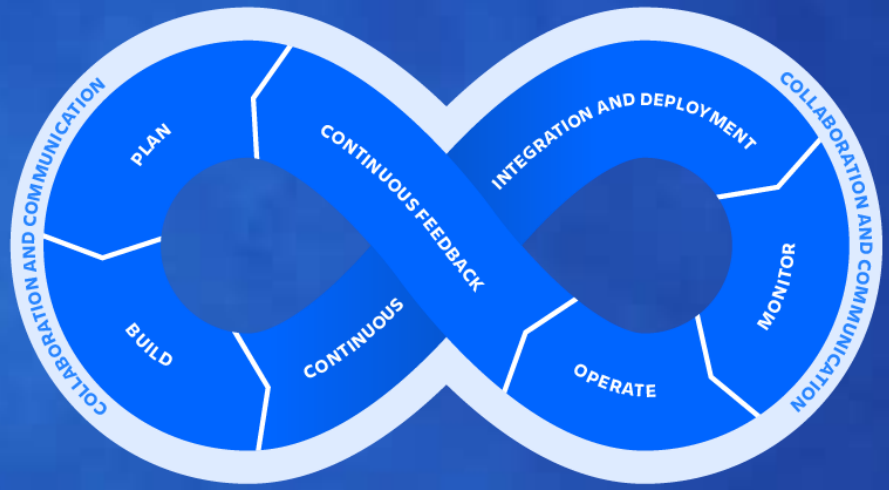


Session 10. Docker - Introduction

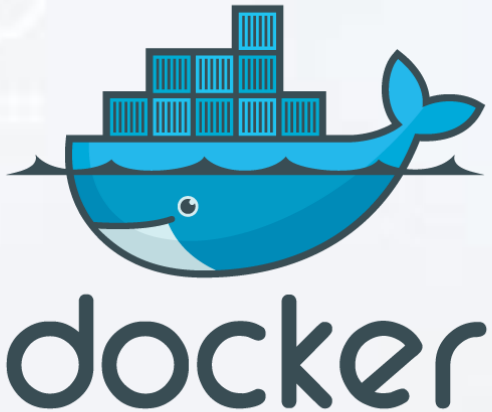
Ram N Sangwan

Agenda

- What is Docker?
- Use case of Docker
- Platforms for Docker
- Containers Vs VM
- Docker Architecture
- Docker Components

What is Docker?

- Docker is "***a platform for developers and sysadmins to develop, ship, and run applications***", based on containers.
- Docker is open-source, created in Go and originally on top of ***libvirt***.
- Docker simplifies and standardizes the creation and management of containers.



Use Cases for Developers

- Build once, run anywhere
  - A clean, safe, hygienic and portable runtime environment for your app.
  - No worries about missing dependencies, packages and other pain points during subsequent deployments.
  - Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying.
  - Automate testing, integration, packaging…anything you can script.
  - Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.

Use Cases for Devops

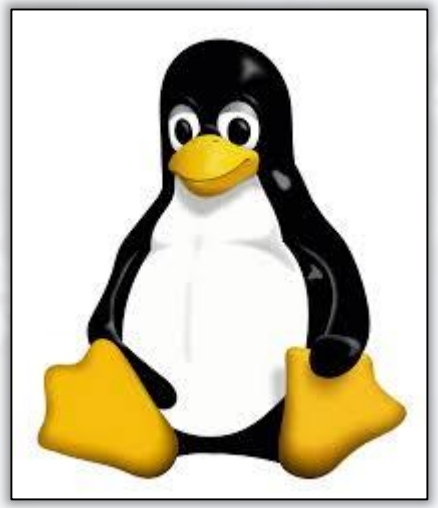
- Configure once, run anything
  - Make the entire lifecycle more efficient, consistent, and repeatable.
  - Increase the quality of code produced by developers.
  - Eliminate inconsistencies between development, test, production, and customer environments.
  - Support segregation of duties.
  - Significantly improves the speed and reliability of continuous deployment and continuous integration systems.
  - Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs.

