted from webdevops/php-apache
7479eb10d8dd: Mounted from webdevops/php-apache
df021786f442: Waiting
7576a954eea3: Waiting
122be11ab4a2: Waiting
7beb13bce073: Waiting
f7eae43028b3: Waiting
6cebf3abed5f: Waiting
```

After that you should be able to see your docker image in your docker hub account, in my case it would be here:

https://cloud.docker.com/repository/docker/bobbyiliev/php-apache

This is a sample from "Introduction to Docker" by Bobby Iliev.

For more information, [Click here](#).