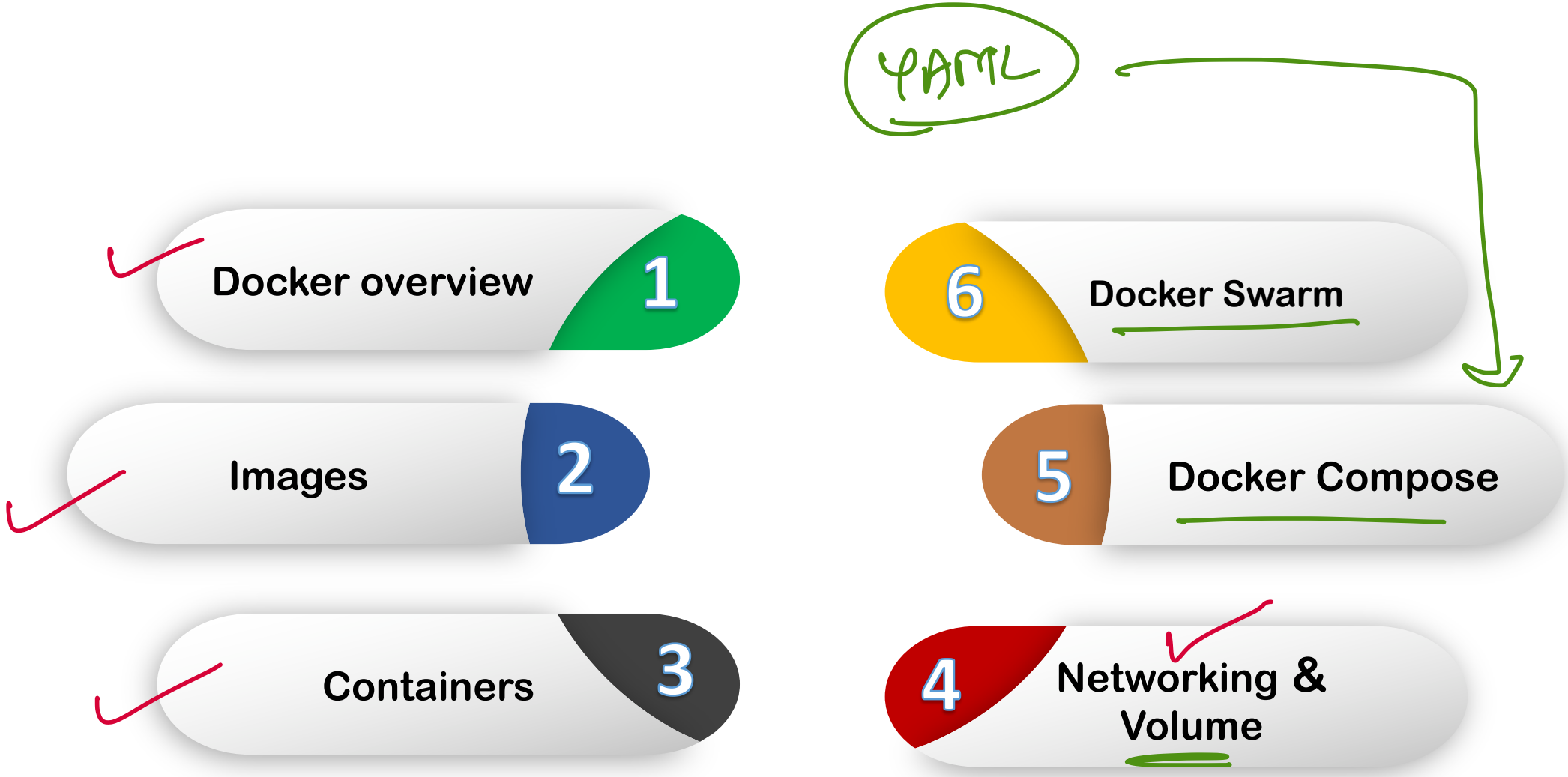
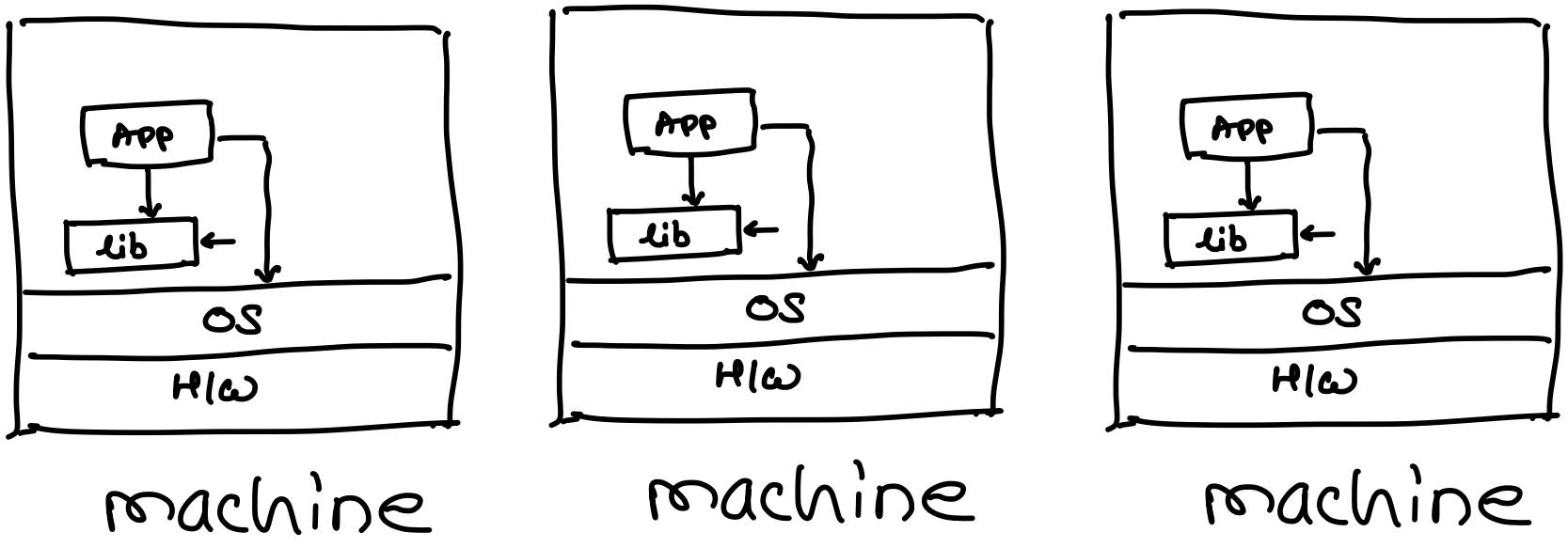

docker


YAML
Docker overview
1
Images
2
Containers
3
4
Networking &
Volume
5
Docker Compose
6
Docker Swarm

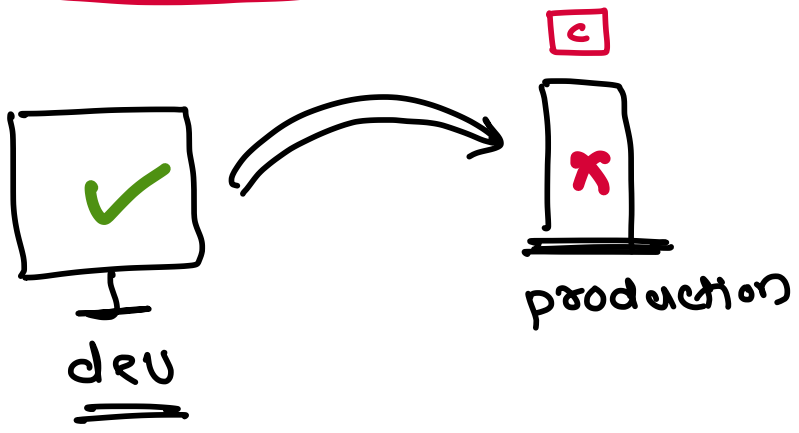
What is a docker ?

