**Virtual machines (VM)**

Vms are basically machines inside real machines. We realize it with the hypervisor. Hypervisor is just an application.

Virtualization is the process of creating a software-based, or "virtual" version of a computer, with dedicated amounts of CPU, memory, and storage that are "borrowed" from a physical host computer—such as your personal computer— and/or a remote server—such as a server in a cloud provider's datacenter. A virtual machine is a computer file, typically called an image, that behaves like an actual computer. It can run in a window as a separate computing environment, often to run a different operating system—or even to function as the user's entire computer experience—as is common on many people's work computers. The virtual machine is partitioned from the rest of the system, meaning that the software inside a VM can't interfere with the host computer's primary operating system.

Hypervisor virtualizes hardware components (cpu,ram,storage…) from the host machine.

A hypervisor, also known as a virtual machine monitor or VMM, is software that creates and runs virtual machines (VMs). A hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing.

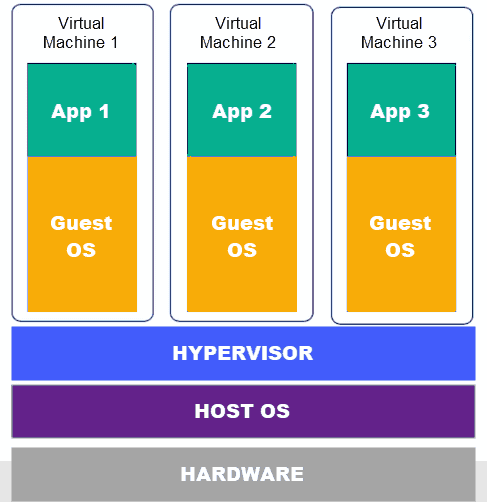
This type of hypervisor is called the type 2 hypervisor. Host os allows guest os’s to use his resources.

So the type 2 hypervisor asks the host’s os for the resources whereas the type 1 hypervisor has direct full control to the host’s hardware.

In type 2 hypervisor the os of the host shares its resources for vms meaning that if there are many vms it has to share its resources for all of them.

We install ISO files of the os that we want. An ISO file (often called an ISO image), is an archive file that contains an identical copy (or image) of data found on an optical disc, like a CD or DVD (so when we install an os we use cds here it is an image of the disk). They are often used for backing up optical discs, or for distributing large file sets that are intended to burned to an optical disc.

ISO image as a complete copy of everything stored on a physical optical disc like CD, DVD, or Blu-ray disc—including the file system itself. So we install iso images to have vms. Kali linux iso and etc.



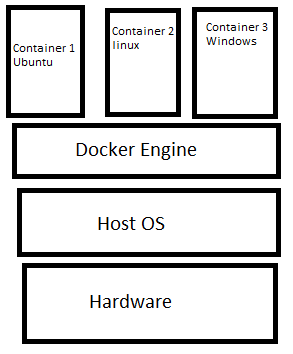
**Docker**

Docker replaces vms. Vms virtualize hardware whereas docker virtualizes os. In docker, you have a single OS (or kernel to be precise), and the resources are shared between the containers. Hence it is lightweight and boots in seconds.

In docker, all of the containers share the same os but it is appearing for each container as if they have their own os.

A container is a way to package application with all the necessary dependencies and configuration and this package is portable, meaning that it can run on any platform. Containers share the same kernel.

Containers live in a special repository called container repository.



Containers are made up of layers. We have linux image layer, application image layer and so on. Images are just binary numbers representing the exact copy of the software.

**Container vs Image**

Container is a running environment for an image. An image is a template for creating an environment of our choice. This could be a database, a web app and etc. An image is a snapshot. You can create multiple snapshots in versions then you can point to a version that you want at a particular time.

So we have an image and from this image we run a container. A container only lives as long as there is a process running inside.

**Pulling an image and running container**

Let’s pull an image from the docker hub and create a container from this image.

After downloading an image, we run a container from the image we pulled. 🡺 Docker run nginx:latest

Latest is the name of the tag. To look at the running containers 🡺 docker container ls or docker ps

To run a container in a detached mode so that we don’t just hang in the command line🡺

Docker run –d nginx:latest

To stop a contaner 🡪 docker stop containerId or containerName

To remove a container 🡺 docker rm containerId or containerName

To remove all the container from a single command🡺

docker rm –f $(docker ps -aq) – here –f is for forcing removal so that running containers are also removed. –a is to display all containers (running and stopped containers) and –q is for only displaying ids.

**Exposing Port**

We wanna go from the host to the container (we wanna access the container from our browser). In this case we have to expose the port. We use –p 🡺 docker run –d –p 8080:80 nginx:latest

So whenever we type “localhost:8080” we want that to be mapped to port 80 on the container.

