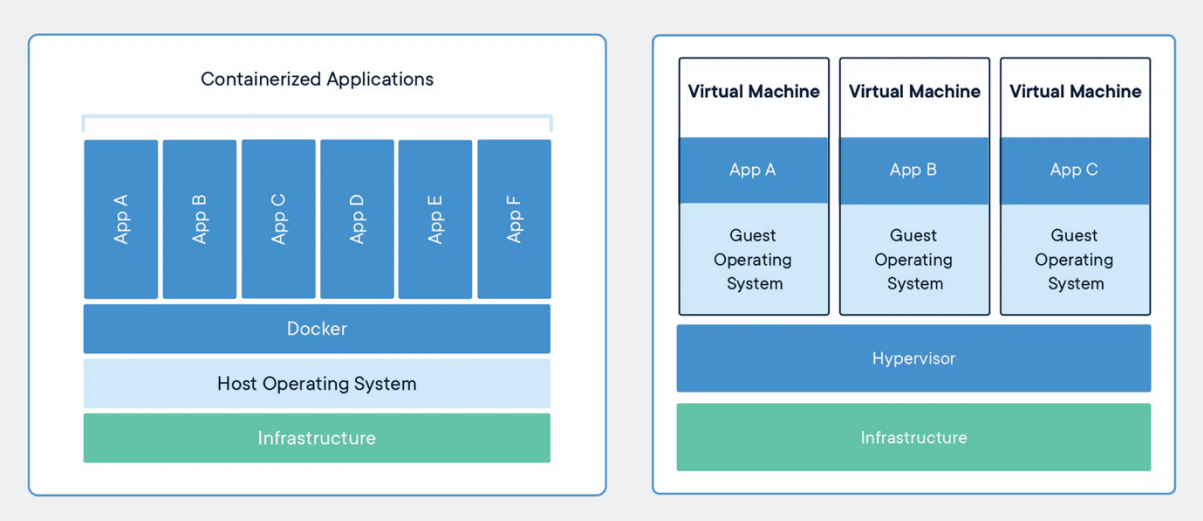
DOCKER

What is a container ?

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.



Containers vs Virtual Machine

Containers and virtual machines are both technologies used to isolate applications and their dependencies, but they have some key differences:

1. Resource Utilization: Containers share the host operating system kernel, making them lighter and faster than VMs. VMs have a full-fledged OS and hypervisor, making them more resource-intensive.

2. Portability: Containers are designed to be portable and can run on any system with a compatible host operating system. VMs are less portable as they need a compatible hypervisor to run.

3. Security: VMs provide a higher level of security as each VM has its own operating system and can be isolated from the host and other VMs. Containers provide less isolation, as they share the host operating system.

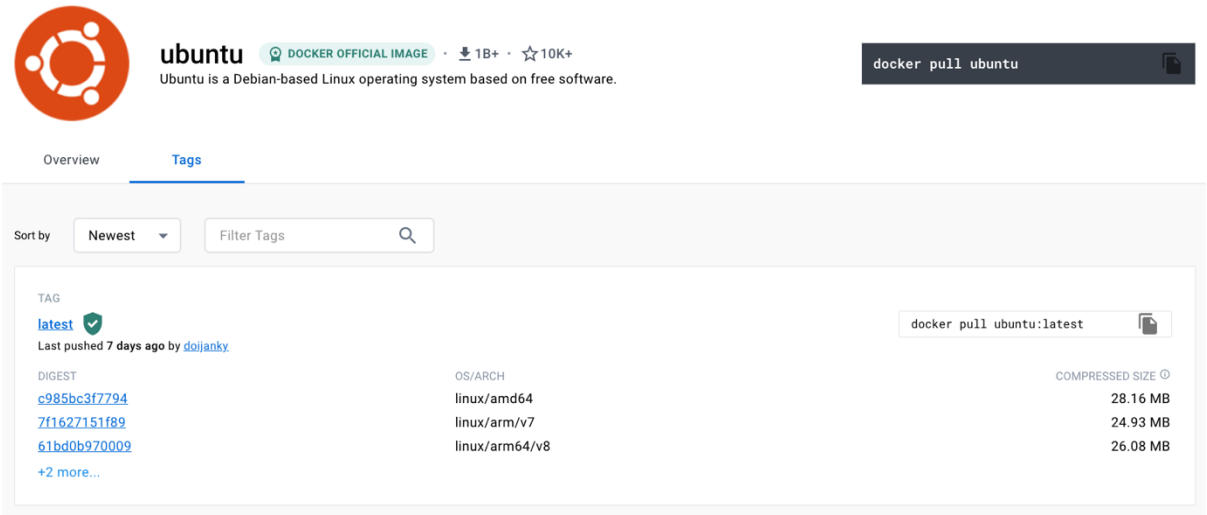
4. Management: Managing containers is typically easier than managing VMs, as containers are designed to be lightweight and fast-moving.