- Docker is containerization platform that enables developer to build, test and deploy the application easily and reliably
- Docker host runs multiple containers maintaining the isolation between the containers



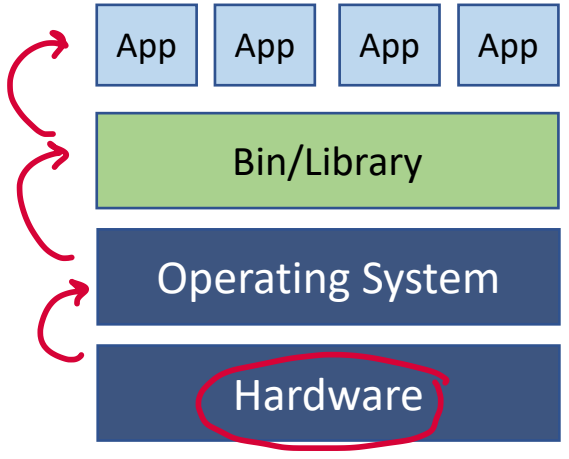
Container

- Allows developers to create and deploy applications faster and more securely
- A container is a standard unit which provides single point of service → application + libraries
- Involves encapsulating or packaging up software code and all its dependencies so that it can run uniformly and consistently on any infrastructure
- It is a operating system virtualization



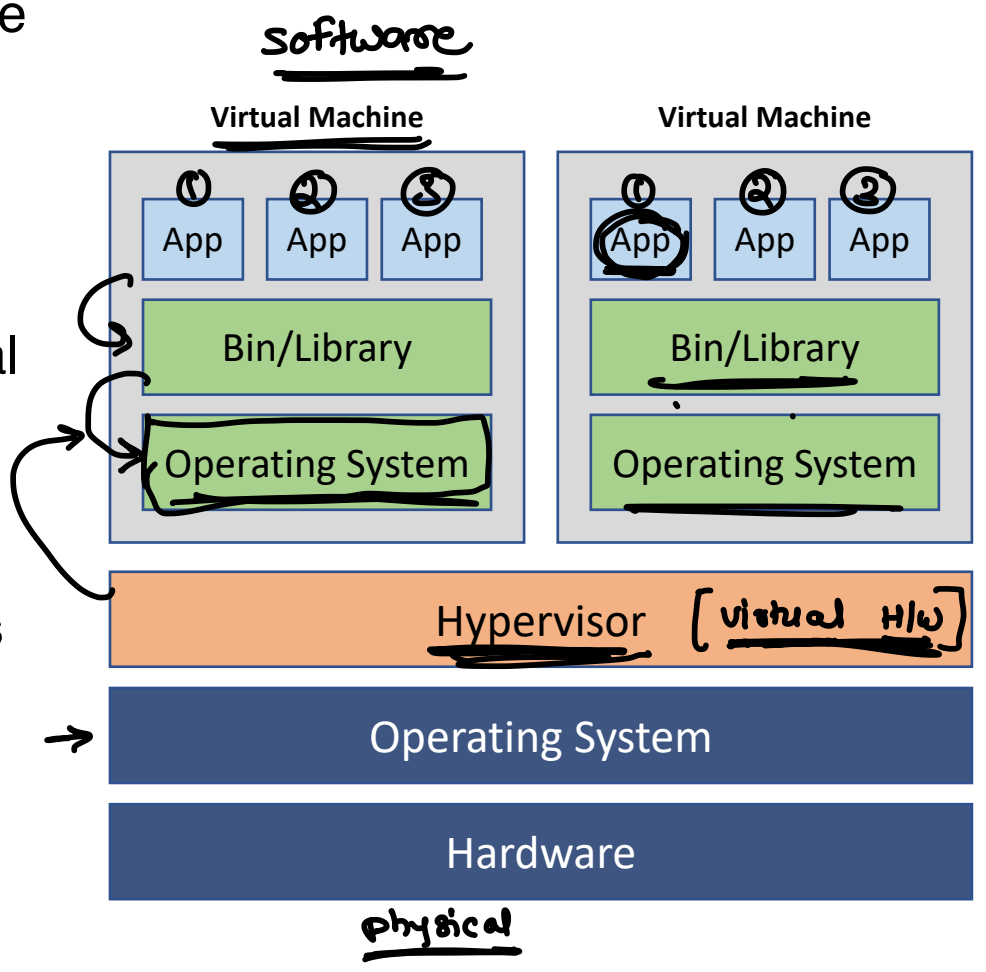
Traditional Deployment

- Early on, organizations ran applications on physical servers
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues
- For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform
- A solution for this would be to run each application on a different physical server
- But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers



Virtualized Deployment

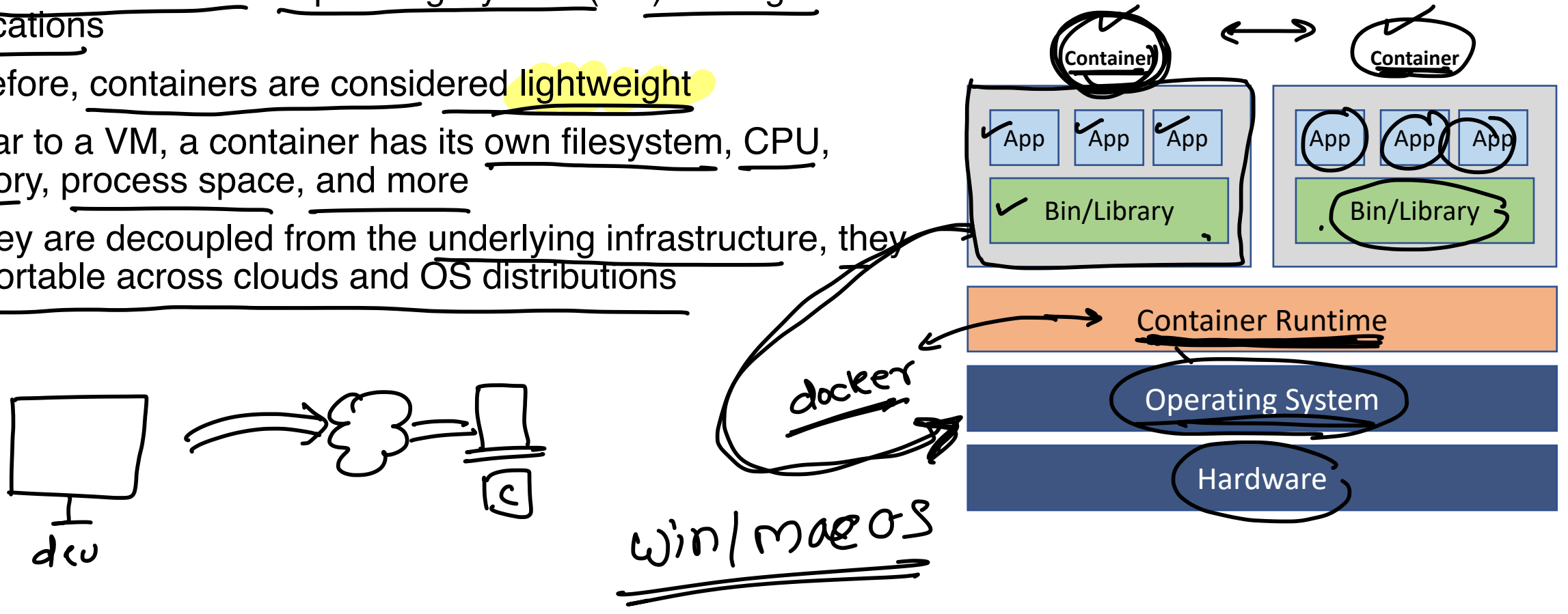
- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU
- Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application
- Virtualization allows better utilization of resources in a physical server and allows better scalability because
  - an application can be added or updated easily
  - reduces hardware costs
- With virtualization you can present a set of physical resources as a cluster of disposable virtual machines
- Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware



