# What are 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 containerized applications. Collections of images are stored in registries like DockerHub or private registries.

## Working with Docker images

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

![](https://cdn.devdojo.com/posts/images/April2020/docker-images-list.png)

The images are stored locally on your docker host machine.

To take a look a the docker hub, go to: [https://hub.docker.com](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](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As Ubuntu 14.04 is really outdated, to delete the image use the `docker rmi` command: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

```

## Modifying images ad-hoc

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 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:

![](https://cdn.devdojo.com/posts/images/April2020/docker-commit.png)

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

```

docker tag IMAGE\_ID YOUR\_TAG

```

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

![](https://cdn.devdojo.com/posts/images/April2020/docker-tag.png)

## Pushing images to Docker Hub

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:

![](https://cdn.devdojo.com/posts/images/April2020/docker-push.png)

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

![](https://cdn.devdojo.com/posts/images/April2020/docker-hub.png)

## Modifying images with Dockerfile

We will go the Dockerfile a bit more in depth in the next blog post, for this demo we will only use a simple Dockerfile just as an example:

Create a file called `Dockerfile` and add the following content:

```

FROM alpine

RUN apk update

```

All that this `Dockerfile` does is to update the base Alpine image.

To build the image run:

```

docker image build -t alpine-updated:v0.1 .

```

Then you could again list your image and push the new image to the \*\*Docker Hub\*\*!

## Docker images Knowledge Check

Once you've read this post, make sure to test your knowledge with this Docker Images Quiz:

[https://quizapi.io/predefined-quizzes/common-docker-images-questions](https://quizapi.io/predefined-quizzes/common-docker-images-questions)

Now that you know how to pull, modify, and push \*\*Docker images\*\*, we are ready to learn more about the `Dockerfile` and how to use it!