Platforms for Docker

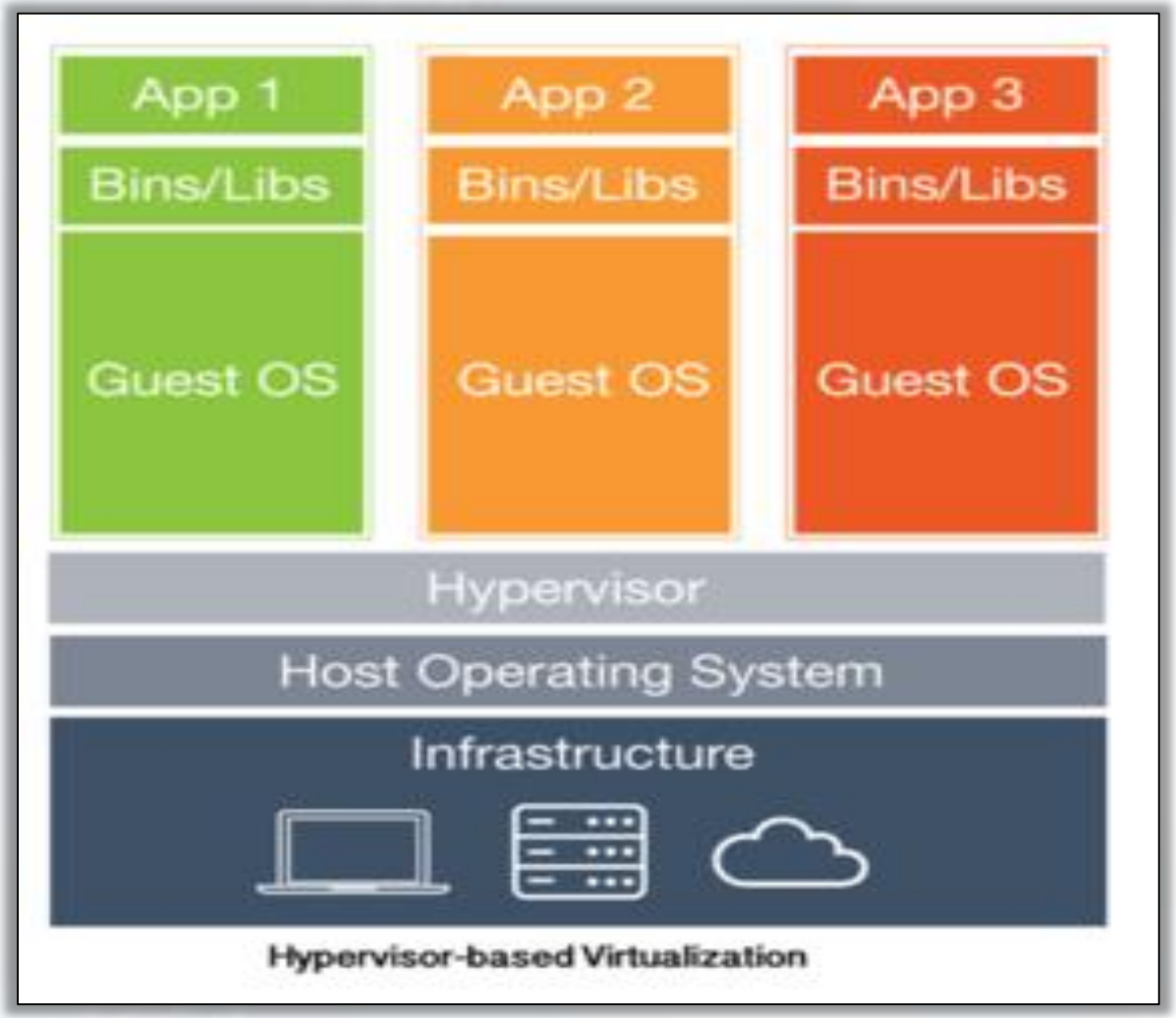
- Docker Engine is available on a variety of Linux platforms, macOS and Windows 10 through Docker Desktop, and as a static binary installation.
- Desktop
  - Docker Desktop for Mac (macOS)
  - Docker Desktop for Windows
- Server
  - CentOS
  - Debian
  - Fedora
  - Raspbian
  - Ubuntu

