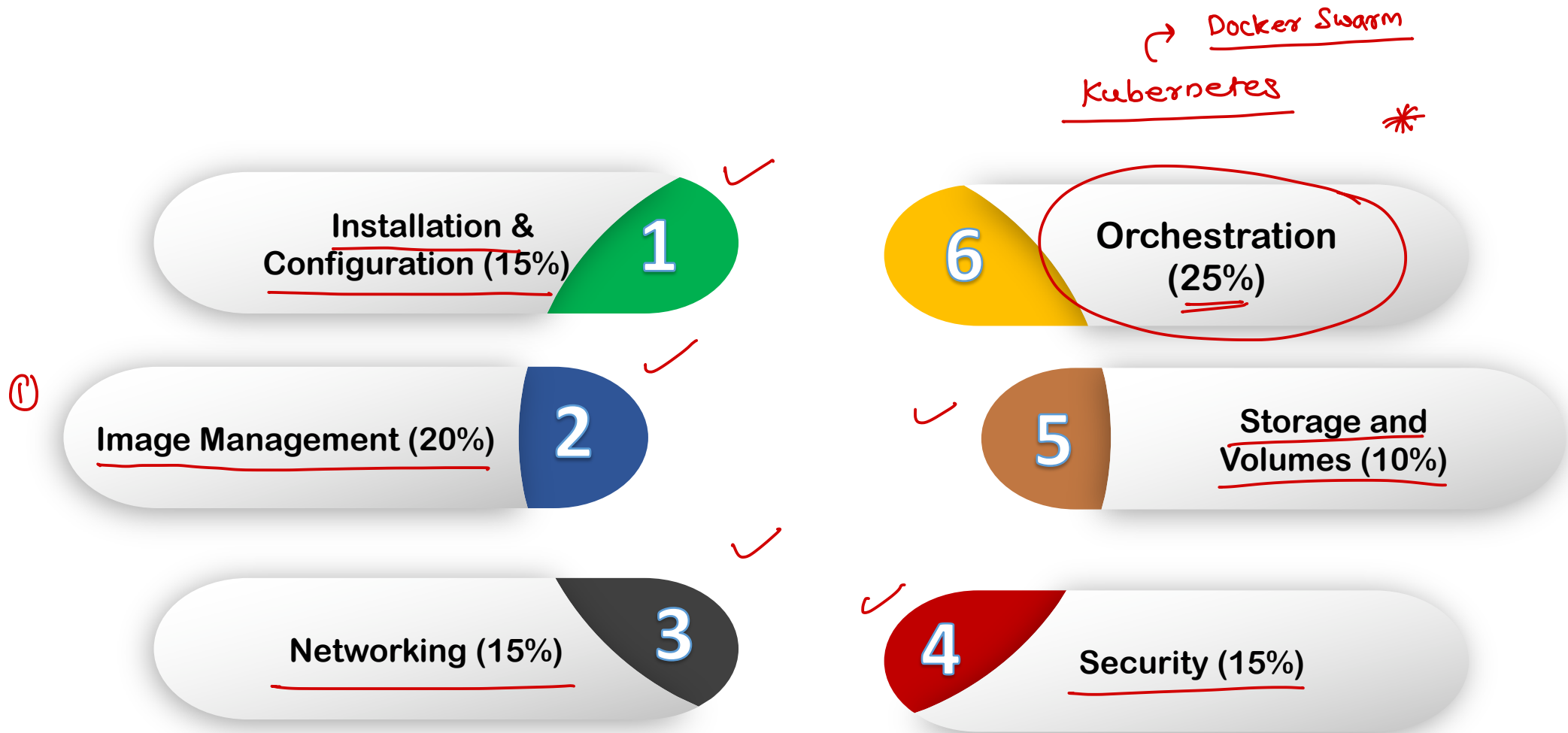


docker

About your instructor

- 12+ years of experience
- Associate Technical Director at Sunbeam
- Worked in various domains using different technologies
- Developed
 - 180+ mobile applications on iOS and Android platforms
 - Various websites using PHP, MEAN and MERN stacks
 - Various machine learning solutions (using Python)
- Certification completed
 - ✓ Certified Kubernetes Application Developer (CKAD)
 - ✓ Certified Kubernetes Administrator (CKA)
 - ✓ Certified Jenkins Engineer (CJE)
 - ✓ Docker Certified Associate (DCA) ➡





Installation and Configuration (15% of exam)

- Demonstrate the ability to upgrade the Docker engine
- Complete setup of repo, select a storage driver, and complete installation of Docker engine on multiple platforms
- Configure logging drivers (splunk, journald, etc)
- Setup swarm, configure managers, add nodes, and setup backup schedule
- Create and manager user and teams
- Interpret errors to troubleshoot installation issues without assistance
- Outline the sizing requirements prior to installation
- Understand namespaces, cgroups, and configuration of certificates
- Use certificate-based client-server authentication to ensure a Docker daemon has the rights to access images on a registry
- Consistently repeat steps to deploy Docker engine, UCP, and DTR on AWS and on premises in an HA config 1, 2, 3
- Complete configuration of backups for UCP and DTR
- Configure the Docker daemon to start on boot



Image Creation, Management, and Registry (20% of exam)

- Describe Dockerfile options(add, copy, volumes, expose, entrypoint, etc)
- Show the main parts of a Dockerfile
- Give examples on how to create an efficient image via a Dockerfile
- Use CLI commands such as list, delete, prune, rmi, etc to manage images
- Inspect images and report specific attributes using filter and format
- Demonstrate tagging an image
- Utilize a registry to store an image
- Display layers of a Docker image
- Apply a file to create a Docker image
- Modify an image to a single layer
- Describe how image layers work
- Deploy a registry (not architect)
- Configure a registry
- Log into a registry
- Utilize search in a registry
- Tag an image
- Push an image to a registry
- Sign an image in a registry
- Pull an image from a registry
- Describe how image deletion works
- Delete an image from a registry



Networking (15% of exam)

- Create a Docker bridge network for a developer to use for their containers
- Troubleshoot container and engine logs to understand a connectivity issue between containers
- Publish a port so that an application is accessible externally
- Identify which IP and port a container is externally accessible on
- Describe the different types and use cases for the built-in network drivers
- Understand the Container Network Model and how it interfaces with the Docker engine and network and IPAM drivers
- Configure Docker to use external DNS
- Use Docker to load balance HTTP/HTTPs traffic to an application (Configure L7 load balancing with Docker EE)
- Understand and describe the types of traffic that flow between the Docker engine, registry, and UCP controllers
- Deploy a service on a Docker overlay network
- Describe the difference between “host” and “ingress” port publishing mode (Host, Ingress)



Security (15% of exam)

- Describe the process of signing an image
- Demonstrate that an image passes a security scan
- Enable Docker Content Trust
- Configure RBAC in UCP
- Integrate UCP with LDAP/AD
- Demonstrate creation of UCP client bundles
- Describe default engine security
- Describe swarm default security
- Describe MTLS



Storage and Volumes (10% of exam)