Why are containers light weight ?

Containers are lightweight because they use a technology called containerization, which allows them to share the host operating system's kernel and libraries, while still providing isolation for the application and its dependencies. This results in a smaller footprint compared to traditional virtual machines, as the containers do not need to include a full operating system. Additionally, Docker containers are designed to be minimal, only including what is necessary for the application to run, further reducing their size.

Let's try to understand this with an example:

Below is the screenshot of official ubuntu base image which you can use for your container. It's just ~ 22 MB, isn't it very small ? on a contrary if you look at official ubuntu VM image it will be close to ~ 2.3 GB. So the container base image is almost 100 times less than VM image.



To provide a better picture of files and folders that containers base images have and files and folders that containers use from host operating system (not 100 percent accurate -> varies from base image to base image). Refer below.

Files and Folders in containers base images

/bin: contains binary executable files, such as the ls, cp, and ps commands.

/sbin: contains system binary executable files, such as the init and shutdown commands.

/etc: contains configuration files for various system services.

/lib: contains library files that are used by the binary executables.

/usr: contains user-related files and utilities, such as applications, libraries, and documentation.

/var: contains variable data, such as log files, spool files, and temporary files.

/root: is the home directory of the root user.

Files and Folders that containers use from host operating system

The host's file system: Docker containers can access the host file system using bind mounts, which allow the container to read and write files in the host file system.

Networking stack: The host's networking stack is used to provide network connectivity to the container. Docker containers can be connected to the host's network directly or through a virtual network.

System calls: The host's kernel handles system calls from the container, which is how the container accesses the host's resources, such as CPU, memory, and I/O.

Namespaces: Docker containers use Linux namespaces to create isolated environments for the container's processes. Namespaces provide isolation for resources such as the file system, process ID, and network.

Control groups (cgroups): Docker containers use cgroups to limit and control the amount of resources, such as CPU, memory, and I/O, that a container can access.

It's important to note that while a container uses resources from the host operating system, it is still isolated from the host and other containers, so changes to the container do not affect the host or other containers.

Note: There are multiple ways to reduce your VM image size as well, but I am just talking about the default for easy comparison and understanding.

