Docker

1) History & Motivation
2) Tech overview
   ① Containers
   ② Docker
3) Installation /set up & hello world
4) Using third party containers
5) Demo Application
6) Building Container Images
   (i) Dockerfile Basics
   (ii) " Optimization
   (iii) Buildx + multiarchitecture

7) Container Registries
8) Running Containers
9) Container Security
10) Interacting with docker Objects
11) Development Workflow
12) Deploying Containers

I) Motivation for Containers

developer $\xrightarrow[\text{docker}]{}$ remote (user)

↳ makes the user's system as close or favourable to run the product

Docker Container - It's a stanalone, executable package of software that includes everything needed to run an application

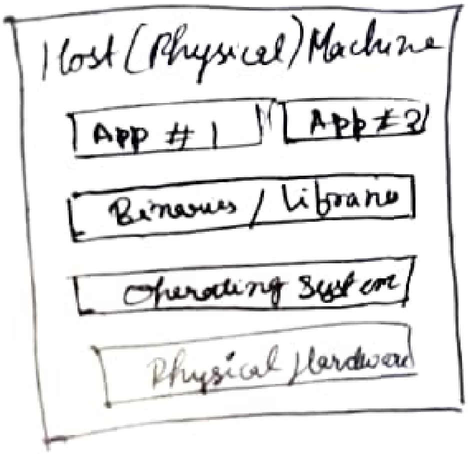
OOP

Container image (class) where as the container we run is an instance

(OCI) Open Container Initative
- Run time Specification
- Image Specification
- Distribution Specification

T1 **Evolution of Virtualization** [Bare Metal] (Most Isolation)

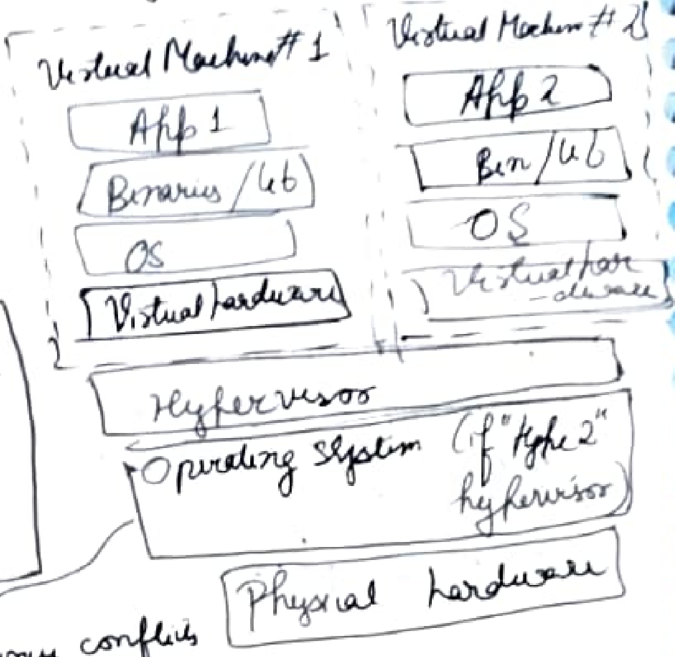
- Hellish dependency conflict
- Low utilization inefficiency



- Large Blast radius
- Slow start up & shutdown speed
- Very slow provisioning & decommissioning (hours to days)

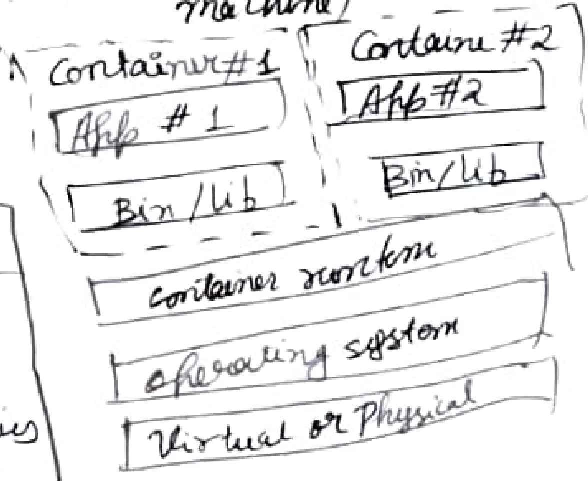
T2 Virtual Machines (Most Isolated)