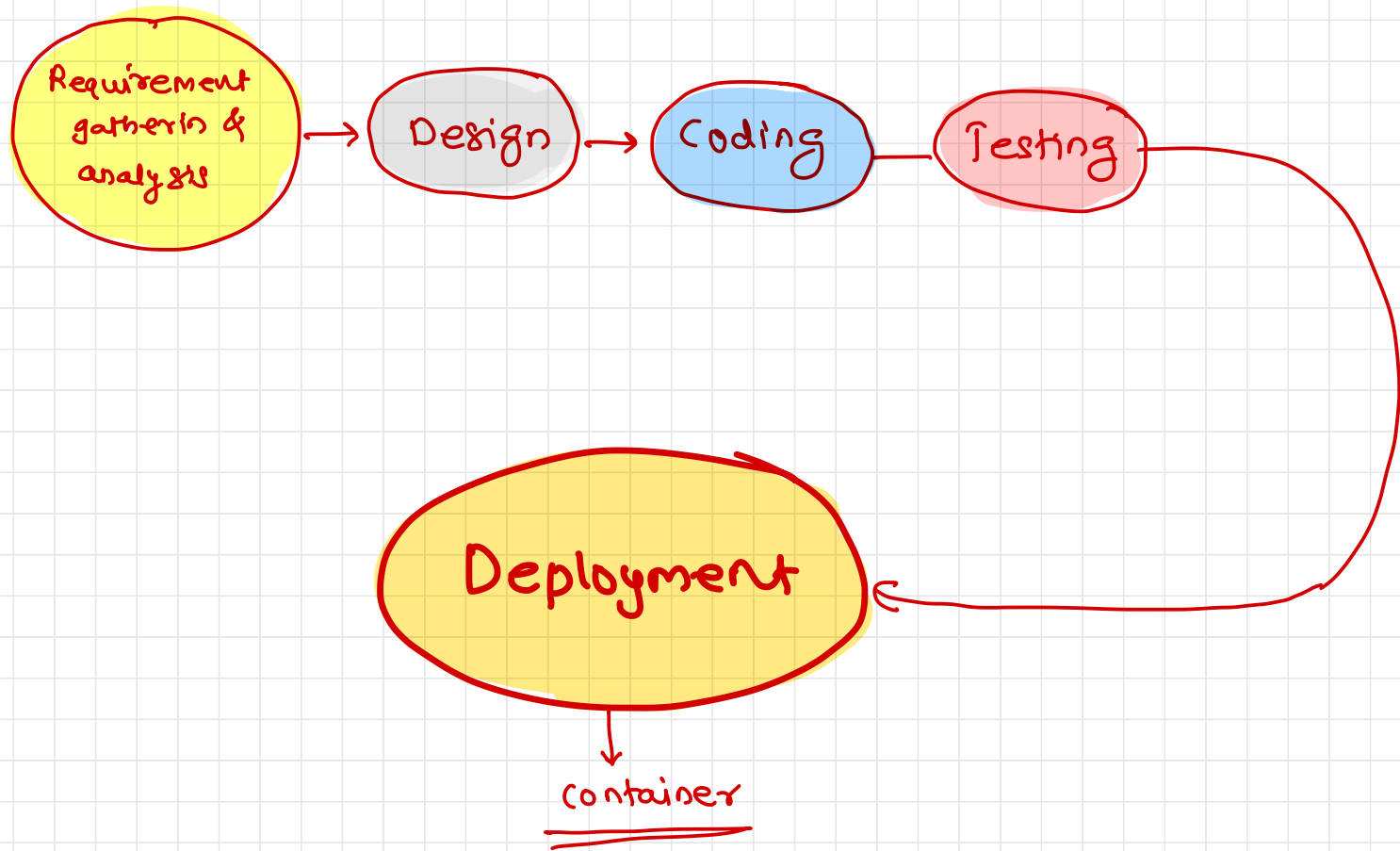
- State which graph driver should be used on which OS
- Demonstrate how to configure devicemapper
- Compare object storage to block storage, and explain which one is preferable when available
- Summarize how an application is composed of layers and where those layers reside on the filesystem
- Describe how volumes are used with Docker for persistent storage
- Identify the steps you would take to clean up unused images on a filesystem, also on DTR
- Demonstrate how storage can be used across cluster nodes
- Identity roles
- Describe the difference between UCP workers and managers
- Describe process to use external certificates with UCP and DTR (UCP from cli, from GUI, print the public certificates), DTR)



Orchestration (25% of exam)

- Complete the setup of a swarm mode cluster, with managers and worker nodes
- State the differences between running a container vs running a service
- Demonstrate steps to lock a swarm cluster
- Extend the instructions to run individual containers into running services under swarm
- Interpret the output of “docker inspect” commands
- Convert an application deployment into a stack file using a YAML compose file with “docker stack deploy”
- Manipulate a running stack of services
- Increase number of replicas
- Illustrate running a replicated vs global service
- Mount volumes
- Add networks, publish ports
- Identify the steps needed to troubleshoot a service not deploying
- Apply node labels to demonstrate placement of tasks
- Sketch how a Dockerized application communicates with legacy systems
- Paraphrase the importance of quorum in a swarm cluster
- Demonstrate the usage of templates with “docker service create”





Containerization

Docker

Self managed
cluster

cloud managed
services

✓ AWS
├── ECS
└── EKS

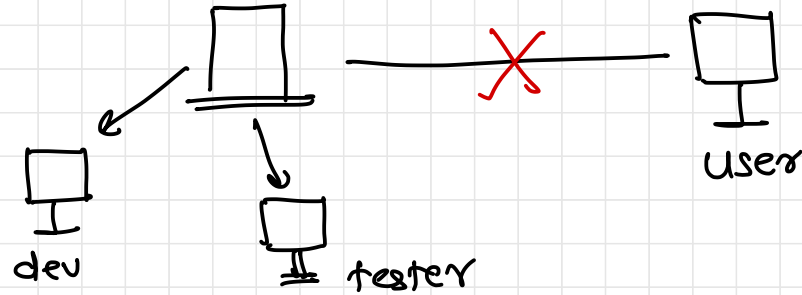
✓ Azure
├── CS
└── kube

✓ GCP
Kube

Fundamentals



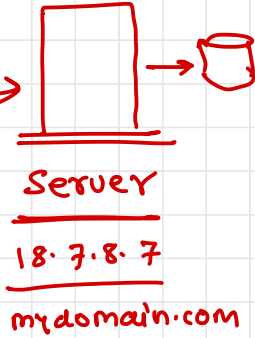
Dev / Test



deployment

production

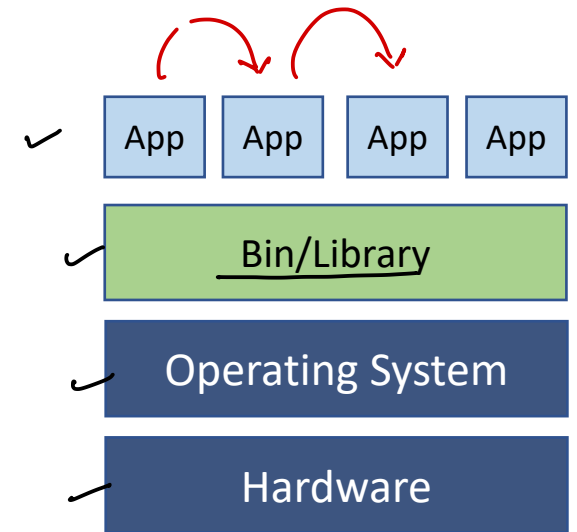
http:// ↔ /..