Linux Containers

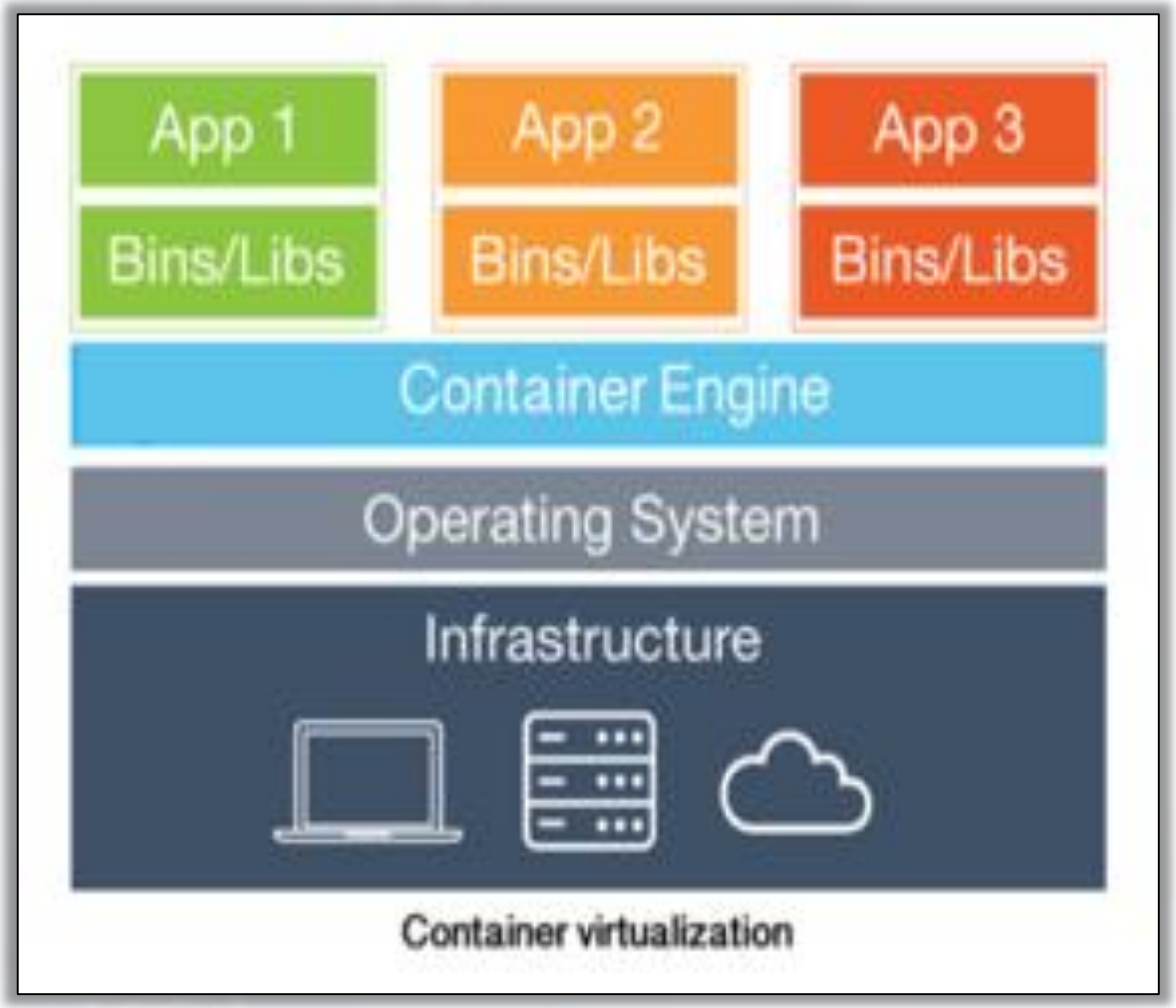
- An operating system–level virtualization method for running multiple isolated Linux systems (containers) on a single control host.



VMs


App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure
Hypervisor-based Virtualization