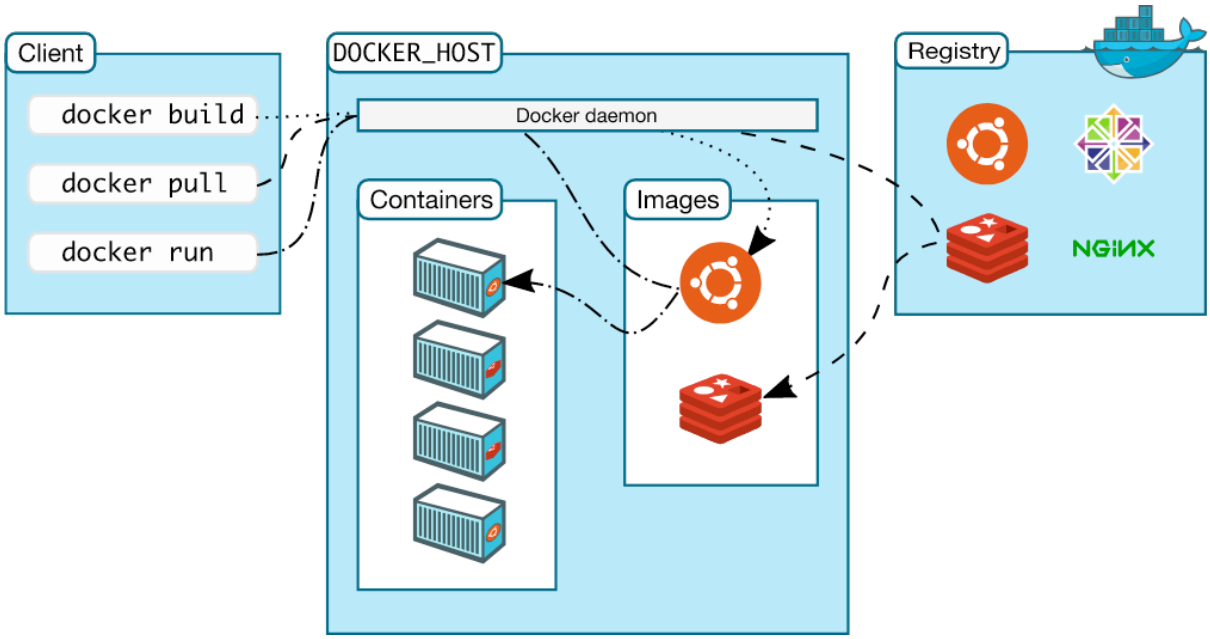
so, in a nutshell, container base images are typically smaller compared to VM images because they are designed to be minimalist and only contain the necessary components for running a specific application or service. VMs, on the other hand, emulate an entire operating system, including all its libraries, utilities, and system files, resulting in a much larger size.

What is Docker ?

Docker is a containerization platform that provides easy way to containerize your applications, which means, using Docker you can build container images, run the images to create containers and also push these containers to container regestries such as DockerHub, Quay.io and so on.

In simple words, you can understand as containerization is a concept or technology and Docker Implements Containerization.

Docker Architecture ?



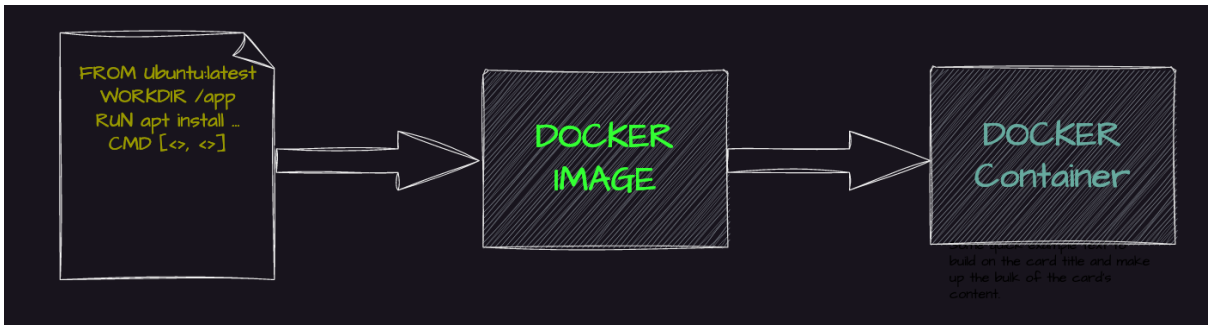
The above picture, clearly indicates that Docker Deamon is brain of Docker. If Docker Deamon is killed, stops working for some reasons, Docker is brain dead :p (sarcasm intended).

Docker Life Cycle

We can use the above Image as reference to understand the lifecycle of Docker.

There are three important things,

1. docker build -> builds docker images from Dockerfile
2. docker run -> runs container from docker images
3. docker push -> push the container image to public/private regestries to share the docker images.



Understanding the terminology (Inspired from Docker Docs)

Docker daemon

The Docker daemon (docker) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to docker, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Desktop

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (docker), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry. Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Docker file

Docker file is a file where you provide the steps to build your Docker Image.

Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Docker file with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Docker file creates a layer in the image. When you change the Docker file and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.