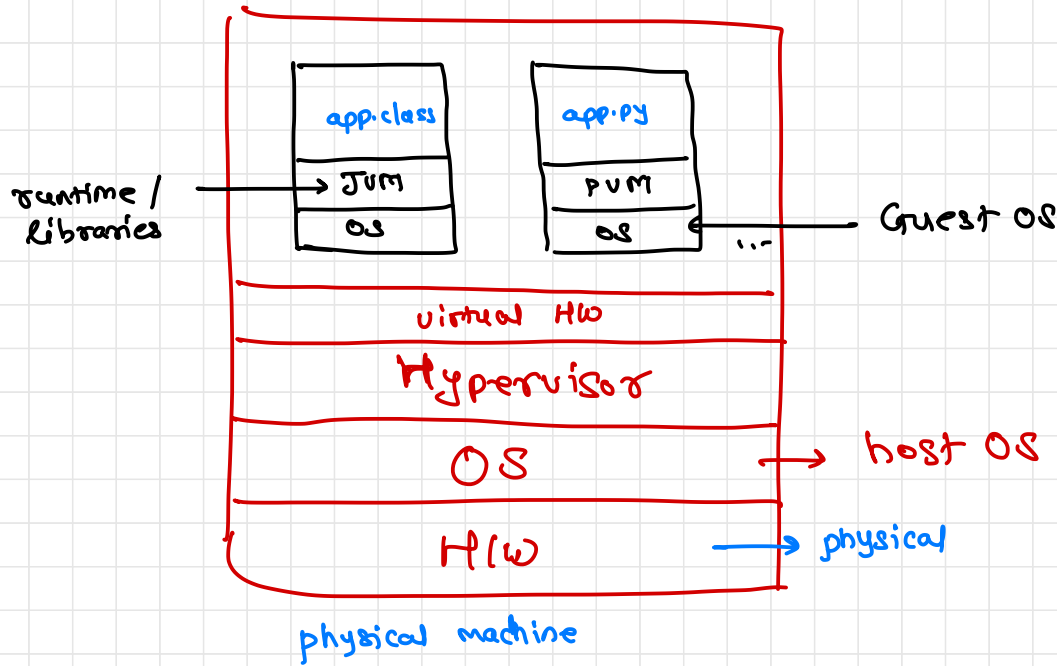


Traditional Deployment (Pre-Virtualization)

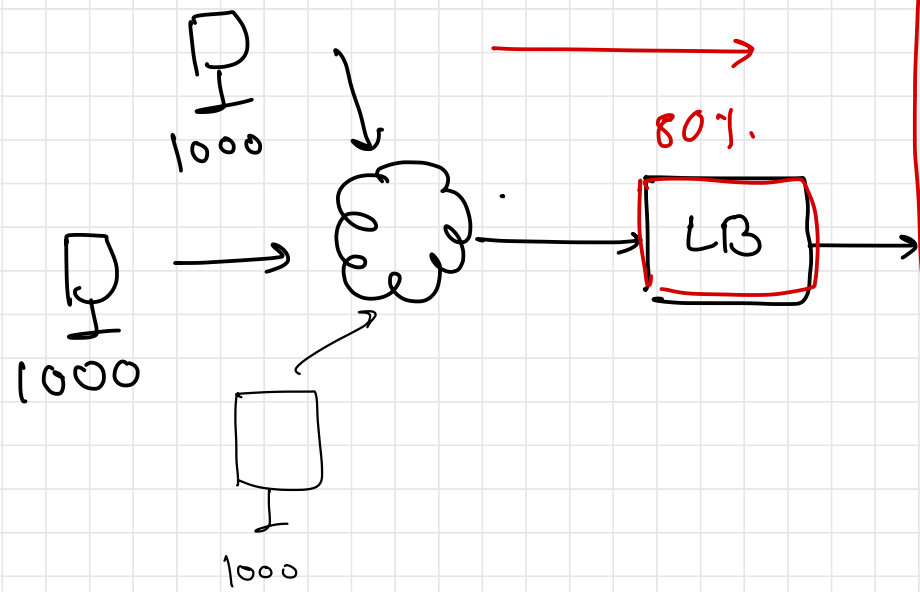
physical machines

- 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

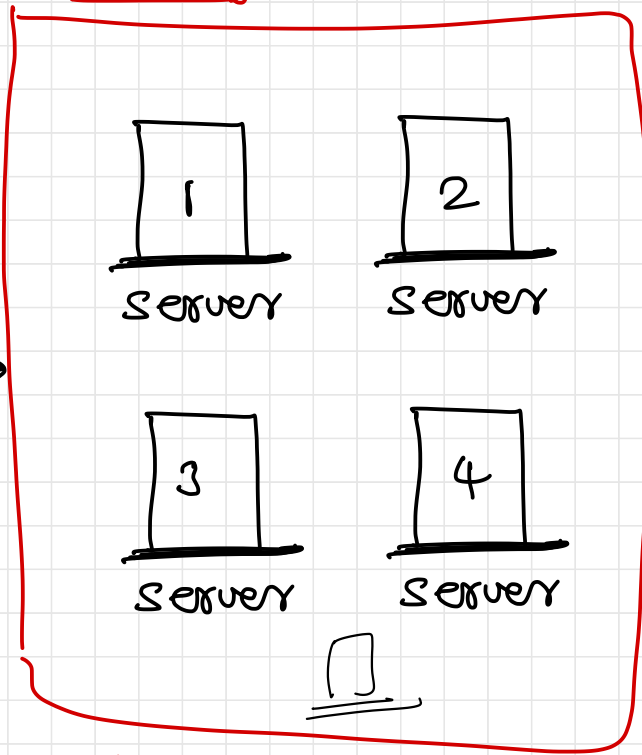




horizontal scaling



Elasticity

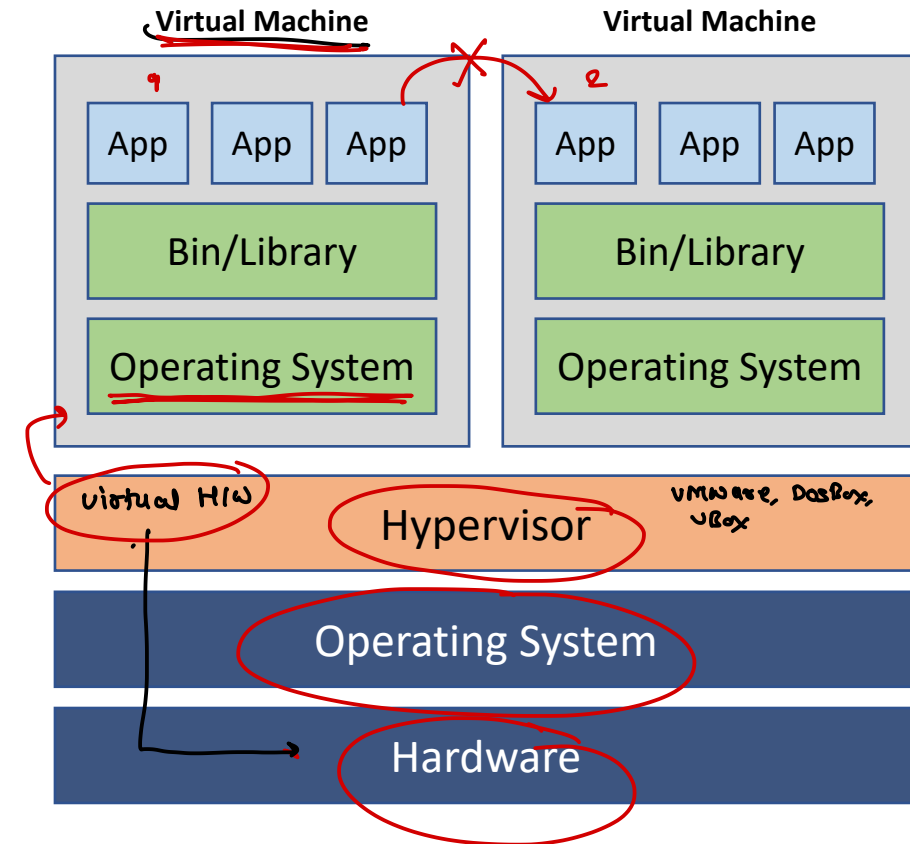


Cluster (physical machine)