Containers

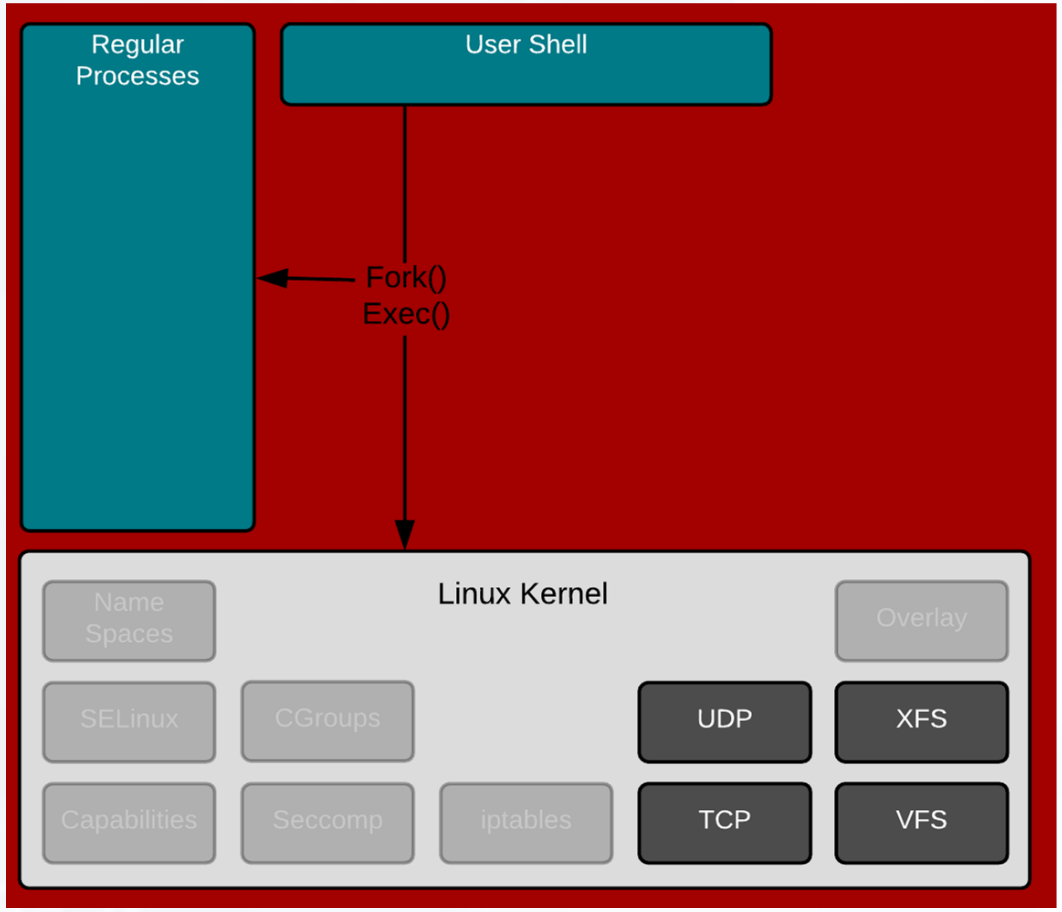

App 1
App 2
App 3
Bins/Libs
Bins/Libs
Bins/Libs
Container Engine
Operating System
Infrastructure
Container virtualization

Containers Vs VMs

	VMs	Containers
Security	More isolated	Less isolated
Size	GBs	MBs
Provision	Mins	Secs
OS	More flexible	Less flexible

Kernel and Containers

- Normal processes are created, destroyed, and managed with system calls:
  - Fork() - Think Apache
  - Exec() - Think ps
  - Exit()
  - Kill()
  - Open()
  - Close()
  - System()

