Operating System Virtualization
UNIT-V-Part-I

Docker and Containers

Motivation

- VMware, Inc. gave the world a gift, the *virtual machine (VM)* — a technology that allowed us to run multiple business applications on a single server safely.
- *But!* As great as VMs are, they're far from perfect.
- For example, every VM needs its own dedicated OS.
- Unfortunately, this has several drawbacks, including:
  - Every OS consumes CPU, RAM, and other resources
  - Every VM and OS needs patching
  - Every VM and OS needs monitoring
- VMs are also slow to boot and not very portable.

Motivation

- While most were using VMs, ***Google*** had already moved on from VMs and were using containers.
- Containers share the host OS, allowing a single machine to run far more containers than VMs (e.g., 50 vs 10), making them more efficient.
- They are also faster and more portable, especially in Linux environments.

History

- Google started using container-like technology **around 2003–2004**
  - Internal system called Borg
  - Used for Search, Gmail, YouTube infrastructure
- This was **before Docker even existed**
- **2006–2013 (Mature internal use)**
- Borg evolved into:
  - **Omega** (next-gen scheduler)
- Google was running:
  - **billions of containers per week**
- Containers were already core to Google's infrastructure
- **2013 Docker** released
- **2014** Google launched:
  - **Kubernetes**
- Kubernetes was inspired by **Borg**

Motivation

- Some of the major technologies behind modern containers include *kernel namespaces*, *control groups (cgroups)*, and *capabilities*.
- However, despite all this, containers were incredibly complicated, and it wasn't until Docker came along that they became accessible to the masses.

Containers

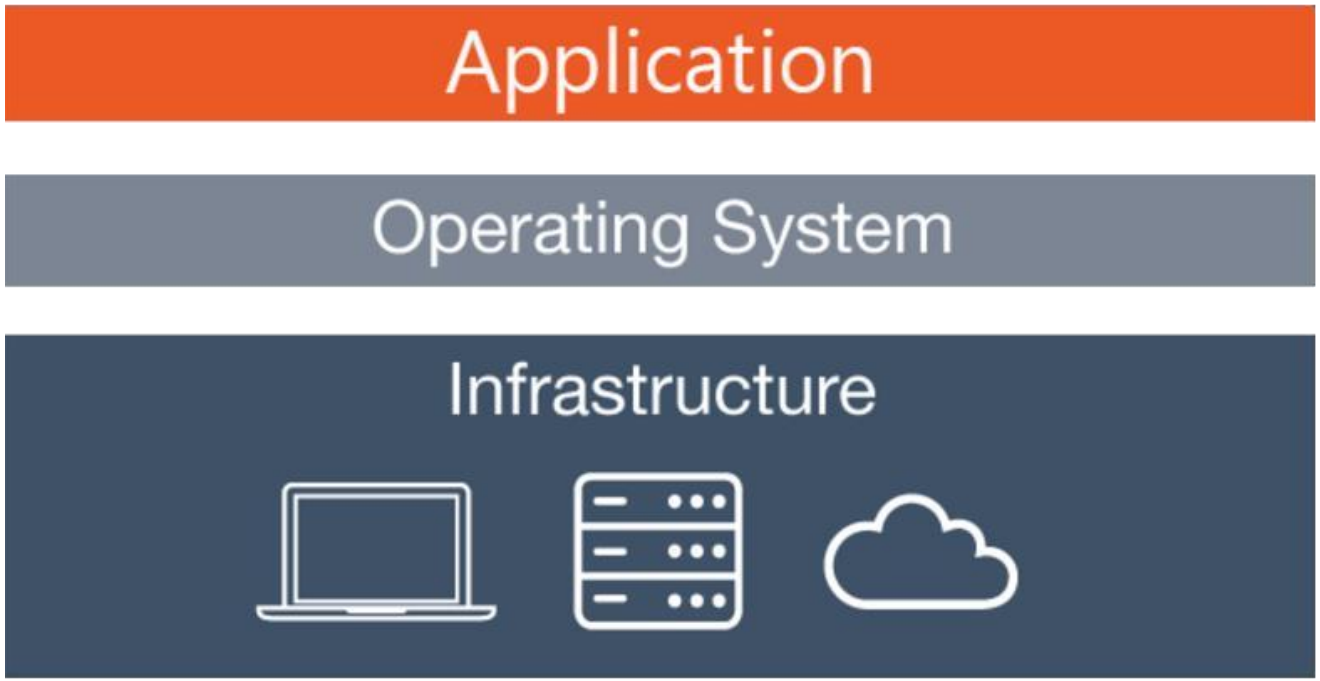
- Windows desktop and server platforms support both of the following:
  - Windows containers
  - Linux containers
- *Windows containers* run Windows apps and require a host system with a Windows kernel.
- Windows 10, Windows 11, and all modern versions of Windows Server natively support Windows containers.
- Windows systems can also run Linux containers via the *WSL 2 (Windows Subsystem for Linux)*.
- ***A containerized app is an application running as a container.***