Virtualized Deployment

6
0

- 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



Virtual machine

type I ✓

→ VMware ESX

→ Xen VM



No need of Host OS

type II ✓

→ VMware desktop

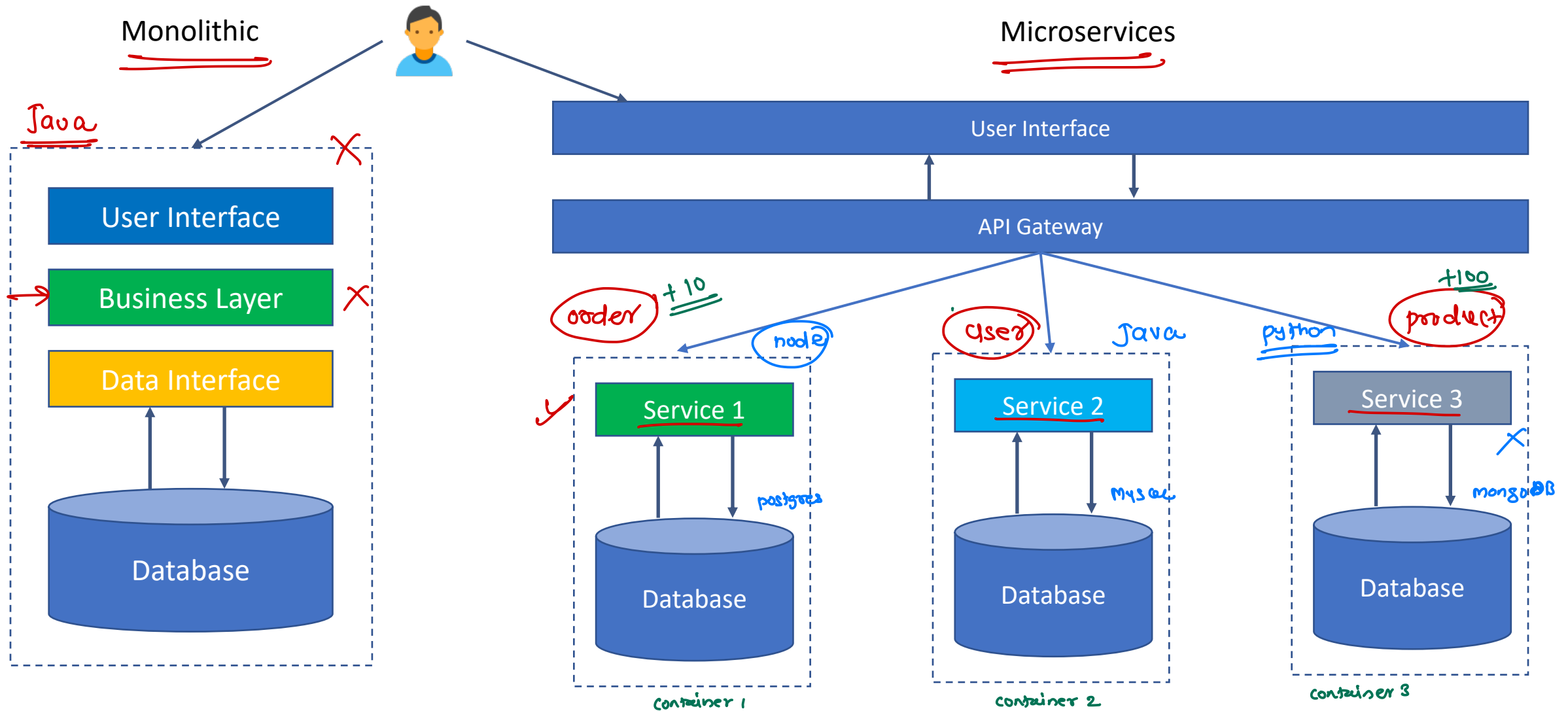
→ Virtual Box

→ Docker



Host OS is needed

Monolithic vs Microservice



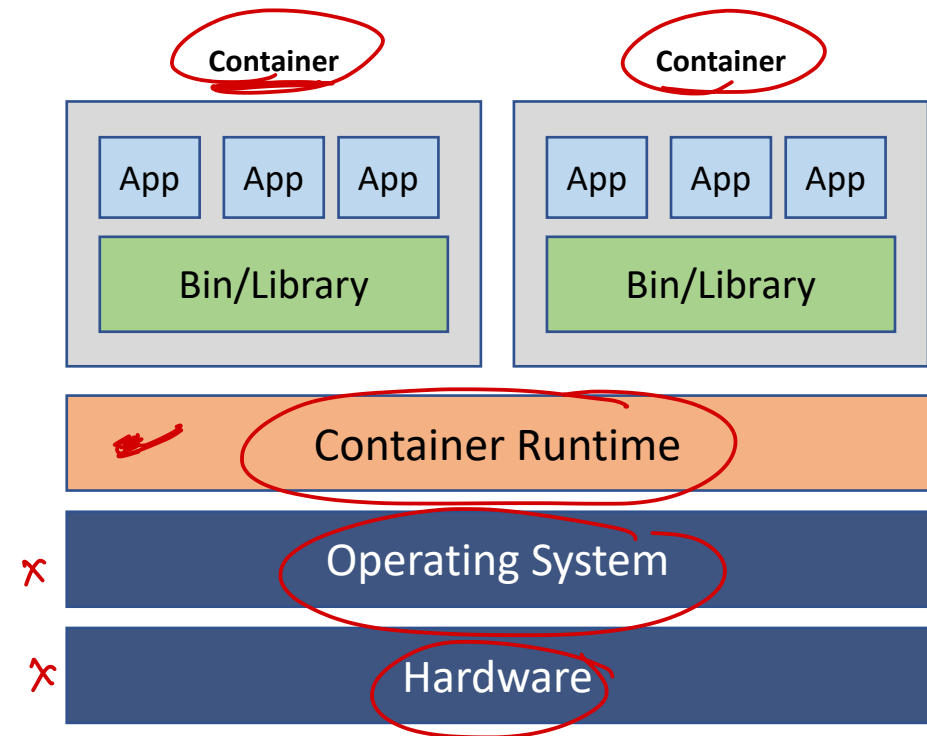
Microservice

- Distinctive method of developing software systems that tries to focus on building single-function modules with well-defined interfaces and operations
- Is an architectural style that structures an application as a collection of services that are
 - Highly maintainable and testable
 - Loosely coupled
 - Independently deployable
 - Organized around business capabilities