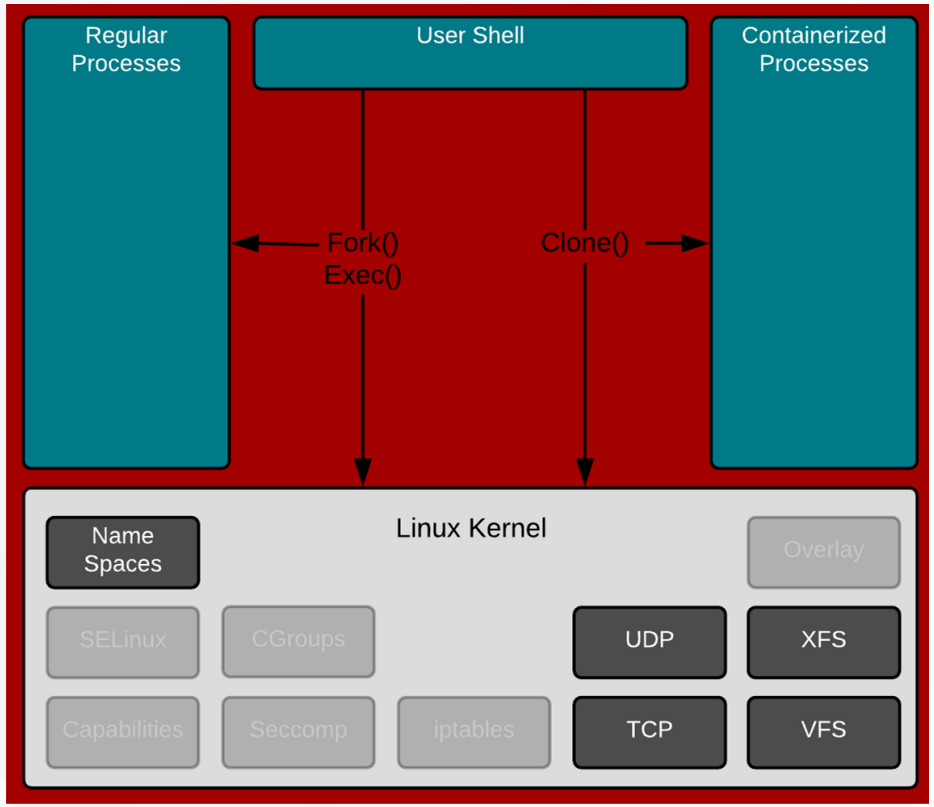


Kernel and Containers

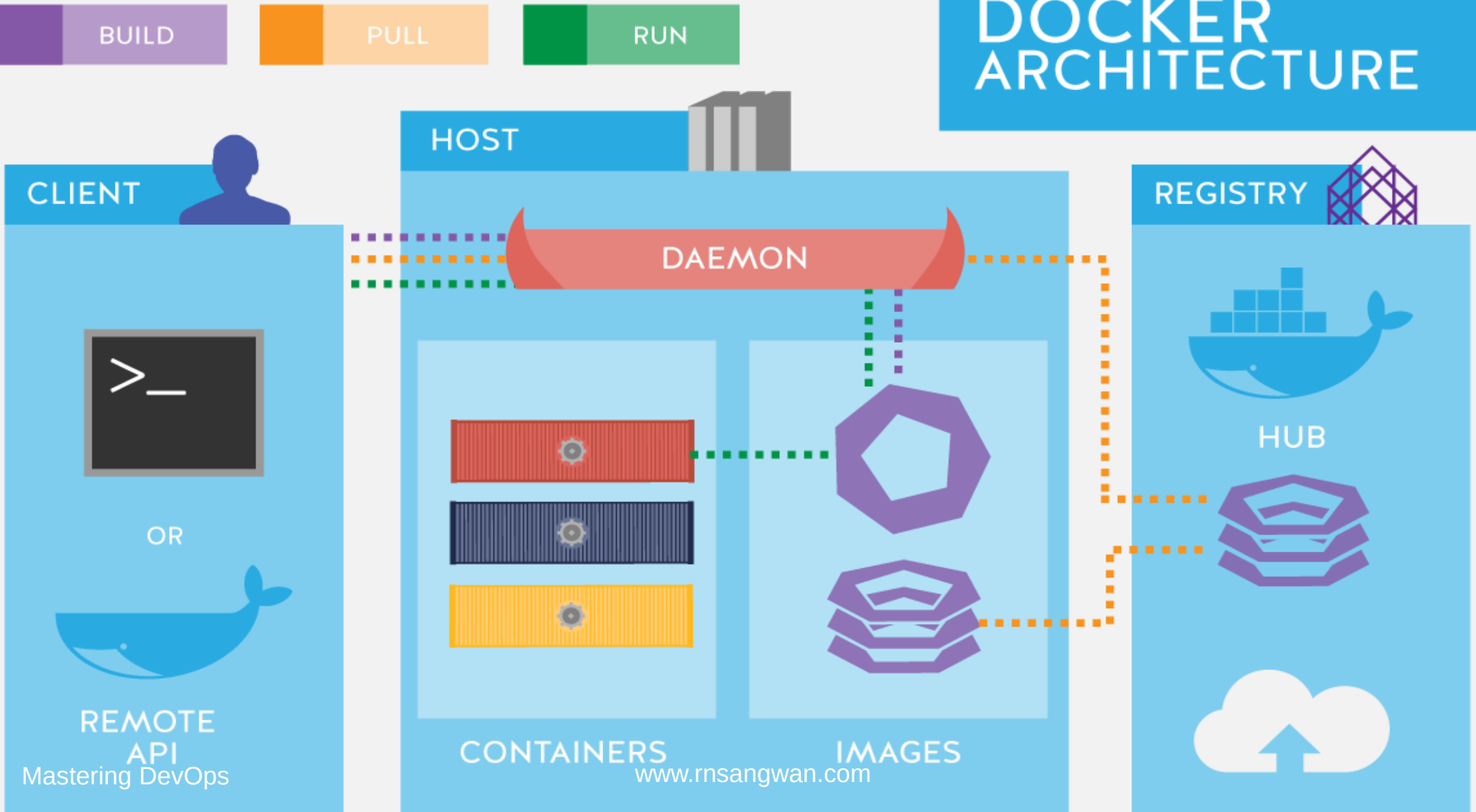
Creating “containerized” Linux processes

What is a container anyway?