Kubernetes

- **Kubernetes** is the industry standard platform for *deploying and managing* containerized apps.
- Older versions of Kubernetes used Docker to start and stop containers.
- Newer container platforms use *containerd*, which is a lightweight container runtime originally extracted from Docker and now used independently by Kubernetes and other systems.
- *The important thing to know is that all Docker containers work on Kubernetes.*

https://training.play-with-docker.com/beginner-linux/

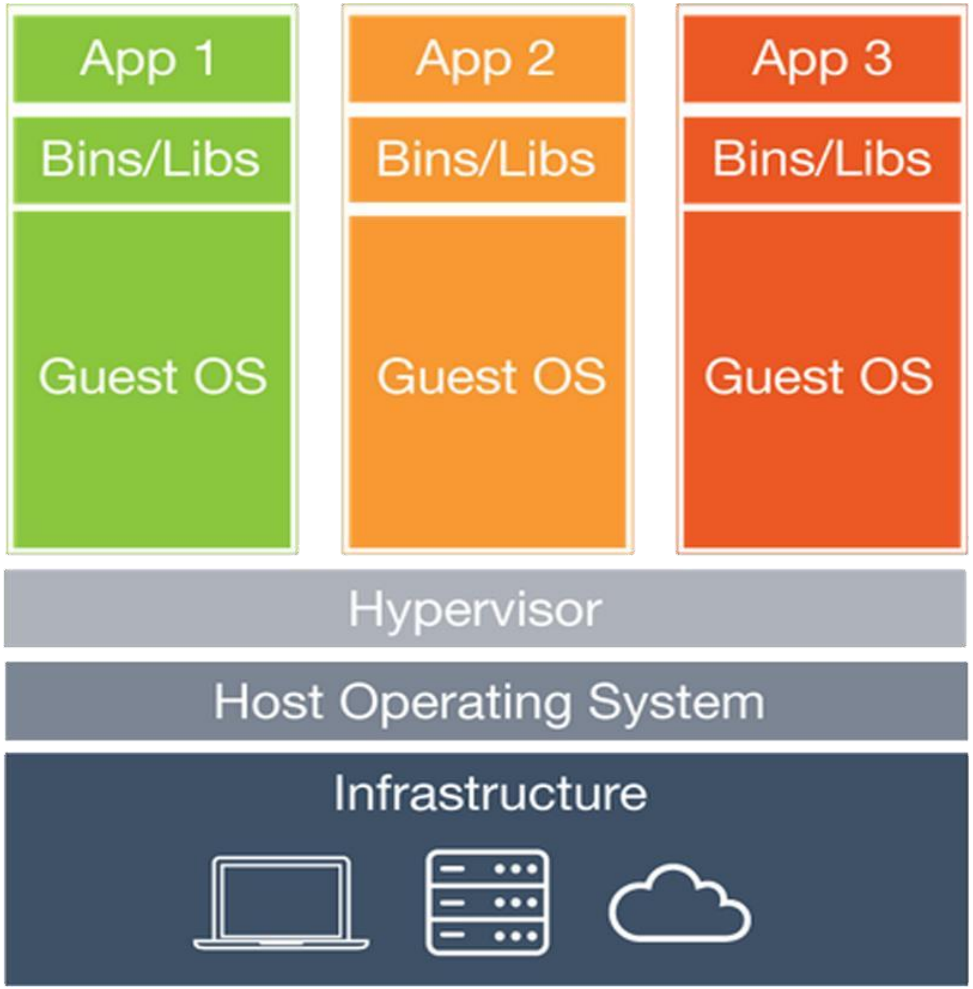
Old Way



Virtualization

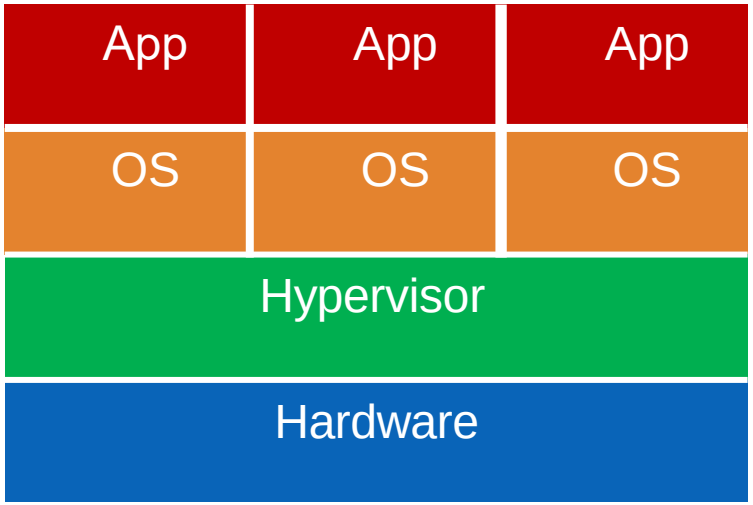
- Virtualization enables abstraction of hardware resources and slices a server up into multiple virtual machines, each which mimics a complete physical server

Virtualization



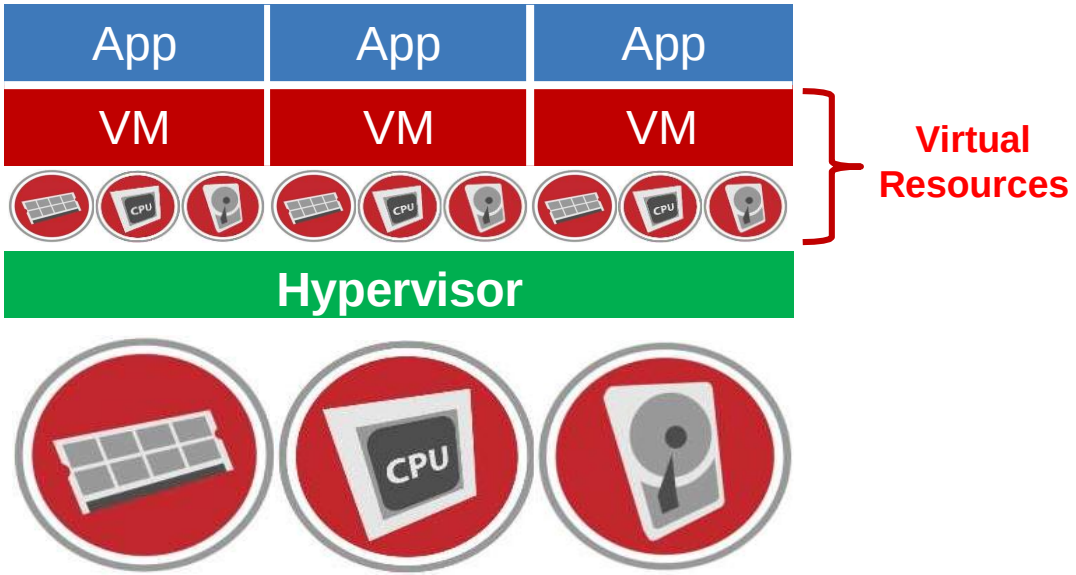
Virtualization Challenges

- Significant overhead
- Significant repetitive work
  - OS deployments
  - System configurations
  - Challenging at scale