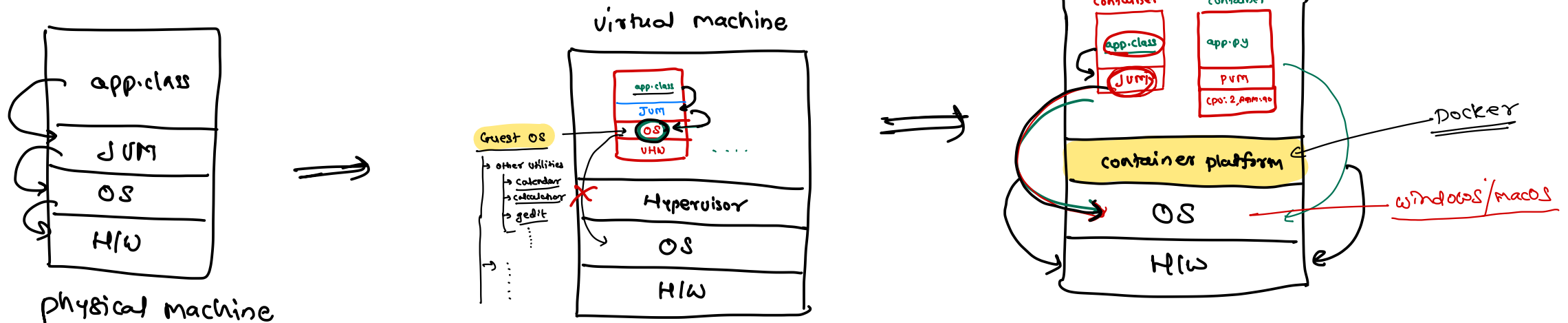
Container deployment

- 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



Container

- Allows developers to create and deploy applications ^{vm} faster and ^{physical servers} more securely
- A container is a standard unit which provides single point of service
- 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



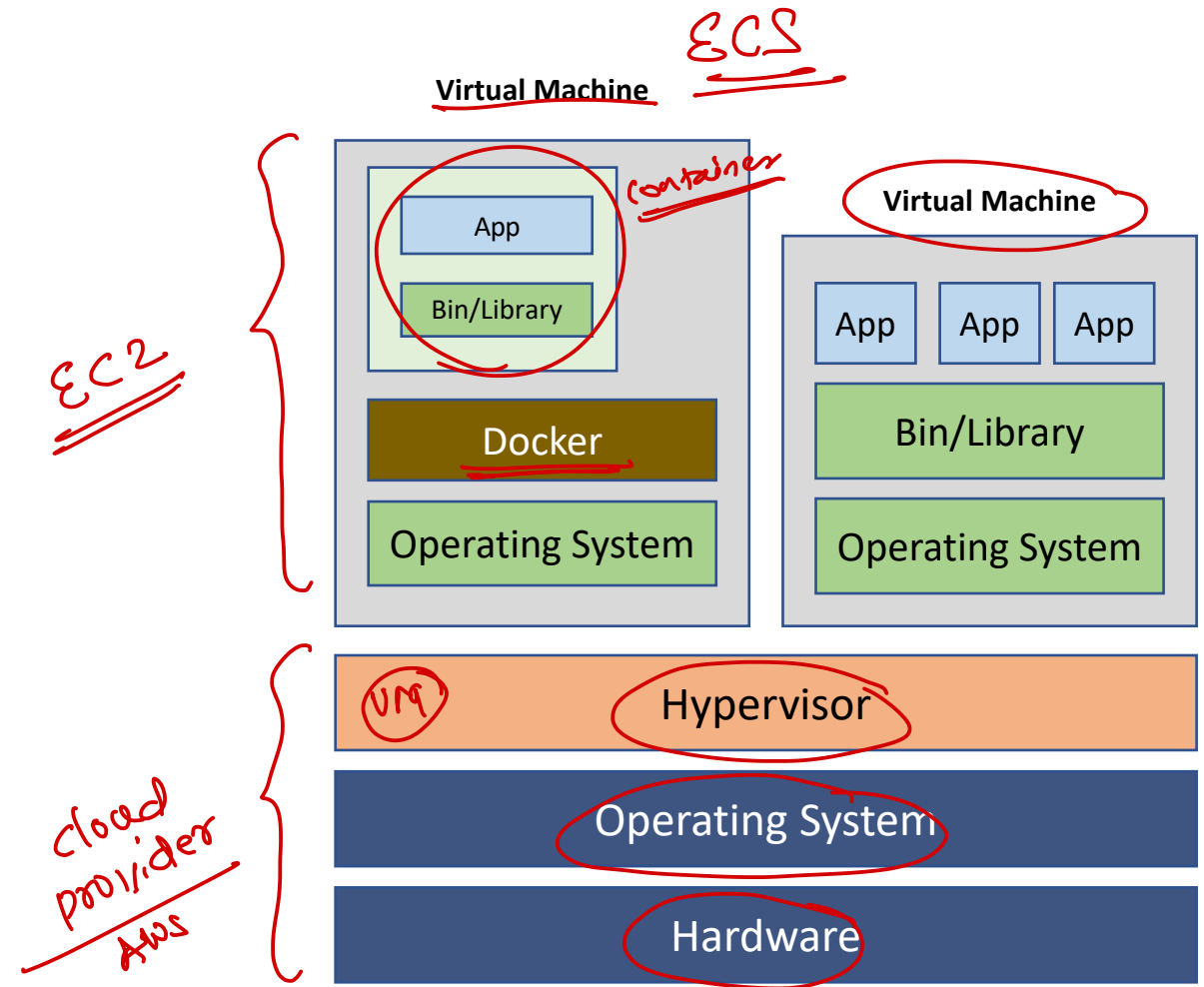
Containerization vs Virtualization

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



Containerization and Virtualization

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



Docker



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



Why Docker?

- It is an easy way to create application deployable packages (image)
- Developer can create ready-to-run containerized applications
- It provides consistent computing environment (os or platform or architecture neutral)
- 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 PaaS 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 Imctfy (Let Me Contain That For You)
 - Rkt is another option for running containers



Docker Editions

Feature	✓ Docker CE	✓ Docker EE
<u>Container engine with orchestration, networking, security</u>	✓	✓
<u>Certified infrastructure, plugins and ISV containers</u>		✓
<u>Image Management</u>		✓
<u>Container App Management</u>		✓
<u>Secure Image Scanning</u>		✓



Where are you running your containers

Docker Desktop

- Free
- Runs on Windows and macOS
- Installs Docker CE
- Community Support is available

Docker CE

- Server or VM for testing, training and development
- Free
- Runs on Windows and macOS
- Known as CE
- Community support is available