Container deployment

Linux

- Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications
- Therefore, containers are considered lightweight
- Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more
- As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions

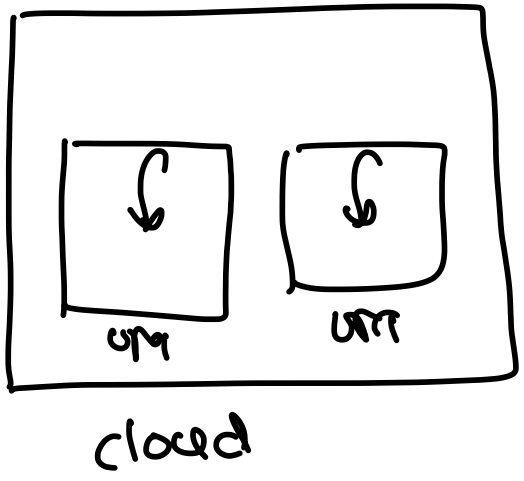


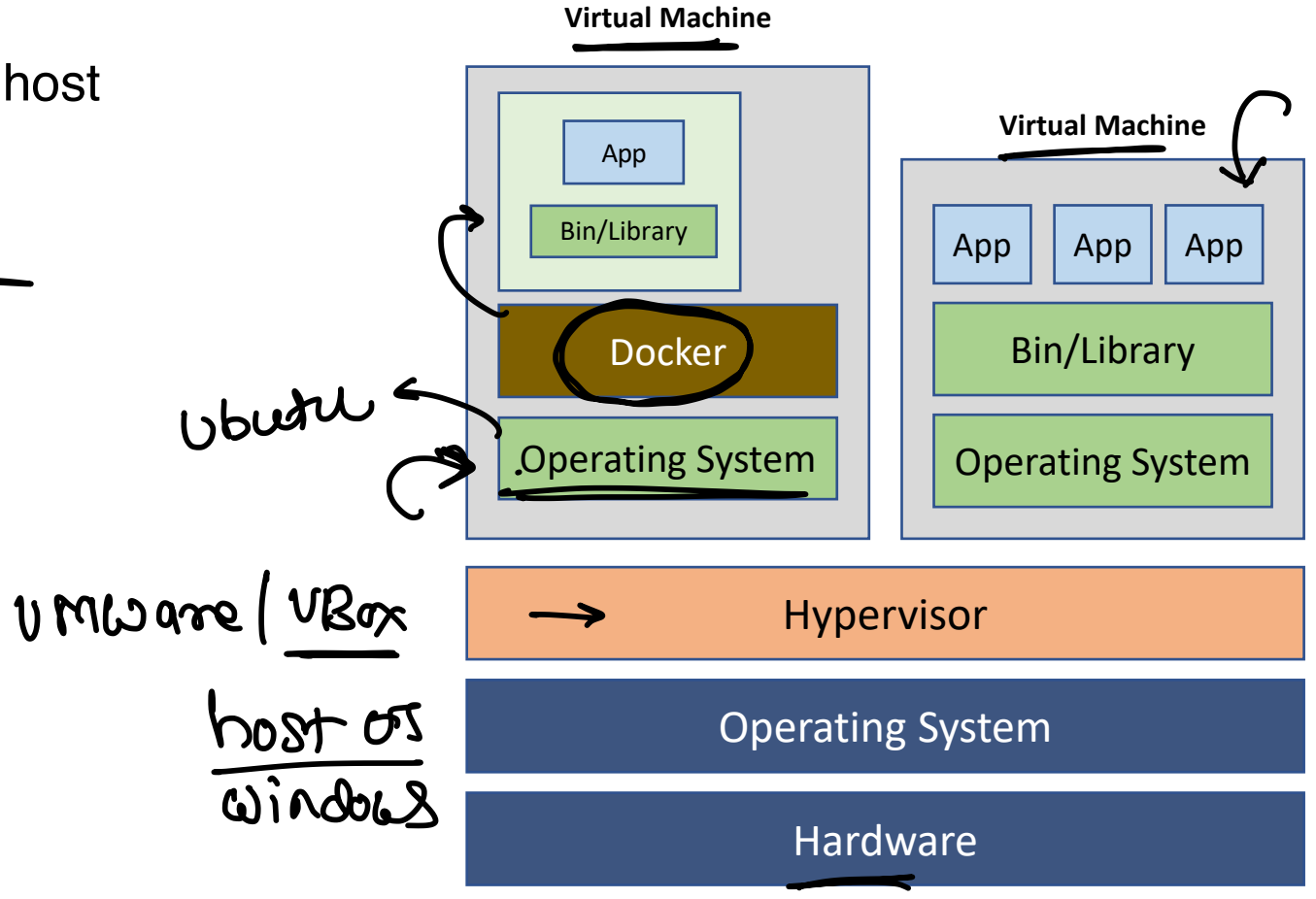
Containerization vs Virtualization

Virtual Machine	Container
Hardware level virtualization	OS virtualization
Heavyweight (bigger in size)	Lightweight (smaller in size)
Slow provisioning	Real-time and fast provisioning
Limited Performance	Native performance
Fully isolated	Process-level isolation
More secure	Less secure
Each VM has separate OS	Each container can share OS resources
Boots in minutes	Boots in seconds
Pre-configured VMs are difficult to find and manage	Pre-built containers are readily available
Can be easily moved to new OS	Containers are destroyed and recreated
Creating VM takes longer time	Containers can be created in seconds

Containerization and Virtualization

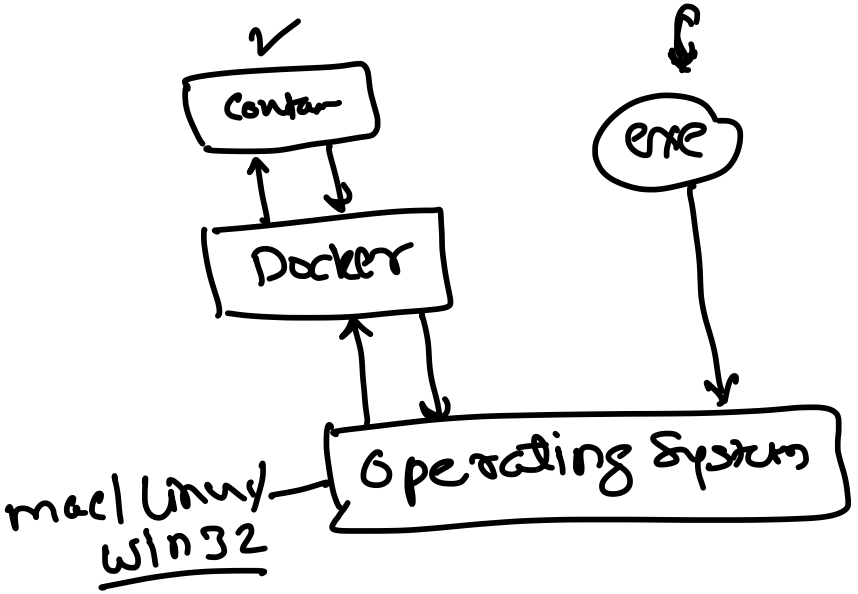
- Containers can run inside virtual machines
- In which case, the a physical machine can host VM that may house Docker containers
- This is preferred in the cloud environment





Why Docker?