Virtualization Challenges

- Every operating system needs individual configuration
- Getting things just right takes time



What is OS?

- OS virtualization is a technique that allows multiple **isolated user-space instances (containers)** to run on a ***single operating system kernel***.
  - Also called container-based virtualization
  - Containers share the same OS kernel
  - Each container runs independently like a separate system
  - No need for separate guest operating systems

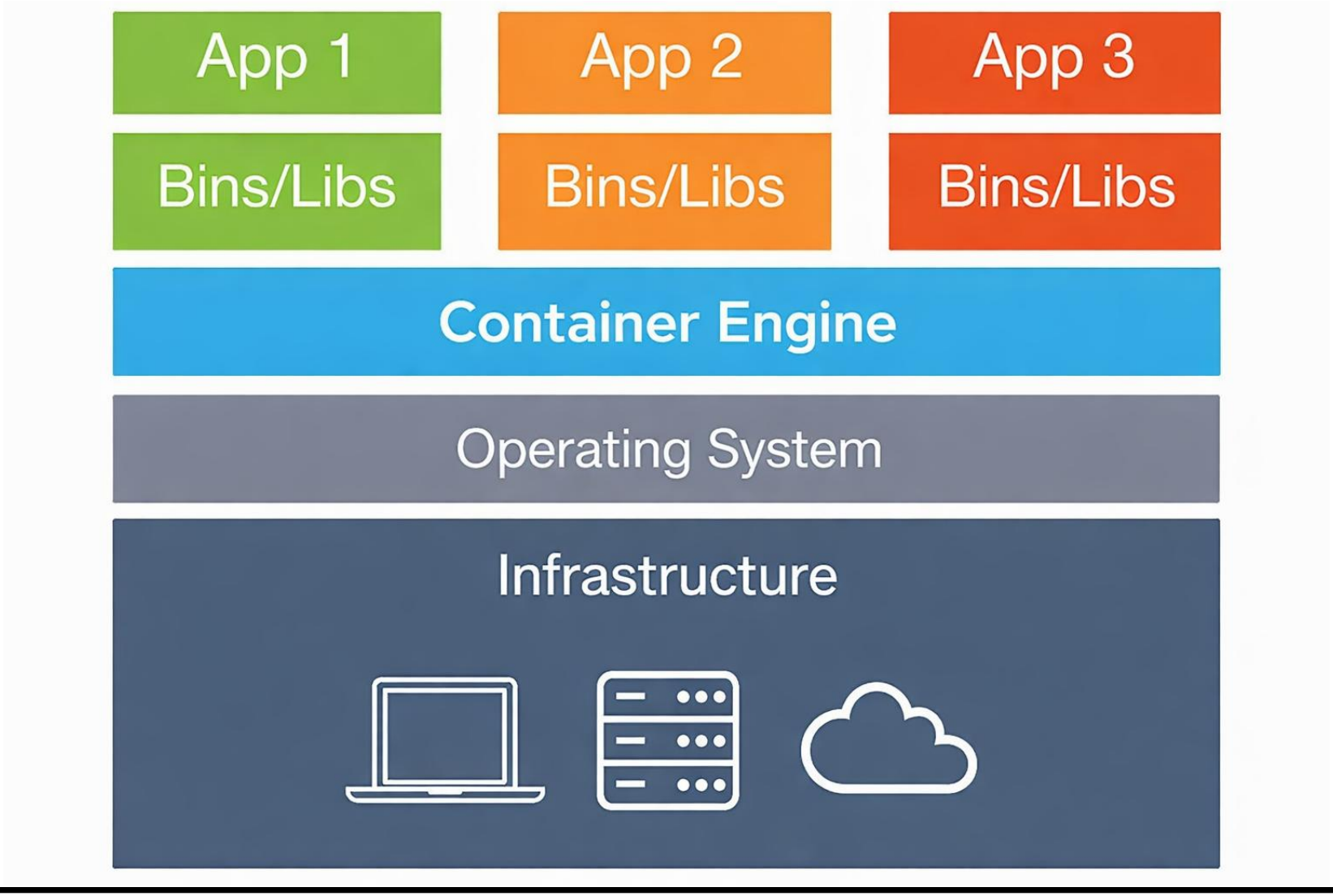
How does it work?

- Uses **host operating system kernel**
- Creates isolation using:
  - **Namespaces** that isolate processes, network, files
  - **Control Groups (cgroups)** that limit CPU, memory
- Each container runs:
  - Its own applications
  - Its own file system view
- Example Tools:
  - Docker
  - LXC
  - Containerd

Containers

- 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.