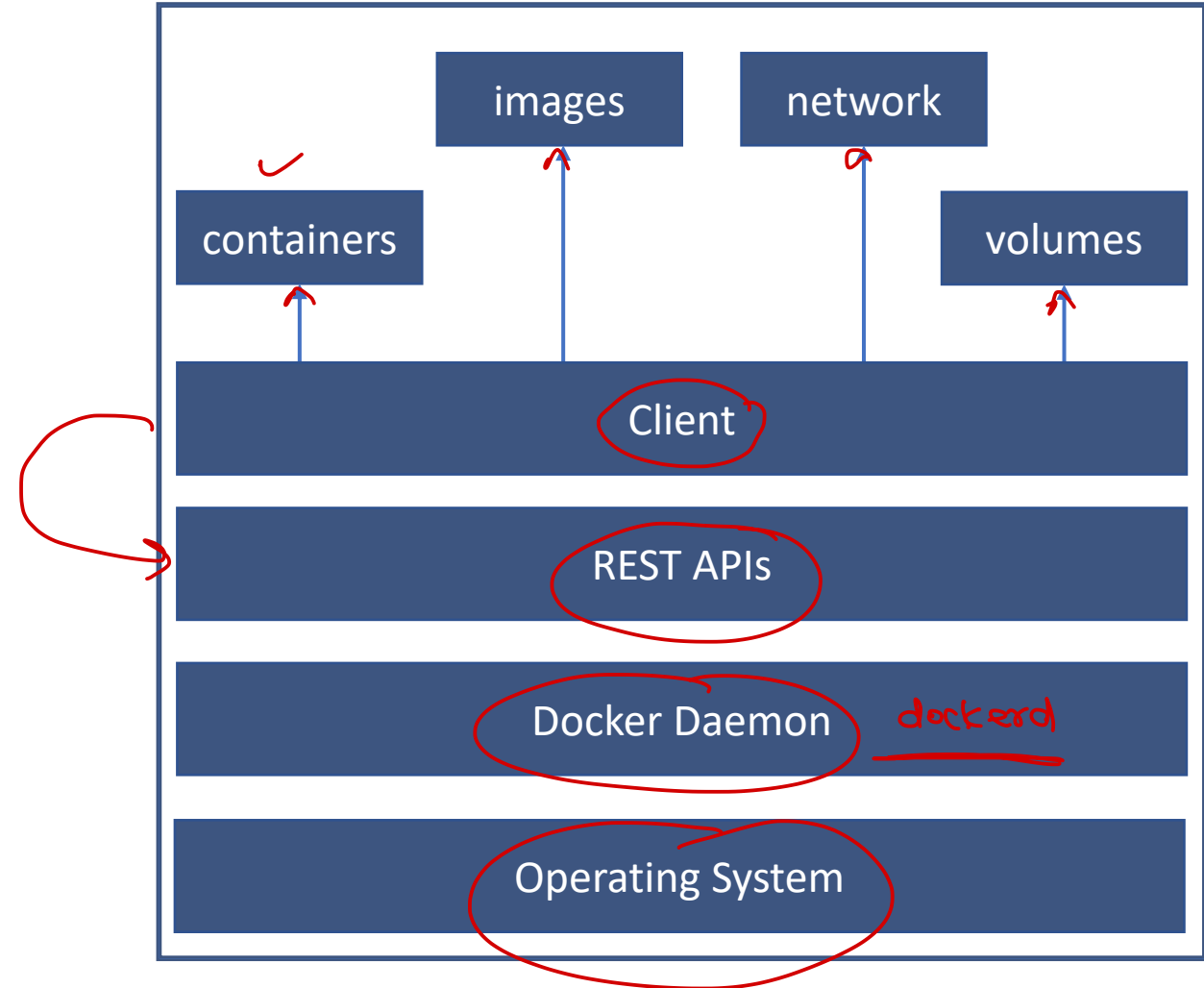
Docker EE

- Production applications in data center or cloud
- Paid license
- Runs on Windows and Linux
- Known as EE
- Full support is available
- Includes advanced features



Docker Architecture

- Docker daemon (dockerd) - *server*
 - 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



libcontainer

- 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 containers
 - It optionally enforces limits and constraints on resource usage
 - It limits how much a container can use
 - E.g. CPU, Disk space, memory

↳ CPU | RAM



- 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 swarm

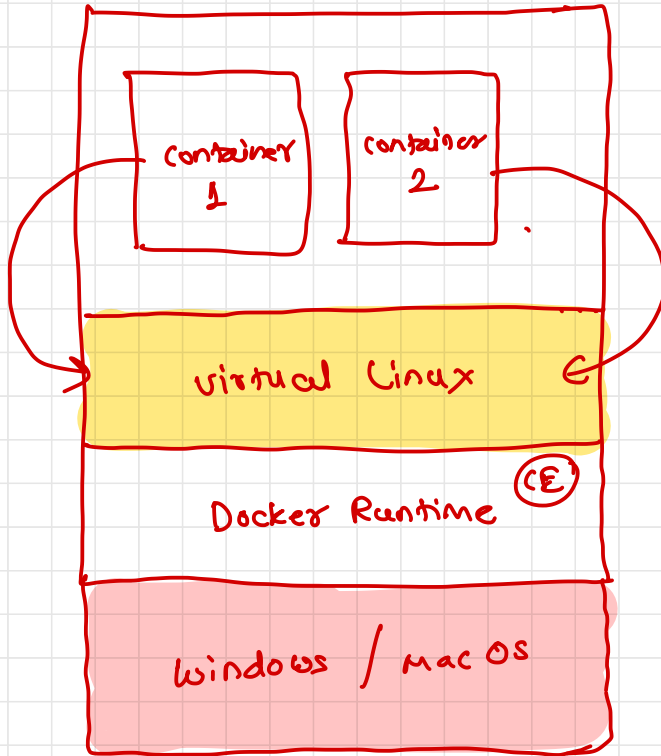


Installation

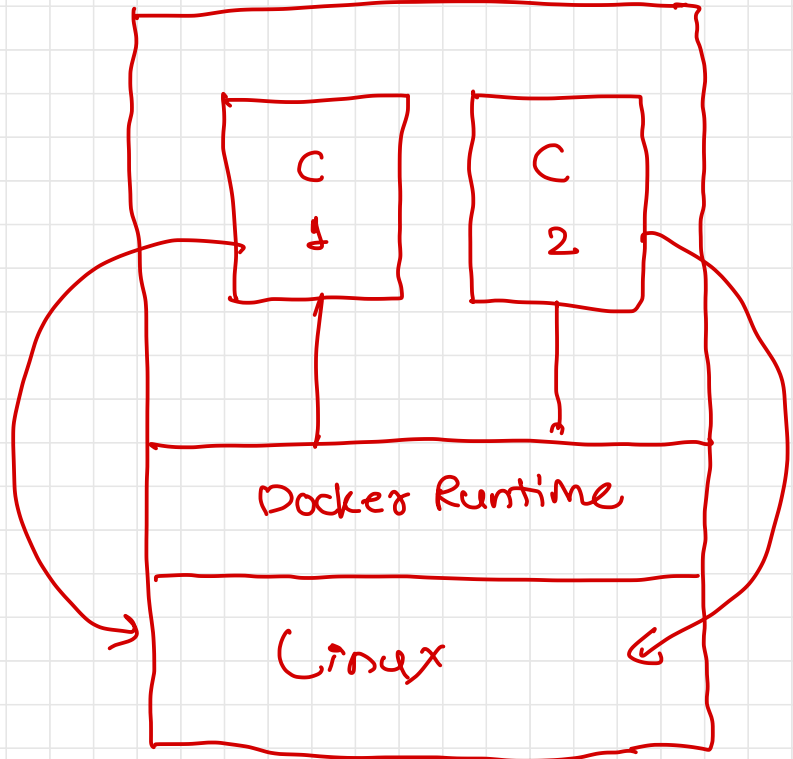
- macOS
 - <https://docs.docker.com/desktop/mac/install/>
- Windows
 - <https://docs.docker.com/desktop/windows/install/>
- Linux (Ubuntu)
 - <https://docs.docker.com/engine/install/ubuntu/>