- No kernel definition for what a container is - only processes
- Clone() - closest we have
- Creates namespaces for kernel resources
  - Mount, UTC, IPC, PID, Network, User
- Essentially, virtualized data structures



DOCKER ARCHITECTURE



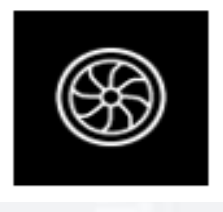
Docker Components

Docker Image

The basis of a Docker container. Represents a full application

Docker Container

The standard unit in which the application service resides and executes



Docker Engine

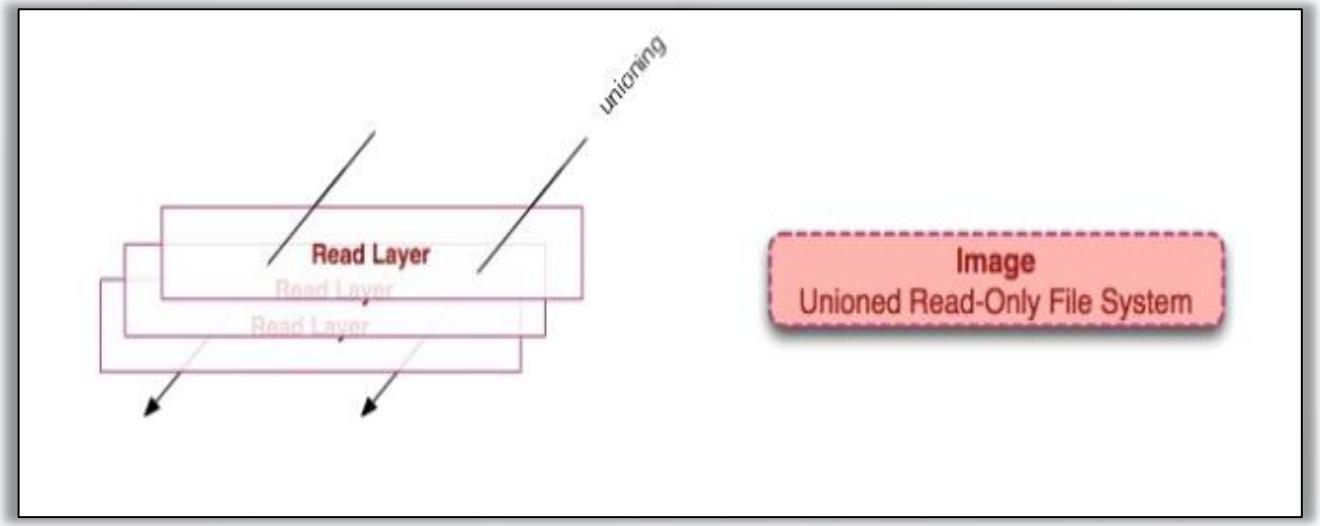
Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider

Registry Service (Docker Hub or Docker Trusted Registry)

Cloud or server based storage and distribution service for your images

Images

- Read only template used to create containers
- Build by you or other ***docker*** users
- Stored in the ***docker hub*** or your local registry
- Every image starts from base image
- Include:
  - Application
  - Dependencies
  - Libraries
  - Binaries
  - Configuration files



Containers

- Isolated application platform
- Contains everything needed to run you application
- Based on one or more images
- Docker containers launched from Docker image
- When Docker container runs, it adds a read-write layer on top of the image