Container Based Virtualization

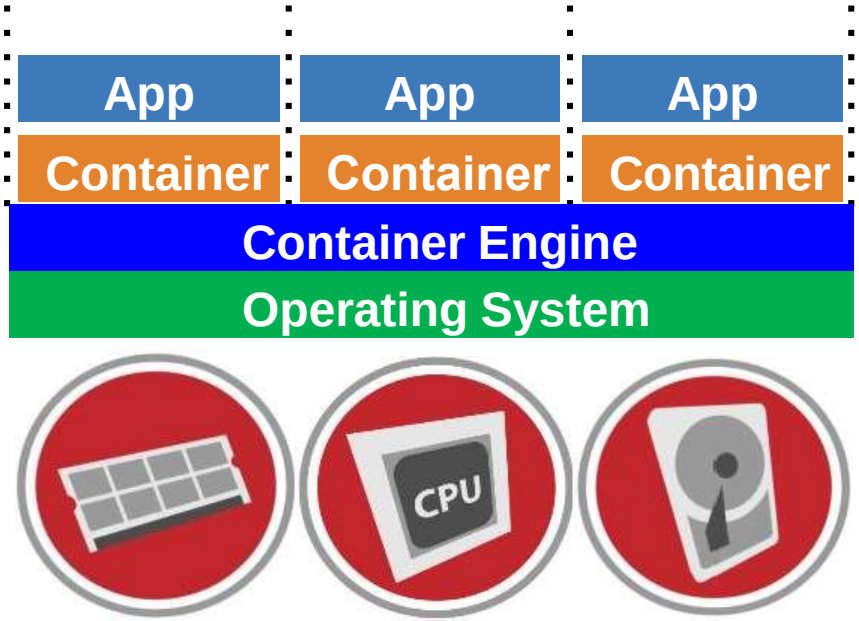


Containers

- Containers enable abstraction of resources at the *operating system* level, enabling multiple applications to share binaries while remaining isolated from one another
- The VM's does this at *hardware level*

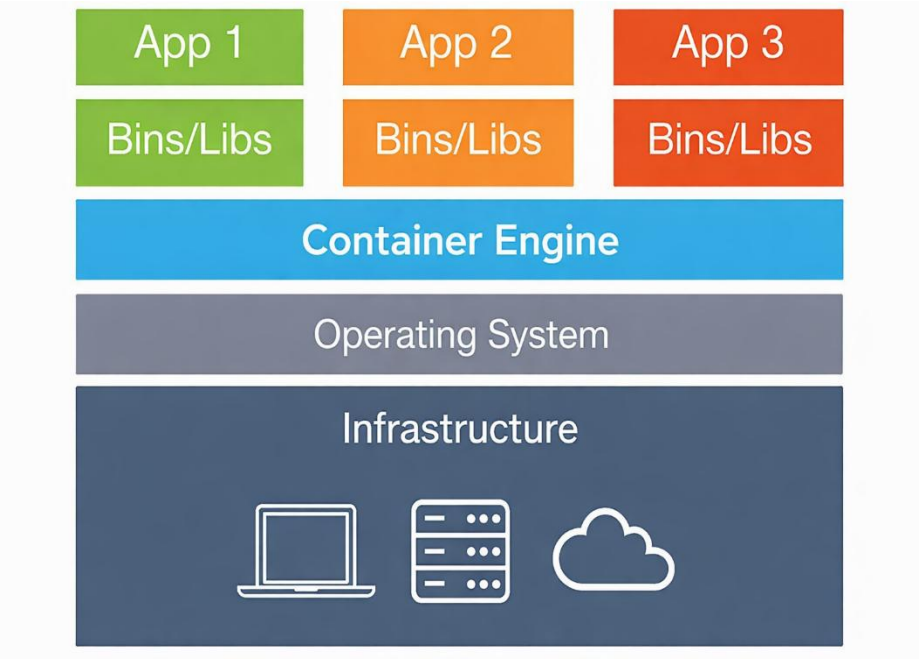
Container Benefits

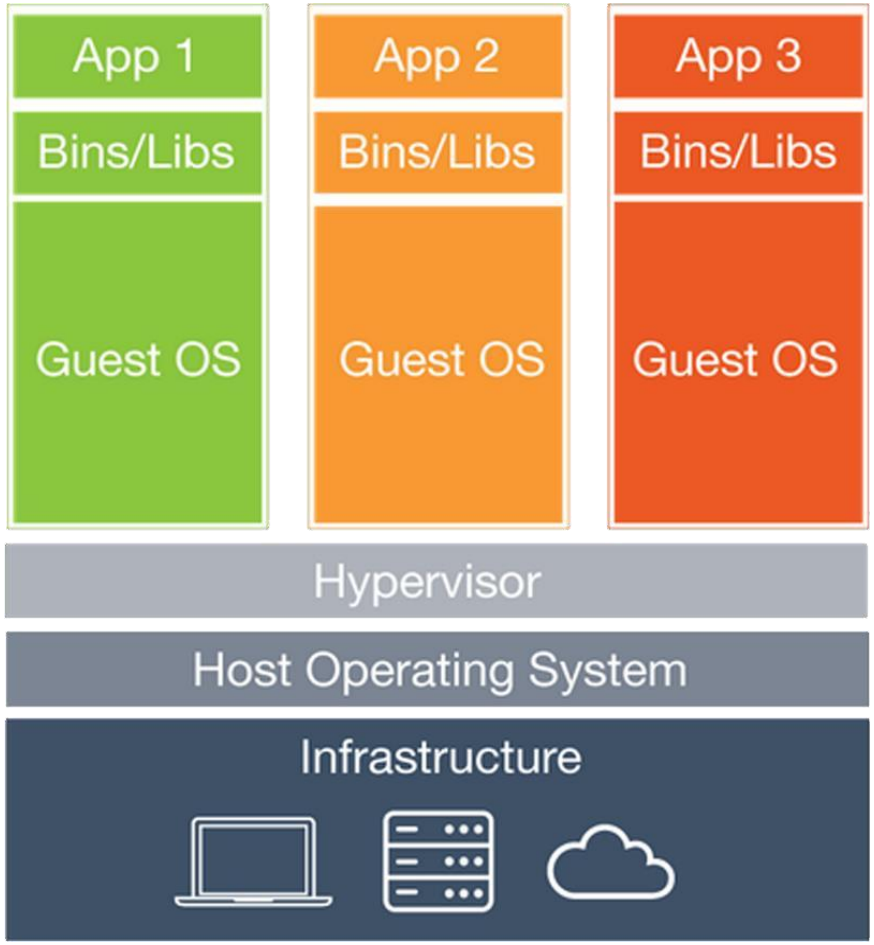
- Far less resource overhead
- Fewer operating systems to support
- Simpler application deployment
- Improved provisioning efficiency



