Docker Documentation

Alejandro Leon Marin

February 24, 2024

Introduction

In this document we will cover the basics of Docker, a containerization platform that allows you to run applications in a controlled environment. Also it will help us to understand the basic concepts of Docker and how to use it. We'll cover the definition of containers, images, and the basic commands to run docker containers.

What is Docker?

Docker is a versatile platform designed for developing, packaging, and running applications within a controlled environment. This platform utilizes containerization, allowing applications to run efficiently on various systems while minimizing resource usage. Developers widely adopt Docker due to its ability to streamline the installation and execution of applications.

A container, in this context, serves as a packaged unit comprising an application and its dependencies, such as libraries and system tools. It functions as an isolated environment, ensuring that the application runs consistently across different systems with Docker installed. Consider a container as a virtual box containing your application's code, language-specific components (e.g., HTML and Node.js), and even environment variables within a designated directory like ".env."

Containers can be easily transported between systems, facilitating seamless sharing among developers and operations teams. Docker containers are stored in repositories, which are akin to GitHub but specifically for containers. There are two types of repositories:

1. Public repositories: Open to the public, allowing anyone to view the containers they contain, with Docker Hub being a prominent example. 2. Private repositories: Access to these repositories is restricted, and only individuals with authorized access can view the containers stored within.

Docker Hub provides a wide array of containers for databases, web servers, and other tools that developers can readily incorporate into their projects. When you run a container, you are essentially launching an instance of an image. This approach becomes particularly advantageous in collaborative projects, where team members may have different software versions installed. Docker simplifies this by enabling users to run an image, based on a specific Linux distribution, with the necessary dependencies—eliminating the need to install them individually.

In summary, Docker facilitates a more efficient and standardized development and deployment process by encapsulating applications and their dependencies into containers, which can be easily shared and executed across diverse environments.

What is an image?

So an image is a file that contains all the necessary files to run a container. It includes the code, the runtime, the libraries, the environment variables, and the configuration files and the container makes that all of this files run in a controlled environment. So the image is the file that you can share with other developers and with the operations team. So what's a container?

What is a container?

A container are layers trough layers of images. Where the lowest layer is most of the time a linux system distribution. The most used is called Alpine linux, this is because it's a very small distribution and it's very fast to download. So over that layer we will have many layers of images until we get to the layer of our application. So the magic of the containers are that they are very small talking in terms memory space if we compare them with actual virtual machines where the vm's can take up Gigabytes of memory and the container can take MegaBytes of memory. And why virtualization?

Why virtualization?

Docker is a way of virtualization. So let's talk about virtualization. Let's take VM as Virtual Machine. So VM is based on 3 layers

- The hardware: It is where the VM is running.
- The kernel: It is the layer that is between the hardware and the VM.It's the layer that allows the VM to run on the hardware.
- The apps: Not much to explain, this are the applications we use when we work with the VM

So when we talk about VM we virtualise The apps and the kernel. In this case the kernel could be a linux distribution, windows iso or mac os iso. And this makes that the images of the VM's size up to Gigabytes. In docker case we only virtualize the apps. So with the kernel docker uses the host kernel. So if you are executing a container in a linux host, the container will use the linux kernel of the host. Same happen with windows and MacOs.

Images commands

Let's start with the basic commands of docker. First to run the commands you need to run docker desktop running. Also to check if you have docker installed you can run the command

```
docker --version
```

Also docker recommend to run the hello-world image to check if the installation is correct. To do that you can run the command

```
docker run hello-world
```

This command will download the hello-world image and run it. If everything is correct you will see a message that says that the installation is correct.

Now our first command will be to list all the images that we have in our local machine. To do that we can run the command

```
docker images
```

Now to download our image we can run the command

```
docker pull <image-name>
```

In this case ¡image-name¿ could python for example. This will download the python image from the docker hub or Node, or mysql, or Postgres, etc. Another thing to have in consideration is that you can pull a image with a specific version. For example if you want to pull the python 3.8.5 you can run the command

```
docker pull python:3.8.5
```

But if you use the command docker pull python you will download the latest version of python.