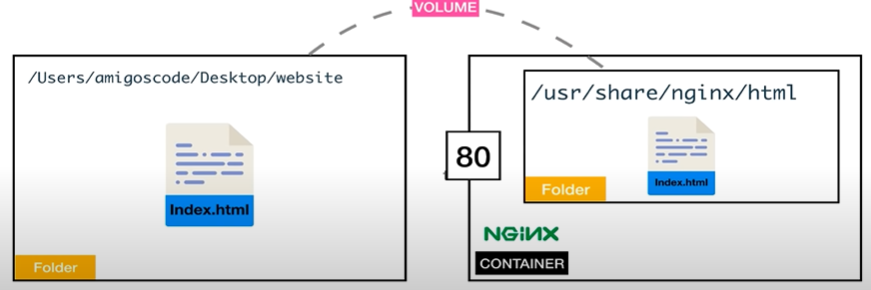
**Container names**

If we don’t specify a container name when we run it, a random name is given. If we want to have a custom container name 🡺

docker run –name containerName –d –p 8080:80 nginx:latest

**Docker Volumes**

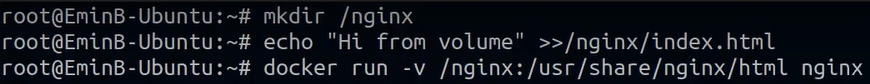
When we have data in a container, once we stop the container the data is gone. So in order to have some persistent data we can take it from the host file system. Volumes allow sharing of data between host and container and also between containers. Files & Folders. A folder in physical host file system is mounted into the virtual file system or docker.



The folder is in the host.



ro- read-only so that it can’t be changed.



We create a ngnix dir and we write “Hi from volume” string with echo and then we mount that file (index.html) into the container.

**Creating custom images**

To create a custom image, we create a Dockerfile. We never build an image from scratch. We always use an existing image as our base image.

From nginx:latest

RUN apt-get update

RUN apt-get install nginx -y

First our base image is nginx. Then update and then install nginx.

**Interactive mode (-it)**

We can get into a running container 🡺 Docker exec –it containerId bin/sh – docker execute in an interactive mode into this container that is running and when you get there execute /bin/sh. If it is alpine then bin/sh for the terminal.

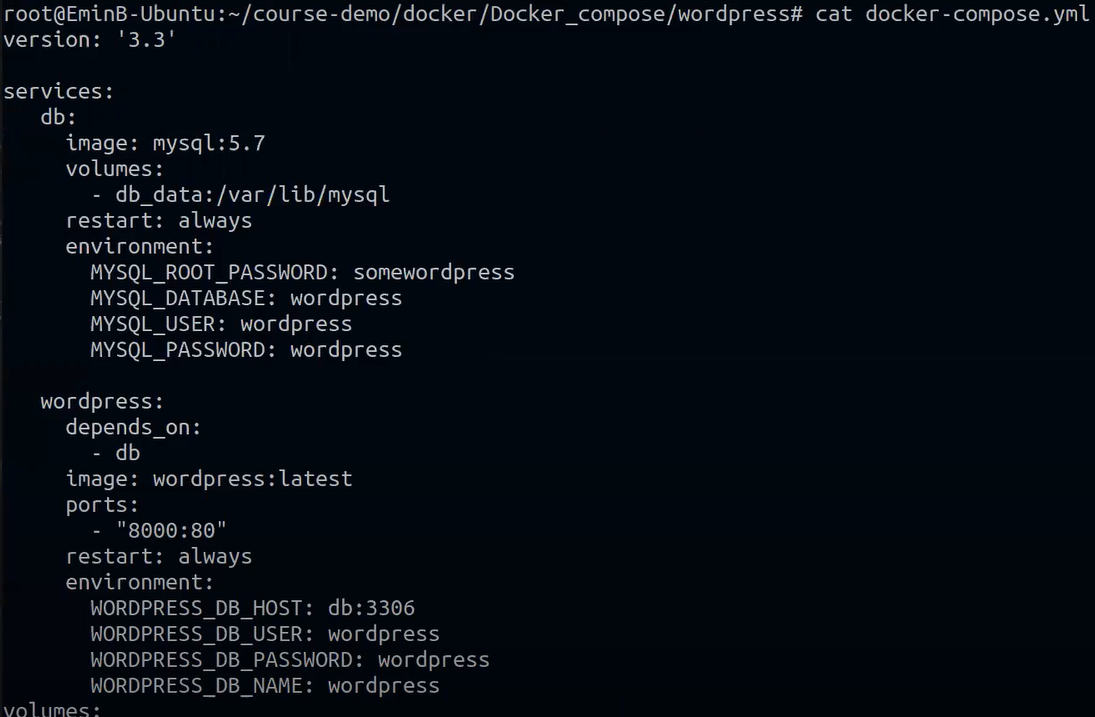
We use this so that we get inside a container and we can run commands inside this container.

**Docker start vs docker run**

Docker run creates a new container but “docker start” restarts already stopped container.

**Docker compose**

Docker compose is just a structured way to contain common docker commands. We just don’t want to have all the commands in one line. It makes editing easier, for example, if we want to change an enviromental variable or if one app requires multiple containers🡺



Services are basically our containers which have their own image.