- Example of containers: Docker, Linux container, Podman

Container vs VM





Minimum dependency or base system to create logical isolation from one to another container

What is Docker?

- *Docker* is a platform for developing, shipping & running distributed applications using container based virtualization technology.
- The word *Docker* is a British expression short for dock worker referring to someone who loads and unloads cargo from ships.

Docker Containers

- 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.

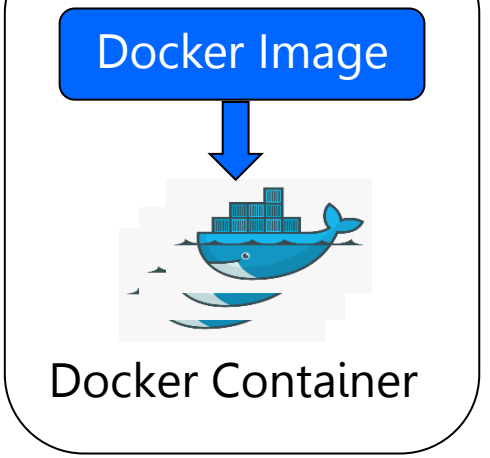
How is this done?

- It does this by virtualizing the *operating system of the computer* on which it is installed and running.
- The first edition of Docker was released in 2013.
- Docker is developed by Docker, Inc., written in Go programming language, and is open source.

Workflow of Docker

https://hub.docker.com/

Pull Image

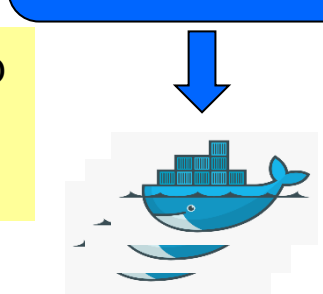


Developer defines application and its dependencies in a file known as **Dockerfile**

Images can also be stored in Docker Hub

Dockerfile

This Dockerfile is used to create Docker images using **build**



Running images give Docker Container, it is a runtime instance of the image

Container vs VM

- Containers are more lightweight.
- No need to install guest OS (only with Linux).
- Otherwise, it has very light weight OS with small functionality.
- Less CPU, RAM, Storage needed.
- More containers per machine than VM.

Container vs VM

Docker containers that run on Docker Engine are:

- **Standard:** Defined standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application
- **Secure:** Docker provides the strongest default isolation capabilities in the industry

How Isolation

It provides minimum dependency or base system to create logical isolation from one to another container (these are not part of the shared Kernel but are with each container)

/bin: Essential command binaries needed for execution.
/lib or /lib64: Core shared libraries required by the binaries.
/etc: Configuration files for system settings and applications.
/dev: Device files, which are isolated by container runtimes.

How Isolation

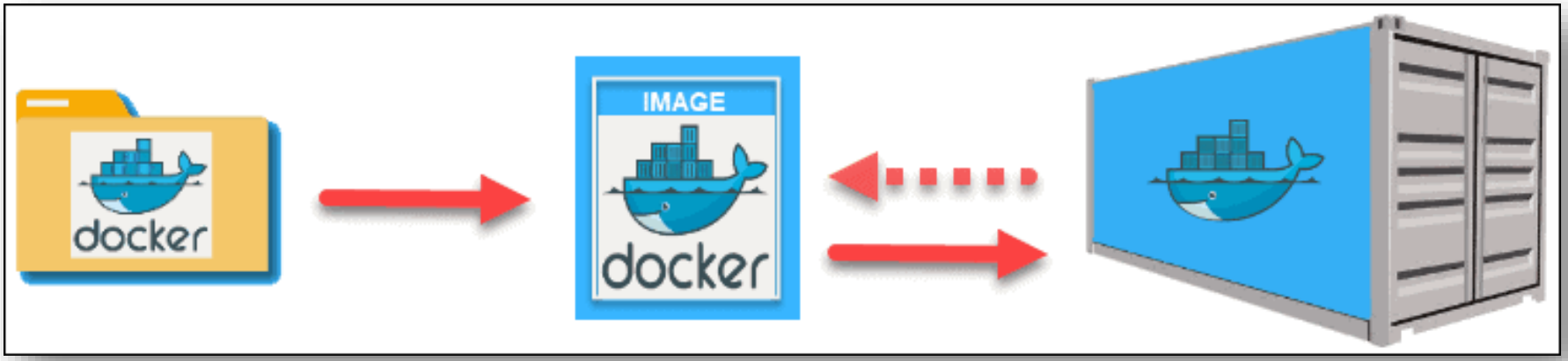
/proc and /sys: Virtual filesystems that provide process and system information, usually mounted at runtime.
/usr: (Optional but common) User utilities and additional applications.

For Ubuntu

- VM size is around 2.3GB
- Container size 22MB!