- It is an easy way to create application deployable packages
- Developer can create ready-to-run containerized applications
- It provides consistent computing environment
- It works equally well in on-prem as well as cloud environments
- It is light weight compared to VM



Little history about Docker

- Docker Inc, started by Solomon Hykes, is behind the docker tool
- Docker Inc started as PasS provider called as dotCloud
- In 2013, the dotCloud became Docker Inc
- Docker Inc was using LinuX Containers (LXC) before version 0.9
- After 0.9 (2014), Docker replaced LXC with its own library libcontainer which is developed in Go programming language
- Its not the only solution for containerization
  - "FreeBSD Jails", launched in 2000
  - LXD is next generation system container manager build on top of LXC and REST APIs
  - Google has its own open source container technology lmctfy (Let Me Contain That For You)
  - Rkt is another option for running containers

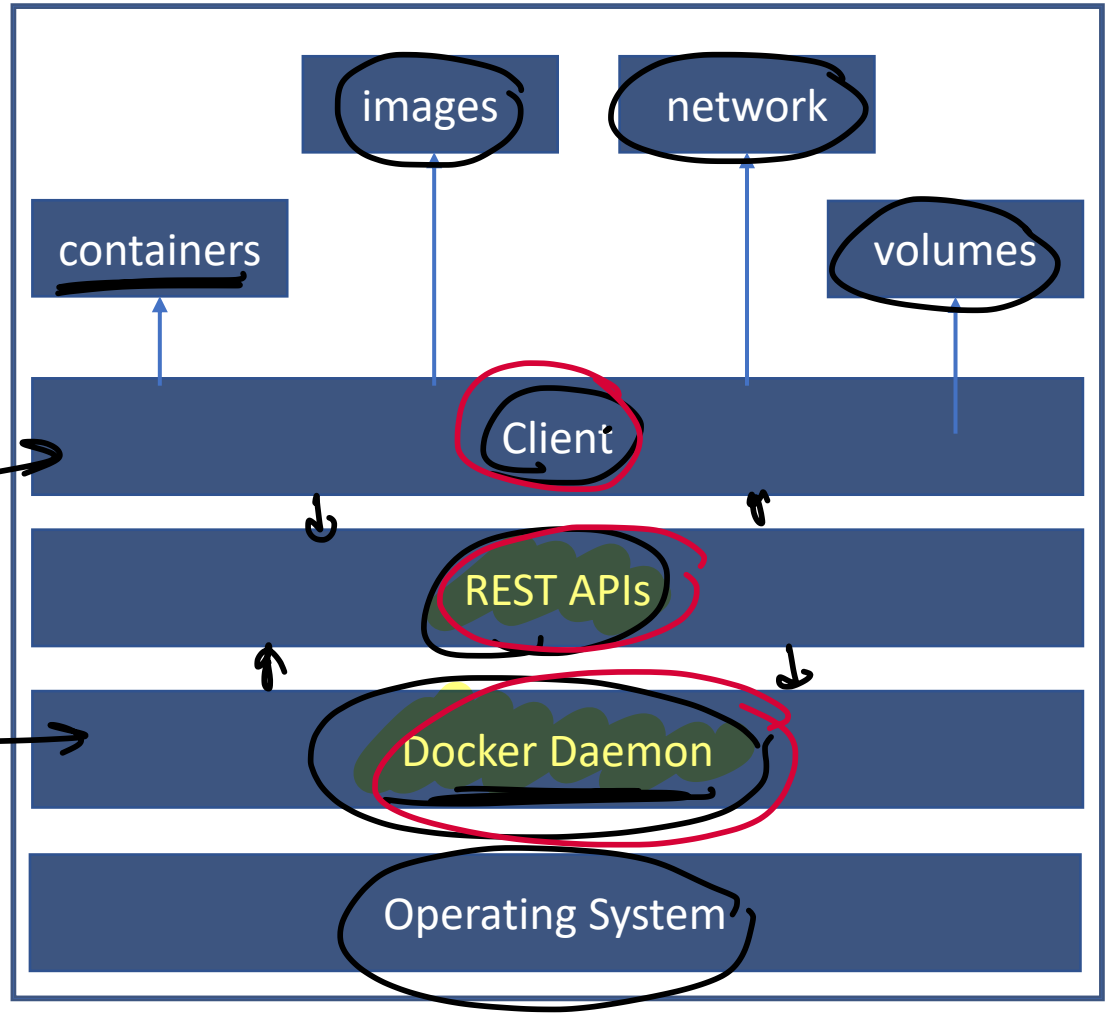
Docker Architecture

- Docker daemon (dockerd)
  - Continuous running process
  - Manages the containers
- REST APIs
  - Used to communicate with docker daemon
- Client (docker)
  - Provides command line interface
  - Used to perform all the tasks

(server)

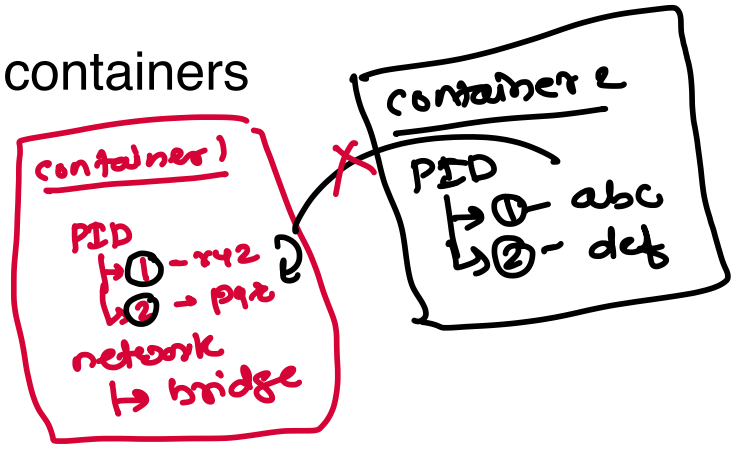
docker

sudo systemctl start docker



libcontainer [Go]

- Docker has replaced LXC by libcontainer, which is used to manage the containers
- Libcontainer uses
  - Namespaces
    - Creates isolated workspace which limits what container can see
    - Provides a layer of isolation to the container
    - Each container runs in a separate namespace
    - Processes running in a namespace can interact with other processes or use resources which are the part of the same namespace
    - E.g. process ID, network, IPC, Filesystem
  - Control Groups (cgroups)
    - Used to share the available resources to the conainers
    - It optionally enforces limits and constraints on resource usage
    - It limits how much a container can use
    - E.g. CPU, Disk space, memory



1 GB | 4 GB
50 MB | 512 GB

libcontainer

- Union File System (UnionFS)
  - It uses layers
  - It is a lightweight and very fast FS
  - Docker uses different variants of UnionFS
    - Aufs (advanced multi-layered unification filesystem)
    - BtrFS (B-Tree FS)
    - VFS (Virtual FS)
    - Devicemapper