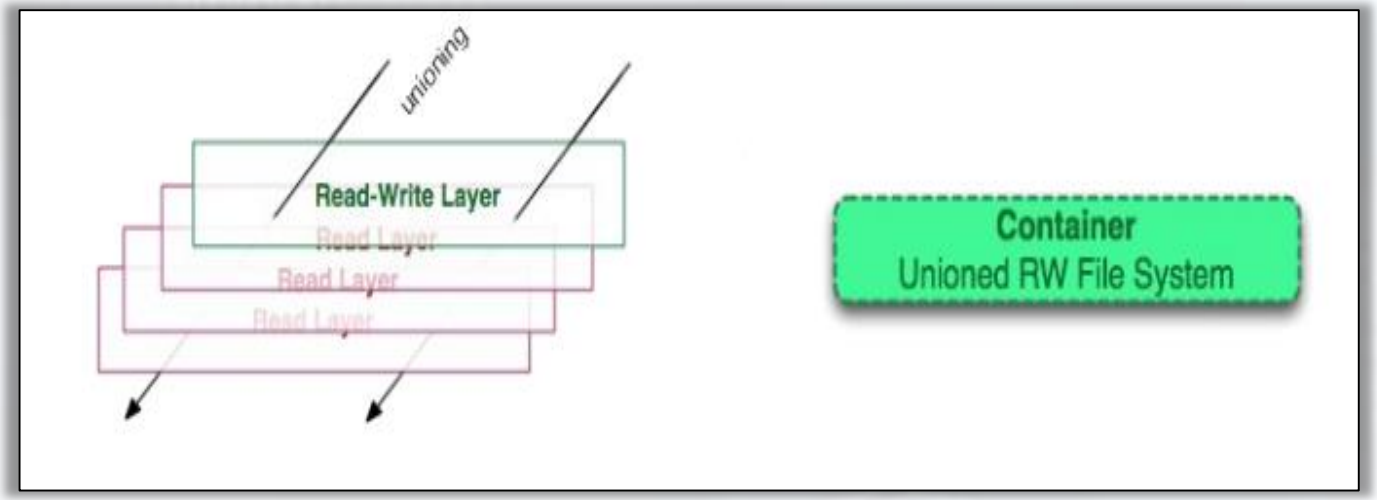
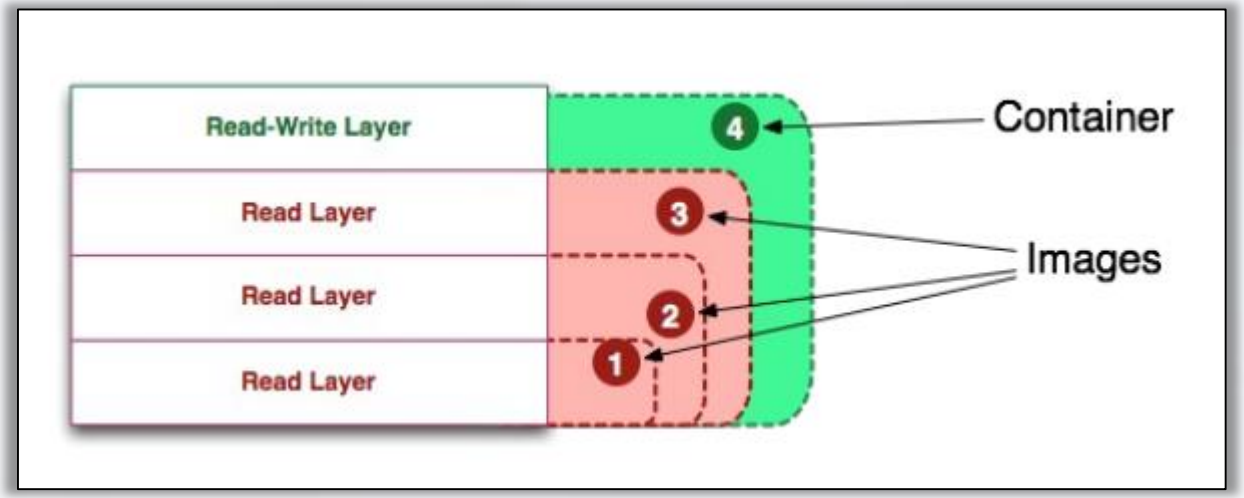
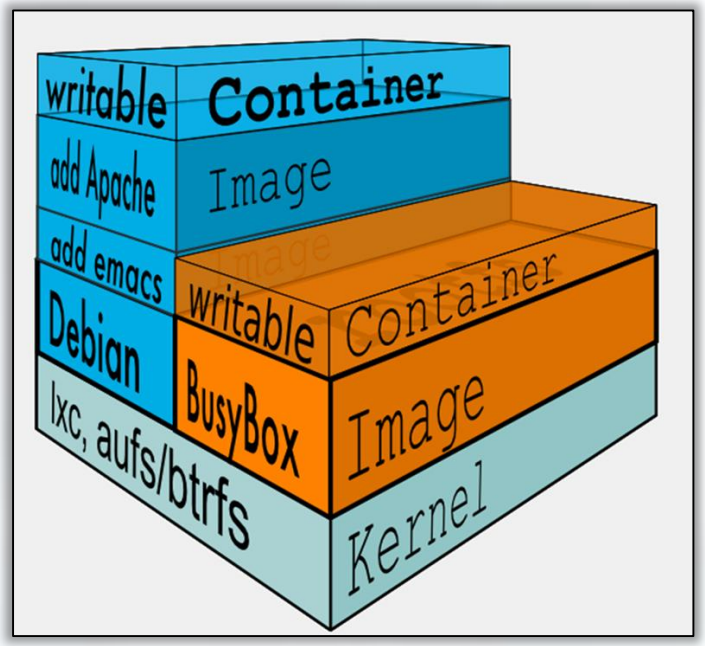