Docker Container vs Image

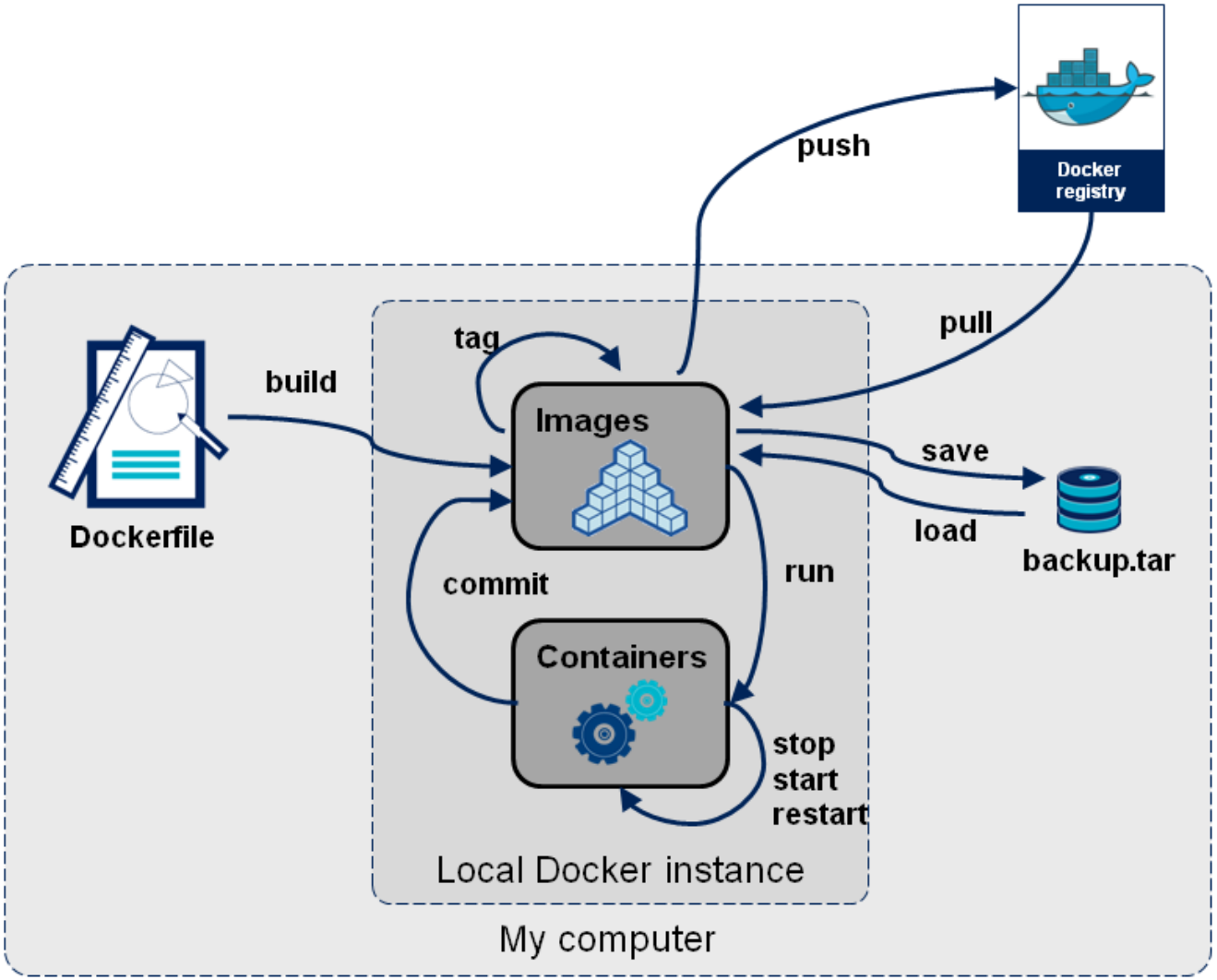
- When using Docker, we start with a base image. We boot it up, create changes and those changes are saved in layers forming another image.



Docker Container vs Image

- An instance of an image is called a container.
- We have an image, which is a **set of layers** as we describe.
- If we start this image, we have a running container of this image and can run on any computer having Docker.
- We can have many running containers of the same image.

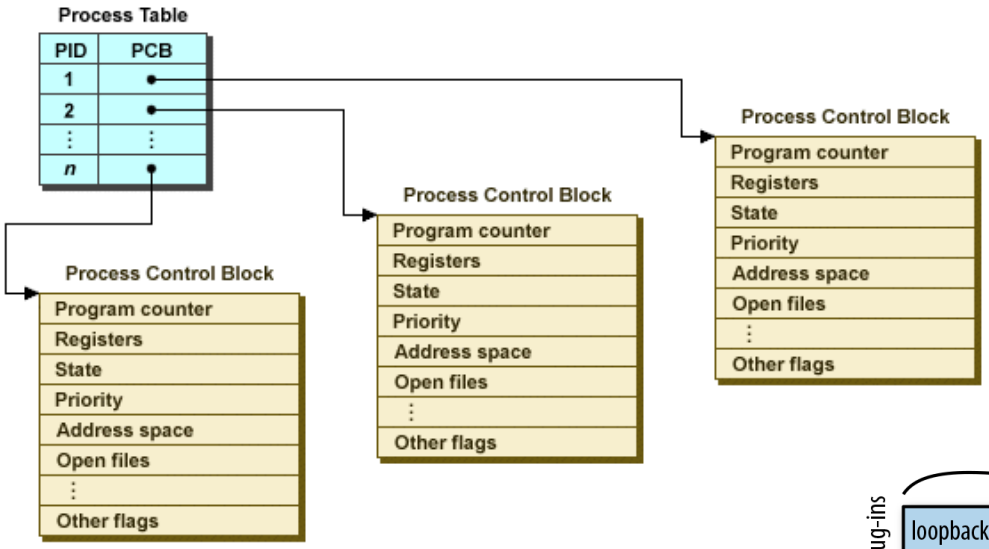
Docker Container vs Image



What will it have?