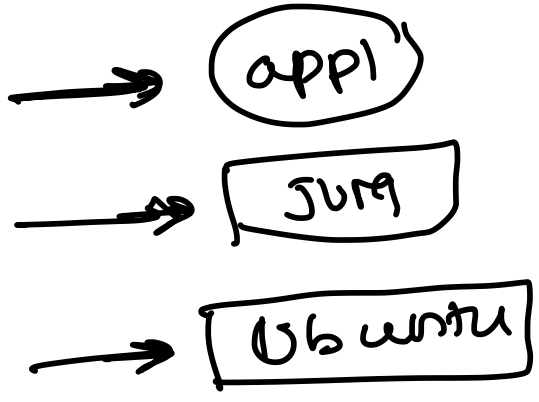
Docker Objects

- **Images**: read only template with instructions for creating docker containers
- **Container**: running instance of a docker image
- **Network**: network interface used to connect the containers to each other or external networks
- **Volumes**: used to persist the data generated by and used by the containers
- **Registry**: private or public collection of docker images
- **Service**: used to deploy application in a docker multi node cluster

Docker Images

Docker Image

- Read only template which has instructions for running docker containers
- In order to run a container, developer first need to package the application along with its dependencies in the form of a docker image
- It is highly portable and can be shared over network, stored and updated
- Docker provides public or private registry which contains collection of pre-built images
- If the image you are looking for is not available publicly, you can create your image using Dockerfile
- One image can be based on another image



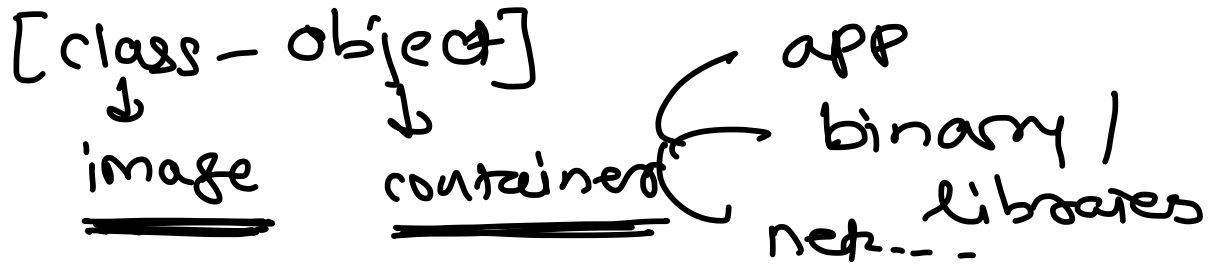
Demo

- Tasks
  - Search images on docker hub
  - Download/pull image
  - Get information of an image
  - List all pulled images
    - Remove an image

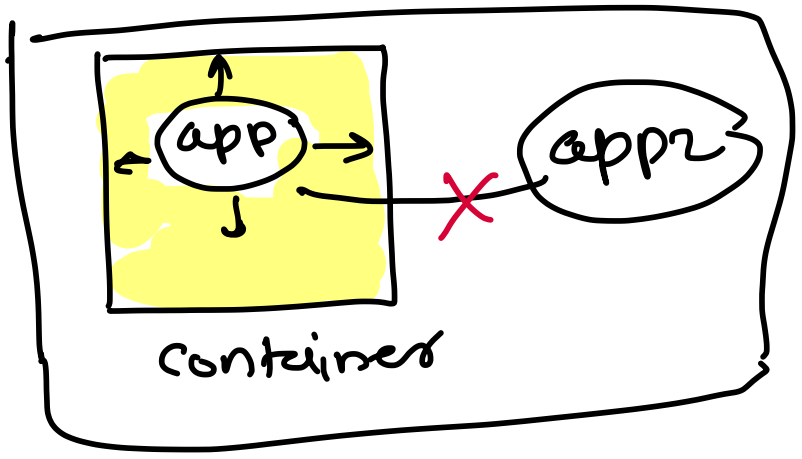
Docker Container

Container

- It is a running aspect of docker image
- Contains one or more running processes
- It is a self-contained environment

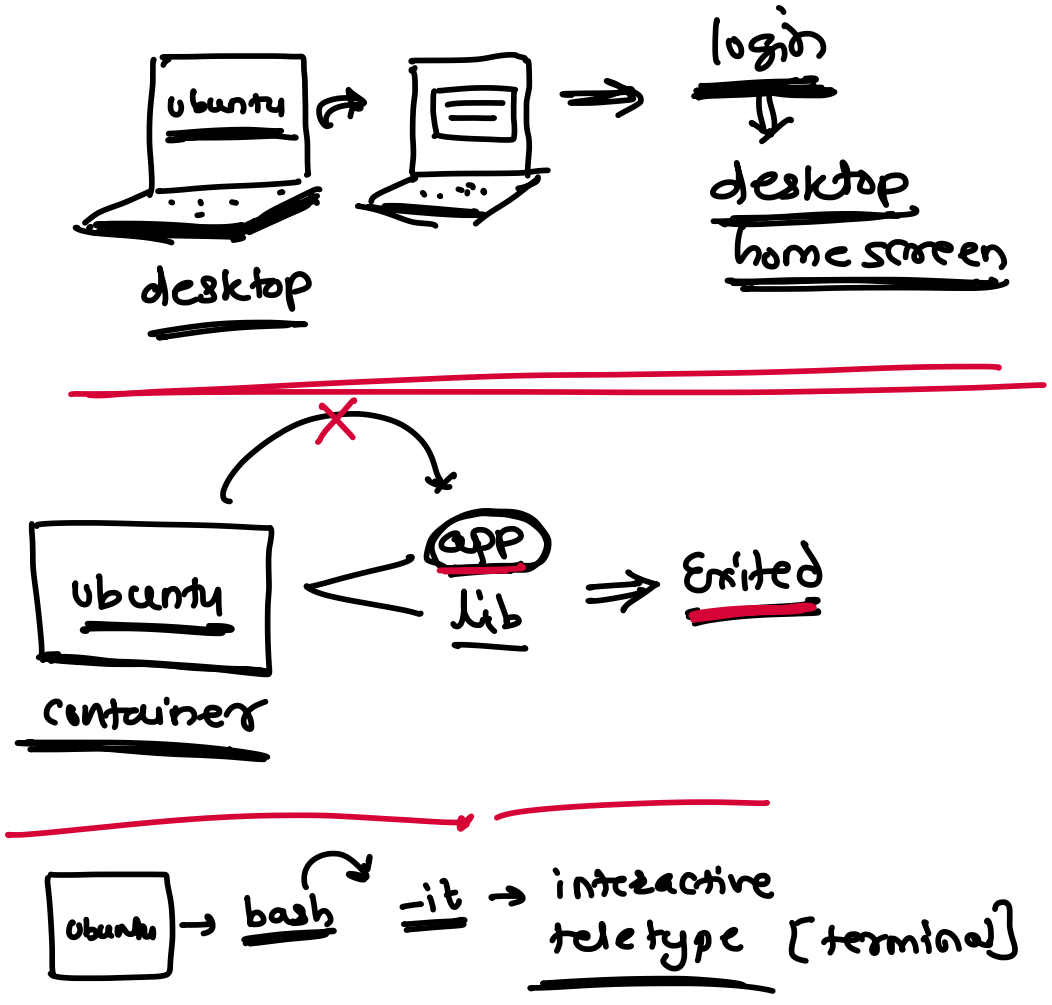


- It wraps up an application into its own isolated box (application running inside a container has no knowledge of any other applications or processes that exist outside the container)
- A container can not modify the image from which it is created
- It consists of
  - Your application code
  - Dependencies
  - Networking
  - Volumes