Host (Physical) Machine



- No dependency conflicts
- Better utilization
- Small blast radius
- Faster startup & shutdown
- Fast provisioning & decommissioning (minutes)

T3 Host (Virtual of Physical machine)



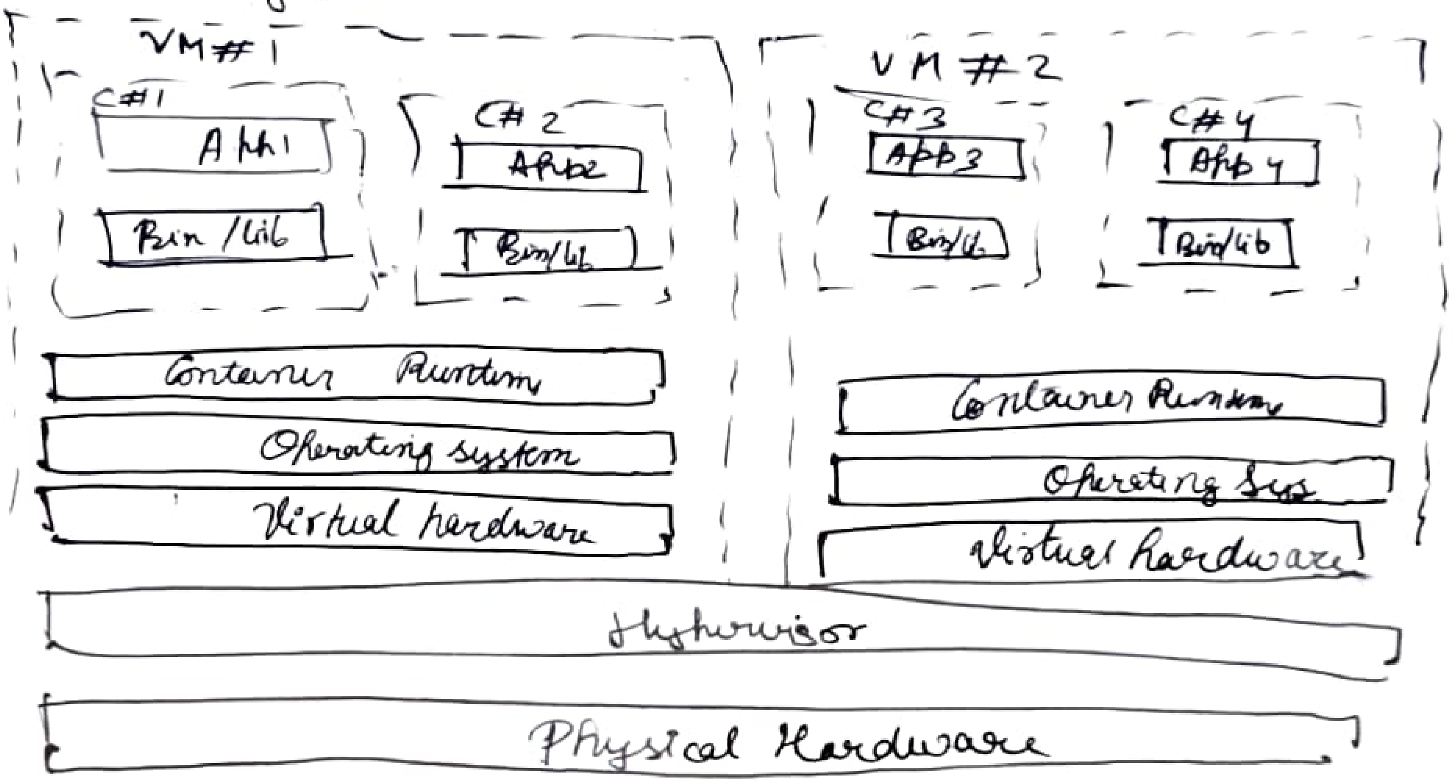
Note ⇒ Shares Linux Kernel

- No dependency conflicts
- Even Better utilization
- Small blast
- Even faster startup & shutdown
- Faster provisioning
- Lightweight enough to use in dev.

Hybrid

VM + Containers + Orchestrator!

Host (Physical) Machine



x → red
– ⇒ yellow
✓ → green

	Bare Metal	Virtual Machine	Containers
Dependency	✗	✓	✓
Utilization	✗	–	✓
Isolation	✓	✓	–
start up	✗	–	✓
Dev/Prod Parity	✗	–	✓
Control	✓	–	–
Performance			
Operational overhead	✗	–	✓

II Tech Overview

Namespaces

A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the name space that they have their own isolated instance of global resource.