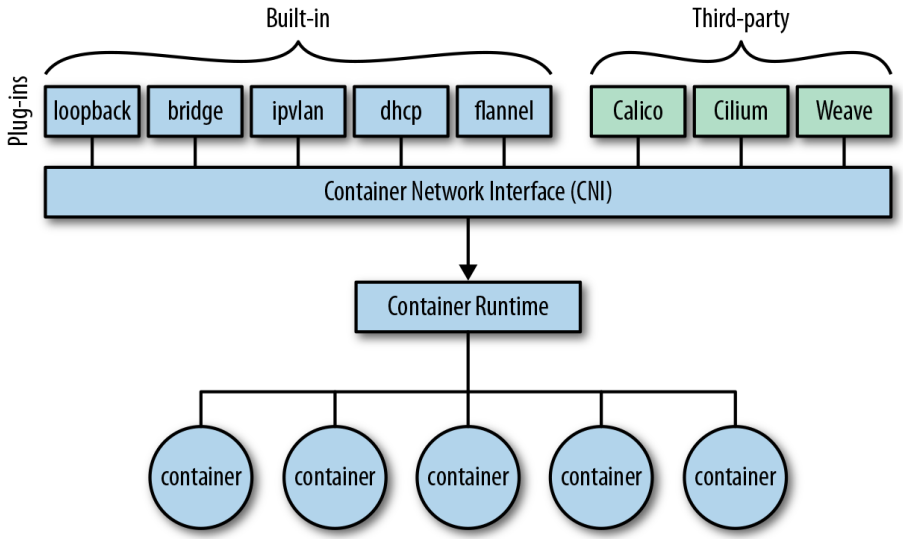
- A virtual copy of the process table, network interface(s), and the file system mount point(s).
- These are inherited from the OS of the host on which the container is hosted and running.

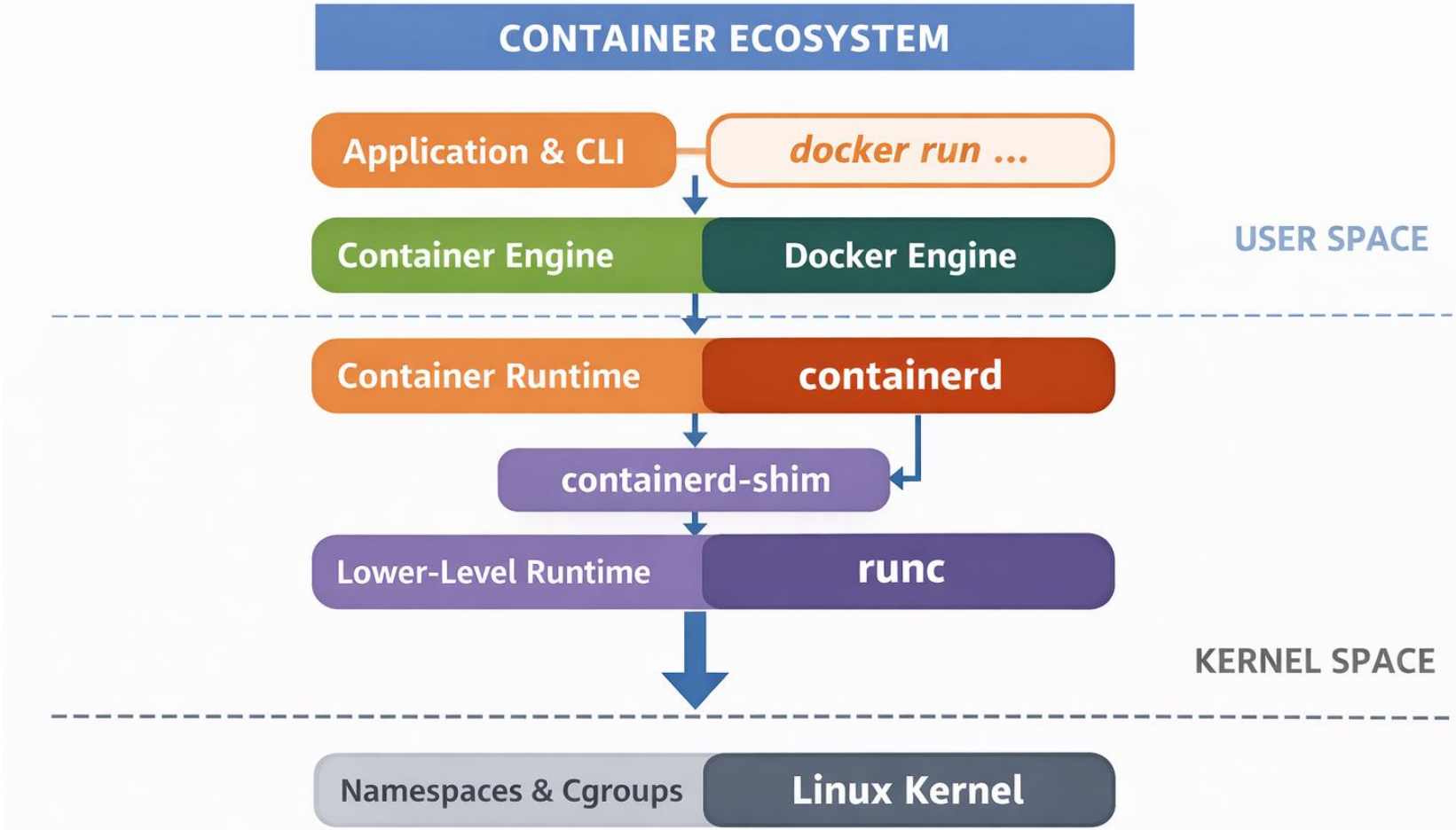
What will it have?