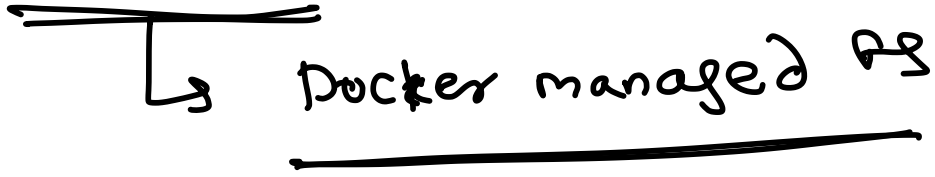
Demo

- Tasks
  - List the running containers on the host
  - Create a container
  - Start a created/stopped container
  - Run a container
  - Stop a container
  - Restart a container



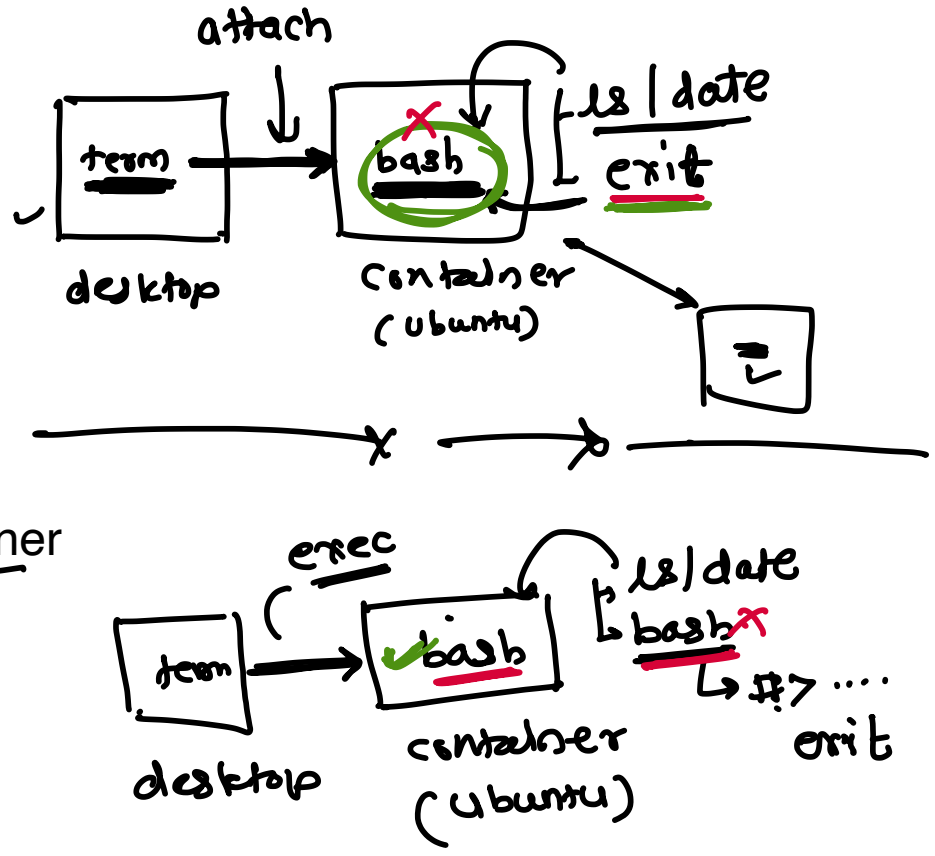
Where are the containers stored?

- Containers are stored under /var/lib/docker
- This directory contains
  - image
  - containers
  - network
  - volumes
  - swarm



Attaching a container

- There are two ways to attach to a container
- Attach
  - Used to attach the container
  - Uses only one input and output stream
  - Task
    - Attach to a running container
- Exec
  - Mainly it is used for running a command inside a container
  - Task
    - Execute a command inside container



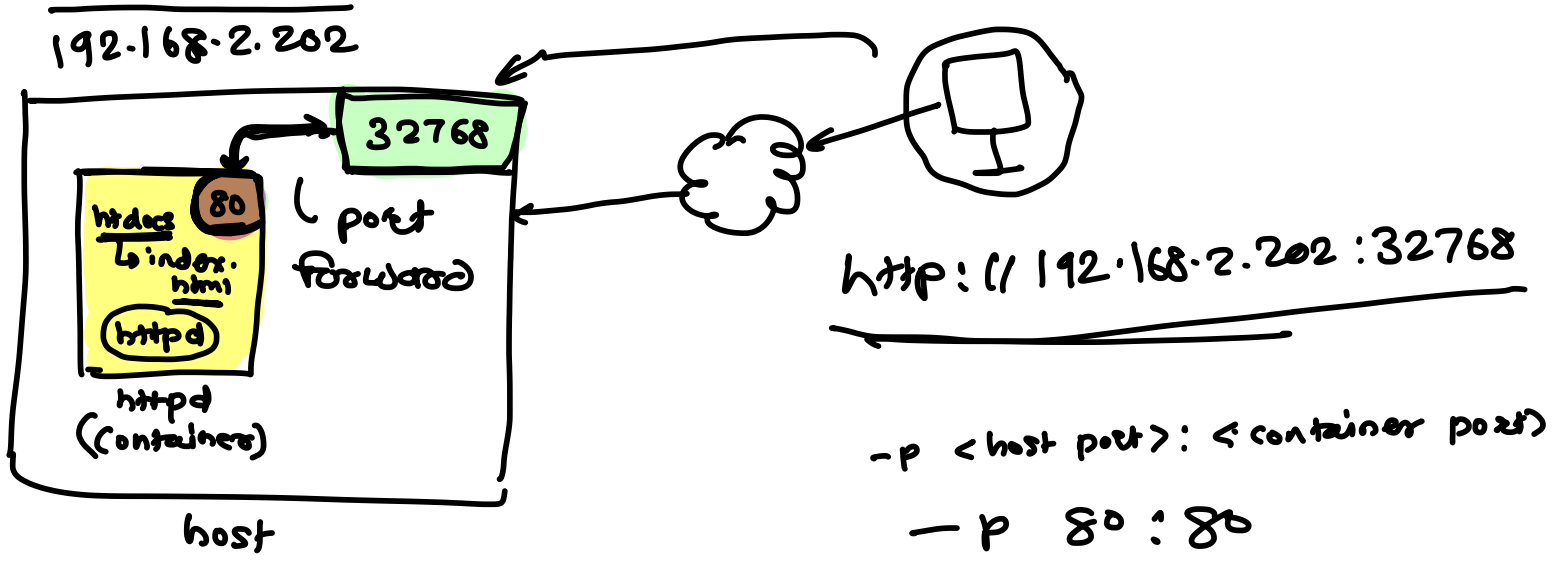
Hostname and name of container

- To check the host name
  - Go inside the container
  - Check the hostname by using a command hostname
- Docker uses the first 12 characters of container id as hostname
- Docker automatically generates a name at random for each container
- Task
  - Change the hostname of a container = --hostname
  - Chane the container name of a container = --name

run

Publishing port on container

- Publishing a port is required to give an external access to your application
- Port can be published only at the time of creating a container
- You can not update the port configuration on running container
- Task
  - Run a httpd container with port 8080 published, to access apache externally



Container information

- Docker top command ← top processes
- Docker stats command ← summary
- Docker inspect command ← information

Deleting container

- To delete container use rm command
- To delete the container automatically use – – rm