Docker Desktop



Docker on Linux

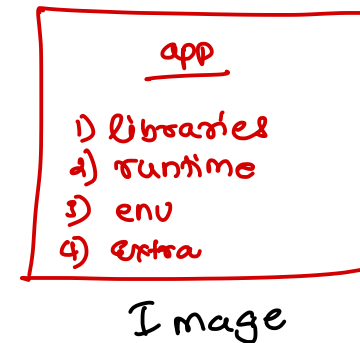
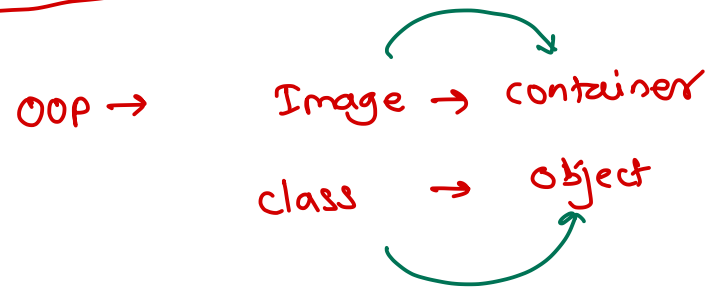


Docker Images



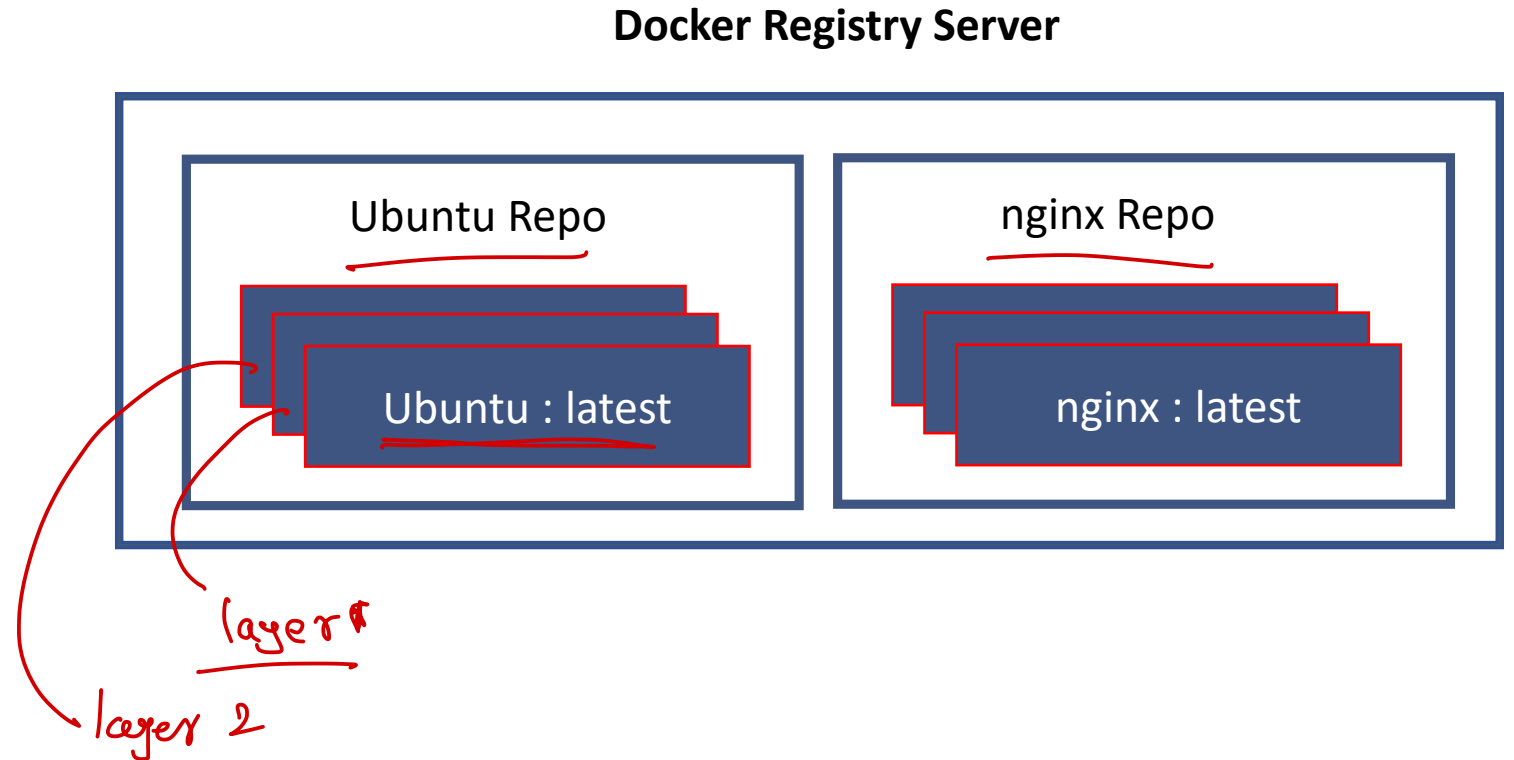
Docker Image

- An executable package that includes everything needed to run an application – the code, a runtime, libraries, environment variables and configuration files
- 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



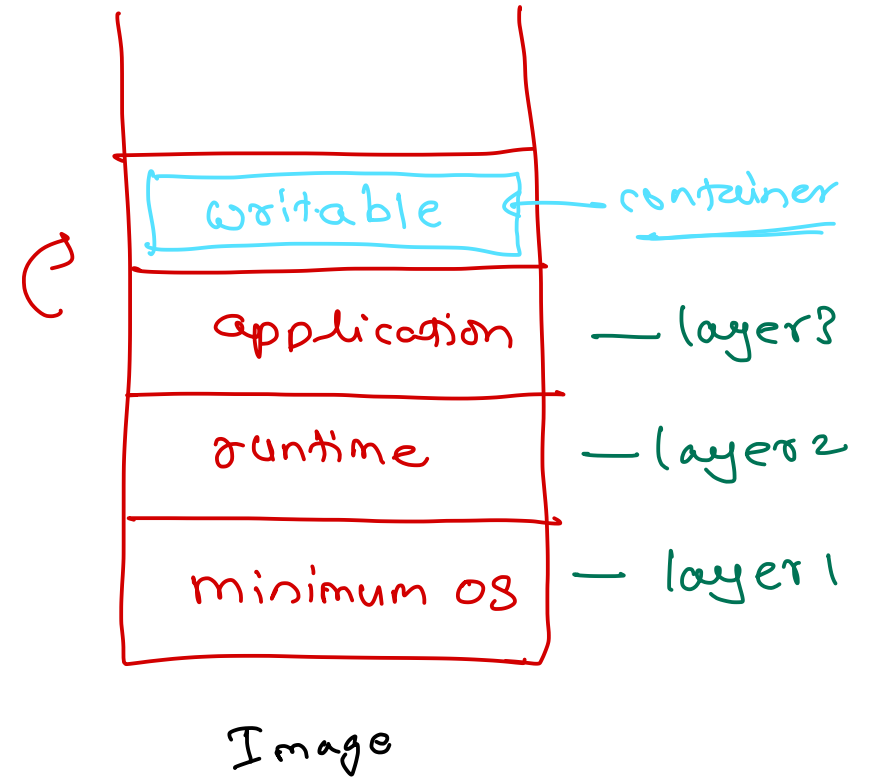
Docker Image

- 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



Layered File System

- Docker images are made up of multiple layers
- Docker uses UnionFS for implementing the layered docker images
- Any update on the image adds a new layer
- When an image is instantiated into a container, a top writable layer is created (which is deleted when container is removed)
- Docker uses storage drivers to manage the contents of image layer and writable layer
- Each storage driver handles the implementation differently, but all drivers use stackable image layers and copy-on-write (CoW) strategy