Image vs. Container

- Docker Image is a class
- Docker Container is a instance of class



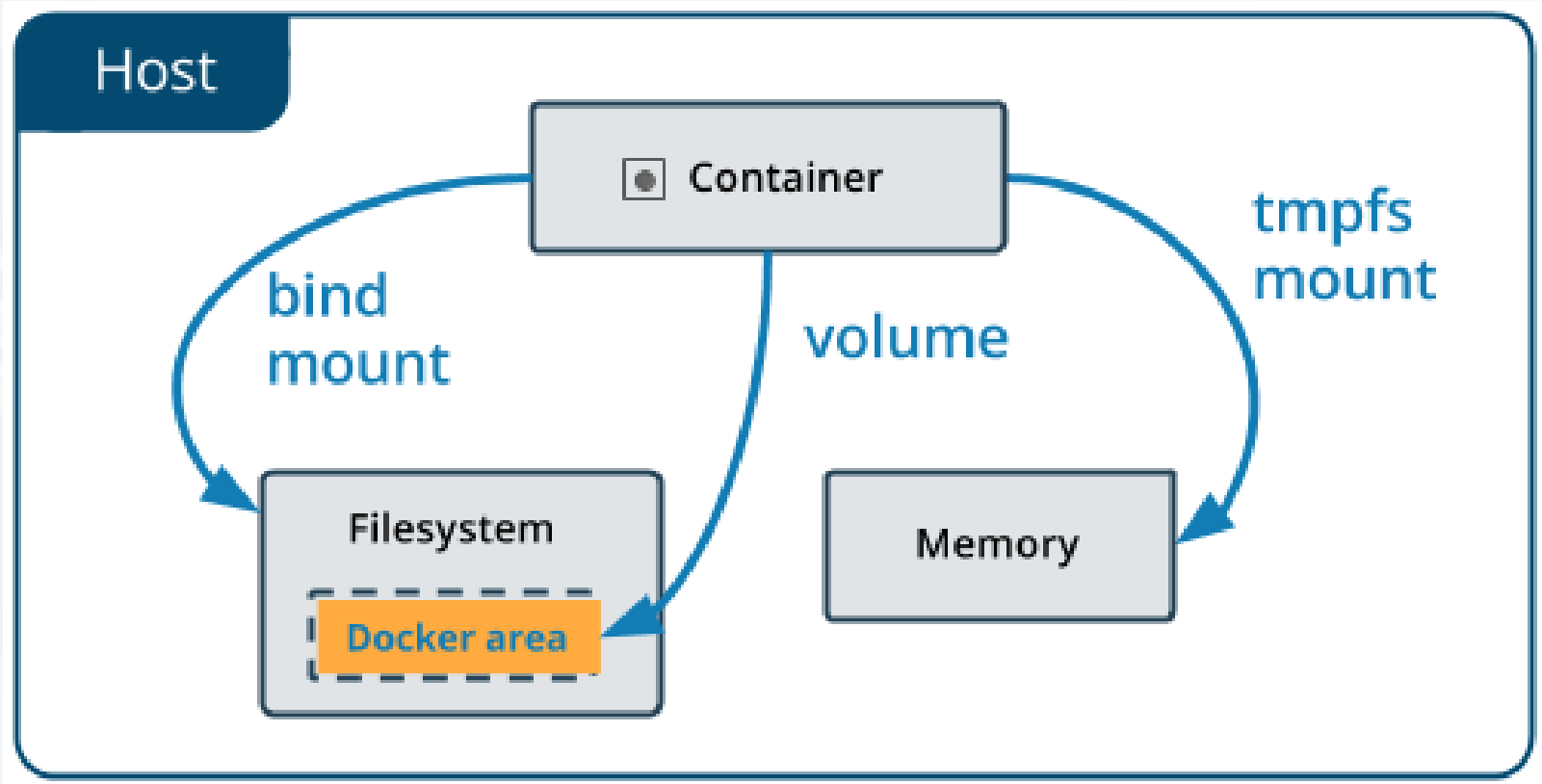


Docker Volumes

- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.
- While ***bind mounts*** are dependent on the directory structure of the host machine, volumes are completely managed by Docker.
- Volumes have several advantages over bind mounts:
  - Volumes are easier to back up or migrate than bind mounts.
  - You can manage volumes using Docker CLI commands or the Docker API.
  - Volumes work on both Linux and Windows containers.
  - Volumes can be more safely shared among multiple containers.
  - Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
  - New volumes can have their content pre-populated by a container.

Docker Volumes

- Volumes are often a better choice than persisting data in a container's writable layer, because a volume does not increase the size of the containers using it, and the volume's contents exist outside the lifecycle of a given container.



Thank You