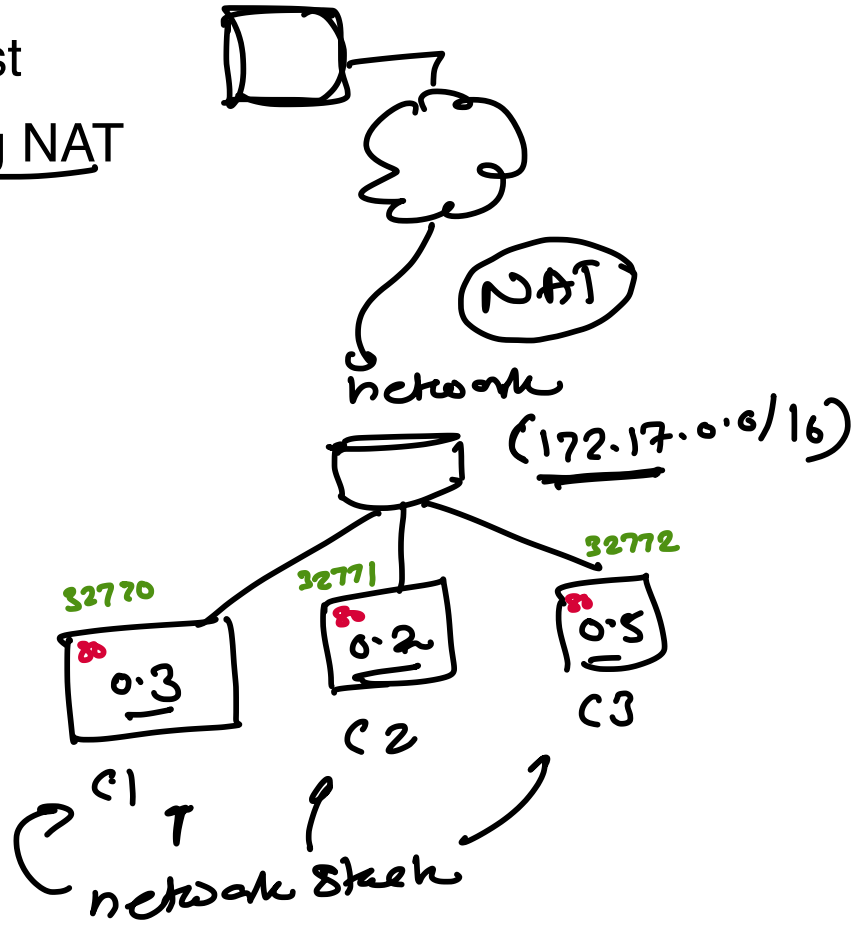
Docker Network

Overview

- By default docker creates following networks on the host
  - Bridge
  - Host - only
  - None
- Task
  - Check the networks on the host machine
  - Get more information of any network

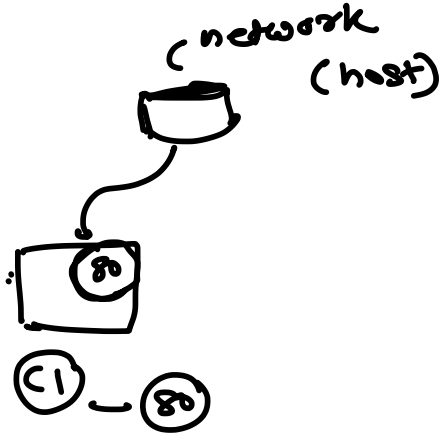
Bridge network

- Containers run on a separate network stacks, internal to docker host
- All of the containers share the external IP of the host machine using NAT
- Docker by default puts new container on bridge network
- Task
  - Get information about the bridge network
  - Run two containers on bridge network with same port published



Host network

- Containers behave just as any other process running in the docker host
- Host network adds the containers on the host's network stack
- There will be no isolation between the host machine and the container
- Does not perform any operation on incoming traffic (NAT)
- Task
  - Run a container on host network and verify the IP address
  - Run two containers on host network with same port published



Modify network settings on container

- Docker allows to modify the network settings without the need to restart the container
- Tasks
  - Start a container on none network
  - Disconnect the none network
  - Connect to bridge network

Custom network

- Docker network command can be used to create custom networks
- To create a custom network we have to use a driver → bridge
- If driver is not mentioned then docker uses bridge by default
- We can create as many networks as we need
- Tasks
  - Create a custom bridge network
  - Check the network interfaces on the docker host
  - Run a container using the newly created network

Remove the network

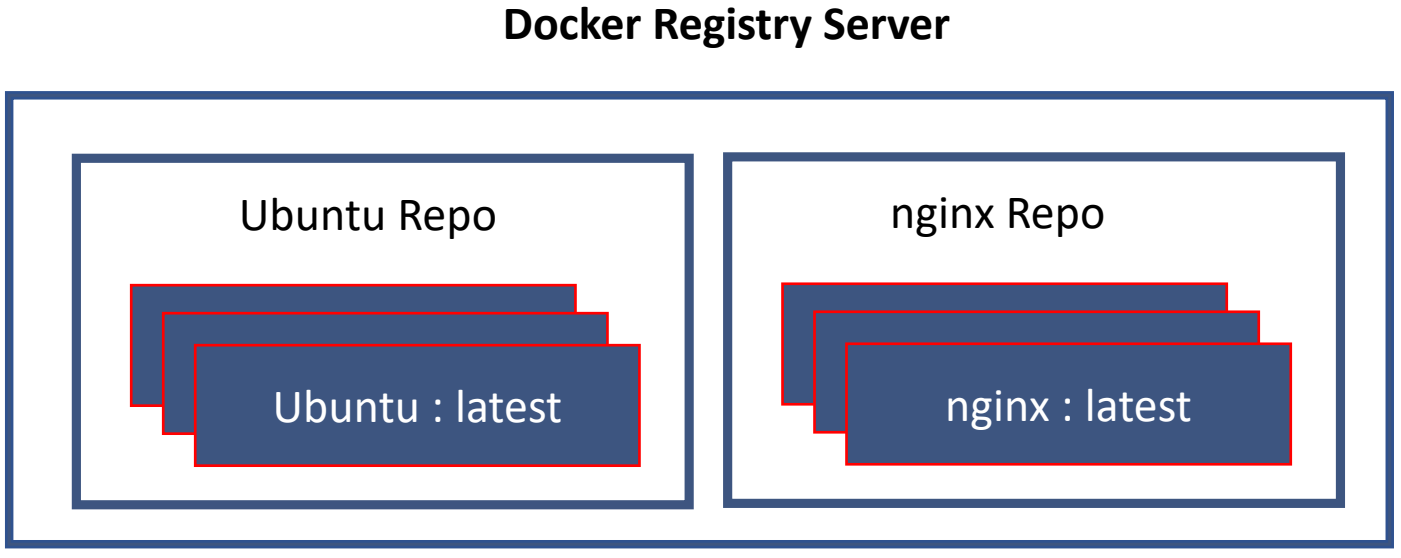
- Default networks can not be removed
- Active networks can not be removed
- Tasks
  - Create a custom network
  - Remove that custom network

- Prune command can be used to remove all unused networks

Docker Images (Advanced)