Changes to the global resource are visible to the other members of namespace, but are invisible to others

Control Groups

(Kernel Feature)

Limit & monitor access of certain resources

Eg App A (cpu shares)

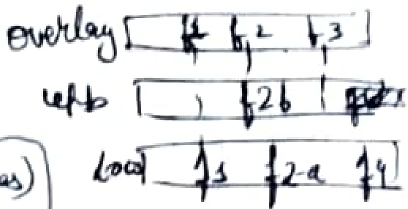
- Use upto 30% cpu cycles
- Use upto 50MB Memory (memory limit in bytes)
- Throttle reads to 5MB/s (blkio, throttle read bps device)
- .
- .

App B

- Use upto 40% of cpu cycles
- Use upto 100MB Memory
- Throttle reads to 10 MB/s (blkio, throttle read bps device)

{Avoid Noise Resource Neighbour problem}

Union FileSystem



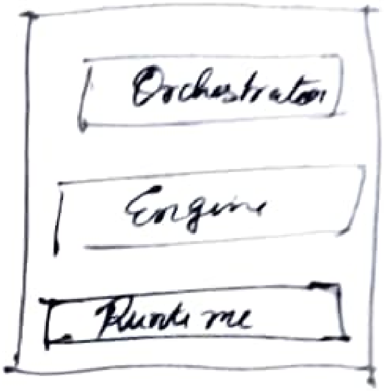
# Many can share same lower level

Part-3

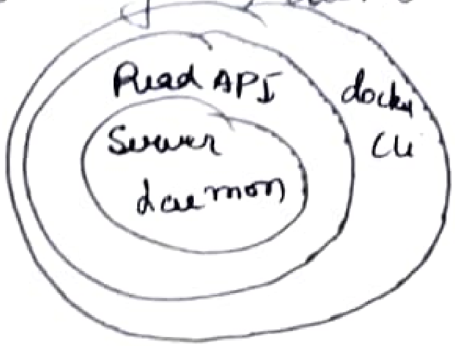
① Runtime ↳ start stop container

→ runc

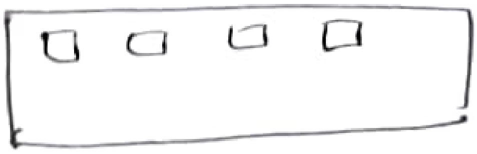
→ containerd



② Engine, Daemon



③ Orchestrator → manages containers



Note

* Image => is the file which all the instruction & everything

Container => Running Instance of Image

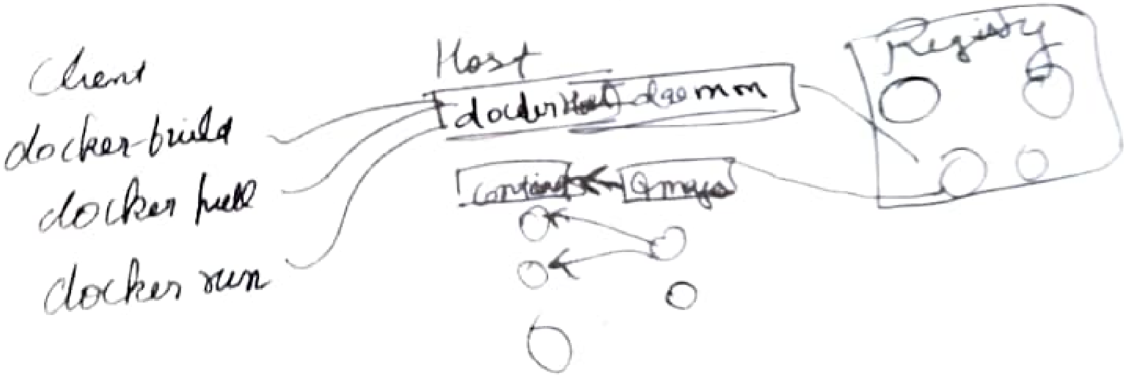
DockerFile → Image → Container

↳ Class ↳ Object

used to create class

List of instruction

Docker Components


Client
docker build
docker pull
docker run
Host
docker ~~Host~~ daemon
Container
Images
Registry