Managing Docker Images

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

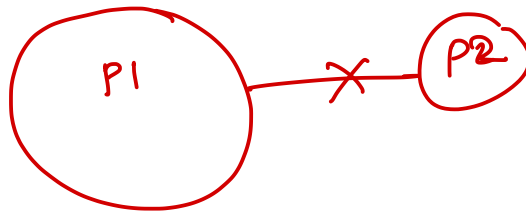


Docker Container



Container

- It is a running aspect of docker image
- Contains one or more running processes
- It is a self-contained environment \Rightarrow *application + libraries + env*
- 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

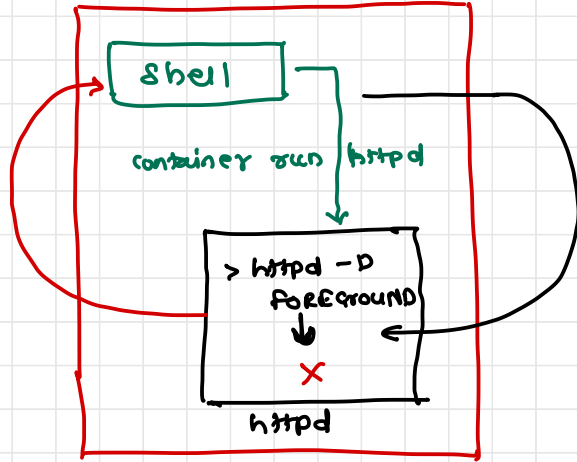


Managing Containers

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

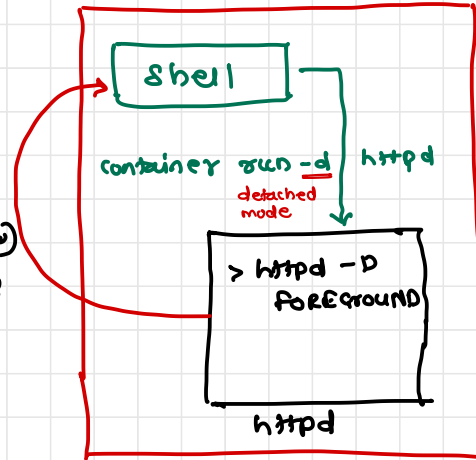


containers
Exits &
control
returns to
shell



attached
container

container
runs in detached
mode (in background)
& control returns
to the shell



Where are the containers stored?

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



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



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



Container information

- Docker top command
- Docker stats command
- Docker inspect command



Docker Network



Overview

- By default docker creates following networks on the host
 - Bridge
 - Host
 - 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
- 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)



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

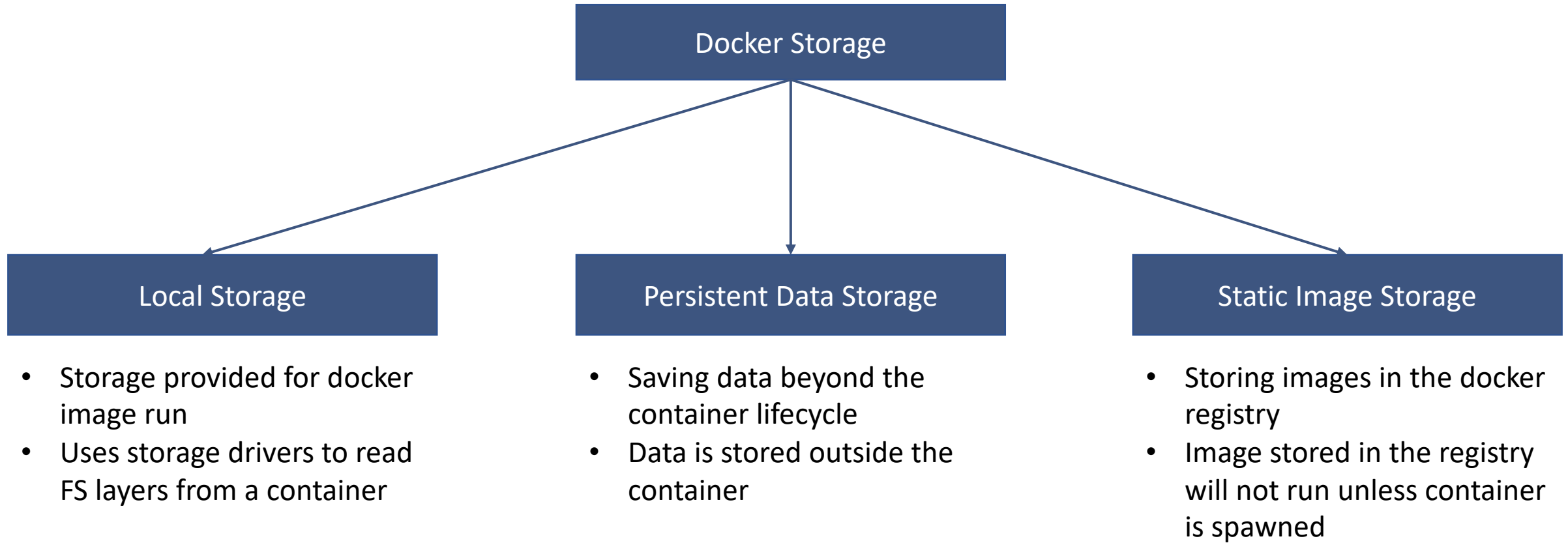
- Docker provides public docker hub to share the images
- Task
 - Create an image using Dockerfile
 - Push the image to dockerhub



Docker Volume



Overview



Storage 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



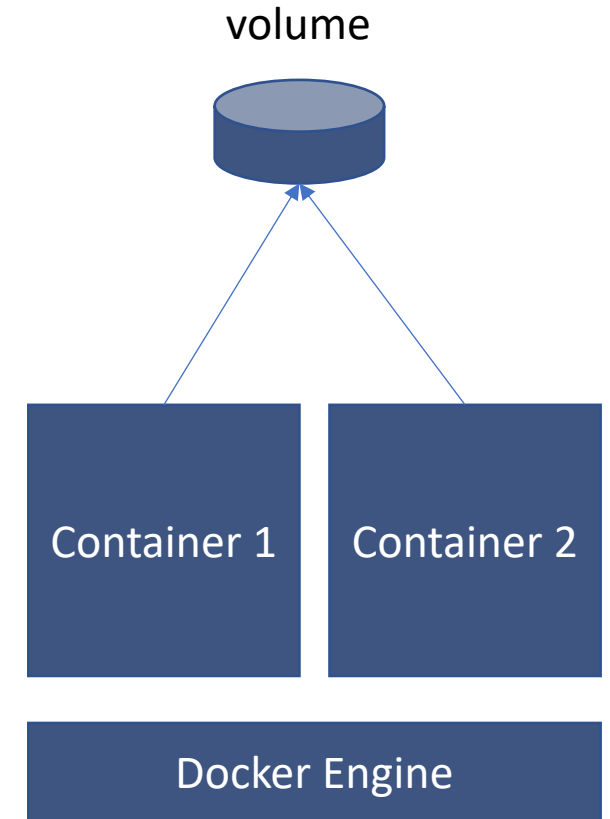
Local Storage

- 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 tmpfs is not saved
 - Available only with docker on linux