Docker Image

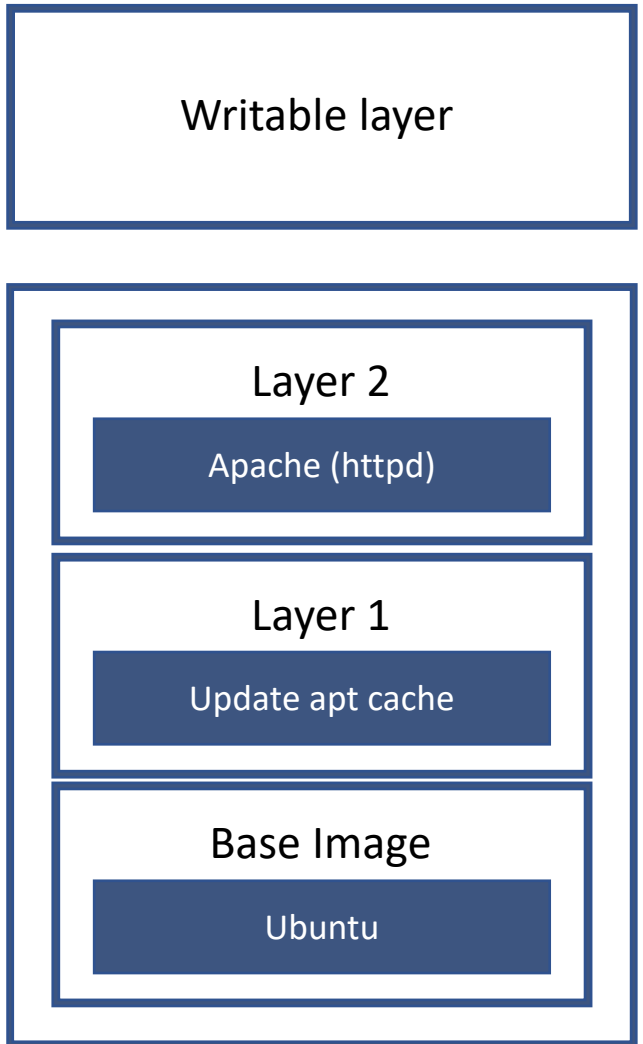
- Read-only instructions to run the containers
- It is made up of different layers
- Repositories hold images
- Docker registry stores repositories
- To create a custom image
  - Commit the running container
  - Use a Dockerfile



- Task
  - Create a container
  - Create a directory and a file within it
  - Commit the container to create a new image

Layered File System

- Docker images are made of layered FS
- Docker uses UnionFS for implementing the layered docker images
- Any update on the image adds a new layer
- All changes made to the running container are written inside a writable layer



Dockerfile

- The Dockerfile contains a series of instructions paired with arguments
- Each instruction should be in upper-case and be followed by an argument
- Instructions are processed from top to bottom
- Each instruction adds a new layer to the image and then commits the image
- Upon running, changes made by an instruction make it to the container

Dockerfile instructions

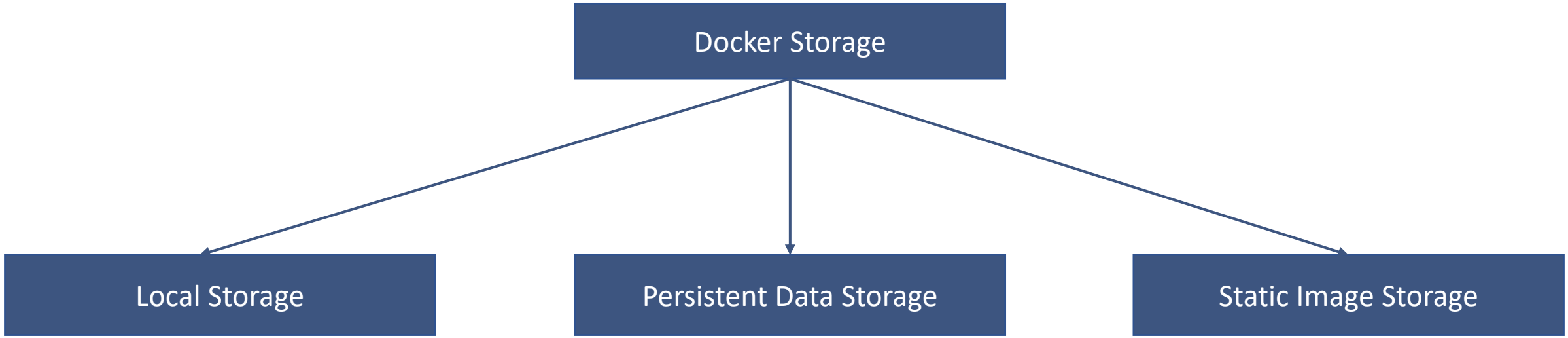
- FROM
- ENV
- RUN
- CMD
- EXPOSE
- WORKDIR
- ADD
- COPY
- LABEL
- MAINTAINER
- ENTRYPOINT

Sharing docker images

- Docker provides public docker hub to share the images
- Task
  - Create an image using Dockerfile
  - Push the image to dockerhub

Docker Volume

Overview



- Storage provided for docker image run
- Uses storage drivers to read FS layers from a container

- Saving data beyond the container lifecycle
- Data is stored outside the container

- Storing images in the docker registry
- Image stored in the registry will not run unless container is spawned

Storage drivers

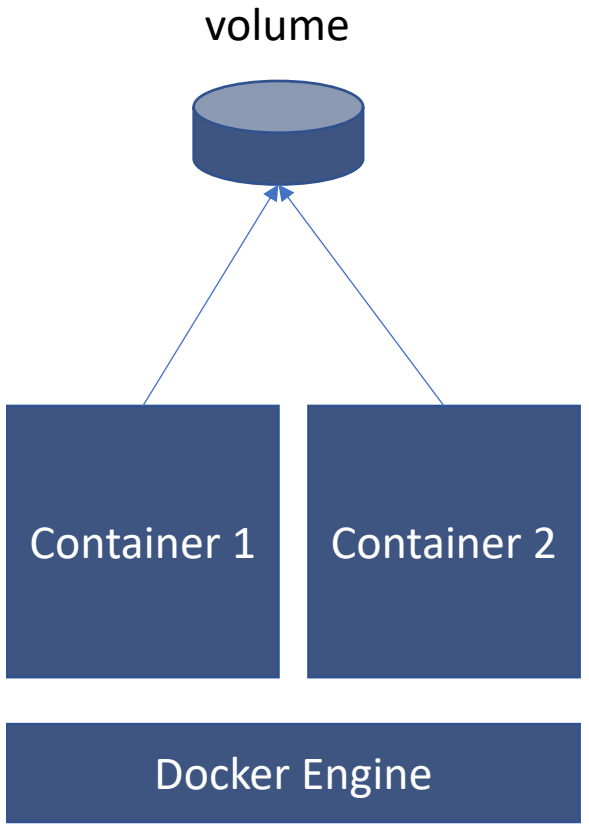
- Docker supports several different storage drivers
- E.g.
  - Overlay2
    - preferred storage driver, for all currently supported Linux distributions,
    - Requires no extra configuration
  - Aufs
    - Preferred storage driver for Docker 18.06 and older, when running on Ubuntu 14.04 on kernel 3.13 which has no support for overlay2
  - Devicemapper
    - is supported, but requires direct-lvm for production environments
  - Btrfs and zfs
    - Used if they are the backing filesystem (snapshots)
  - Vfs
    - Intended for testing purposes
- Tasks
  - Get the docker disk usage
  - Get the current storage driver configured

Local Storage

- Size taken by container
  - Size: data on the writable layer
  - Virtual Size: read-only image data + writable layer size
- Multiple containers share the image hence the image size will be shared
- Tasks
  - Create a container
  - Get the size information

Persistent Storage

- Downsides of using local storage for containers
  - Data does not persist when container is removed
  - Writable layer is tightly coupled to the host machine
- Volume provides persistent storage
- Allows to share the data among containers
- Can be managed using the docker CLI commands
- NOTE: Volume does not increase the size of container using it



Persistent Storage

- Volumes
  - Stored in the docker managed FS of the host
  - Supports the use of Volume Drivers
- BindMounts
  - Stored anywhere in the host
  - E.g. you can mount a local directory with the container to share the contents
- Tmpfs Mounts
  - Temporary and stored in the host's memory
  - When the container stops, the tmpfs mount is removed
  - If the container is committed then tempfs is not saved
  - Available only with docker on linux