And in the process of the pull you will see graphically how the layers of the image are being downloaded and this will help to understand that concepts.

Now in the list of images you will see the image that you just downloaded. This list have the name of repository where the image is, the tag, the image id, the created date and the size of the image. To know what images you can pull you can go to the docker hub and search for the image that you want to pull.

If you have a problem with the image that you just downloaded you can try fixing it with this command

```
docker pull --platform linux/x86_64 <image-name>
```

Now we see the command we need to run to delete the images that we don't need anymore. To do that we can run the command

```
docker image rm <image-id>
```

or

```
docker image rm <image-name>:<tag>
```

Containers commands

First, to mount a container we need a image. So you can download the image you want. Then to create a container you can run the command

```
docker create <image-name>
```

and this will return a container id, save it because you will need it to run the container.

Now to run the container that you create run the command

```
docker start <container-id>
```

and this will return the container id again. And how do i do to check if the container is running? You can run the command

```
docker ps
```

now this will return a table with valuable information. The first column is the container id, the second column is the image name, the third column is the command that the container is running, the fourth column is the creation date, the fifth column is the status of the container, the sixth column is the ports that the container is using and the last column is the name of the container.

Now to stop the container you can run the command

```
docker stop <container-id>
```

this will return the container id again and you can check if the container is stopped with the command

```
docker ps
```

To show all the containers that you have in your local machine you can run the command

```
docker ps -a
```

to delete a container you can run the command

```
docker rm <container-id>
```

Now if you want to assign a name to the container you can run the command

```
docker run --name <container-name> <image-name>
```

and with this you can run, stop and remove the container by the name that you assigned. You can do the 3 actions with the same commands that we saw before just thata change de ¡container-id¿ for the ¡container-name¿

And now we can't use this container because we don't have any port open.So we need to indicate to docker that we want to open a port but this port we need to map it to a port of our container. Because we can access to the container's port. But first let's explain what is Port Mapping

Port Mapping

The concept of port mapping is essential to understand when working with Docker. Basically is the process of forwarding network traffic from one network port to another. Imagine that we have and application that is running in the port 3000 and we have a container that is running in the port 27017. We can map the port 3000 of our local machine to the port 27017 of the container. So when we access to the port 3000 of our local machine we are accessing to the port 27017 and now we implemented the port mapping.

So to map a port you can run the command

```
docker run -p<local-port>:<container-port> <image-name>
```

we can also map the port to a specific ip address. To do that you can run the command

```
docker run -p<ip-address>:<local-port>:<container-port> <image-name>
```

also we can let docker decide the port that we are going to use. To do that you can run the command

```
docker run -p<container-port> <image-name>
```

And now we need to now if the container is being executed in the right way. To do that you can run the command

```
docker logs <container-id>
```

ando to keep tracking the logs we can run the command

```
docker logs --follow <container-id>
```

and with this command we can see the logs in real time. To exit of those logs you can press ctrl + c.

Now we are going to learn a new comman, that will be a combination of pull, create and start. This command is

```
docker run <image-name>
```

This command will pull the image if it's not in your local machine, then it will create the container and then it will start the container. And this command will show the logs of the container in real time and you can detained with the command ctrl + c but this will detained all the container. But if you want to run the container in the background you can run the command

```
docker run -d <image-name>
```

and this will return the container id. Also we can merge all the commands that we saw before in one command.For example

```
docker run -d -p<local-port>:<container-port> --name <container-name> <im
```

So as you can see we can merge all the commands in one command. And this will make the process of creating and running a container It's important to know that every time you run a container you are creating a new container.

Conecting to a container

And now we are going to learn how to connect to a container.

Docker Compose

Volumes

Environments and Hot Reload

Conclusion