process table &

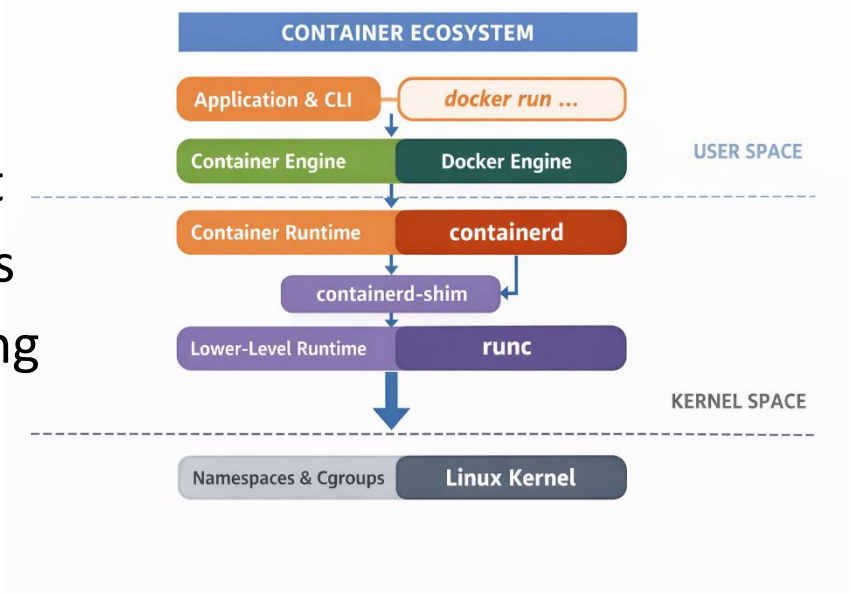
network interface(s)




CONTAINER ECOSYSTEM
Application & CLI
docker run ...
Container Engine
Docker Engine
USER SPACE
Container Runtime
containerd
containerd-shim
Lower-Level Runtime
runc
KERNEL SPACE
Namespaces & Cgroups
Linux Kernel

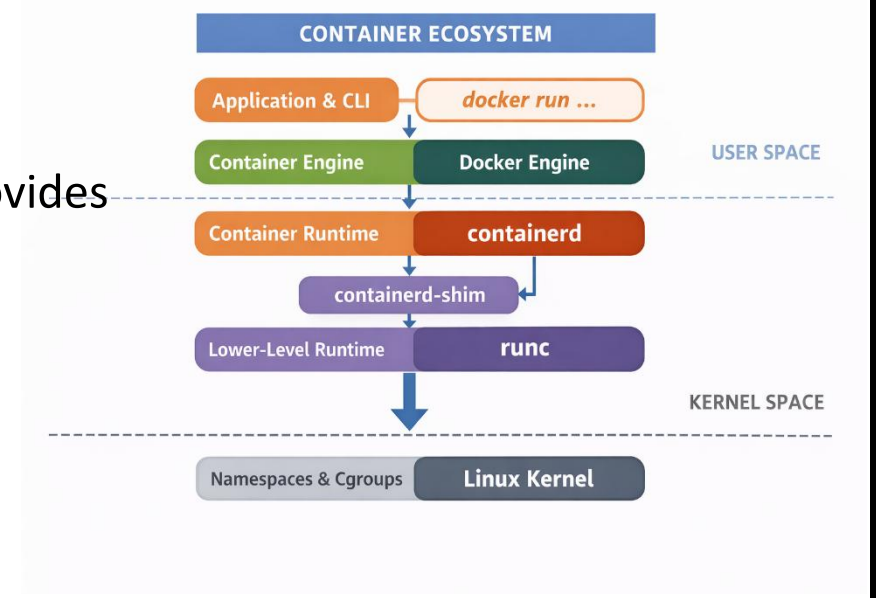
Terminology

- **Docker (Docker Engine)**
  A platform that provides tools to build, manage, and run containers.
- **containerd**
  A high-level container runtime that manages container lifecycle such as pulling images and starting/stopping containers.
- **containerd-shim**
  A lightweight process that allows containers to run independently of containerd and manages their I/O and lifecycle.



Terminology

- **runc**
  A low-level runtime that creates and runs containers using operating system kernel features.
- **Linux Kernel**
  The core of the operating system that provides isolation and resource control using namespaces and cgroups.
- **Namespaces**
  Kernel features that provide isolation of processes, network, and filesystem for containers.
- **cgroups (Control Groups)**
  Kernel features that limit and manage resource usage like CPU, memory, and I/O for containers.



What will it be?

- Each container to be isolated from the other present on the same host.
- Thus it supports multiple containers with different application requirements and dependencies to run on the same host, as long as they have the same operating system requirements.

Advantages of using Docker?

- A container doesn't include full OS within each instance.
- It consumes very little memory in comparison to a VM.
- This also reduces the bootup time
- Reduces costs: Docker is less demanding for the hardware required to run it.

Disadvantages of using Docker?

1. *Applications with different operating system requirements cannot be hosted together on the same Docker Host.*
2. *Windows Subsystem for Linux (WSL) is required for running Linux Docker containers on Windows 11 Home.*

Why WSL 2

- Docker containers rely on Linux kernel features such as namespaces and cgroups.
- While Linux provides native support, Windows now supports containers either through Windows-native container features or via WSL 2 for Linux containers.

namespaces provide isolation for containers by creating separate environments, while cgroups manage and limit resource usage (like CPU, memory, and network) for those containers.

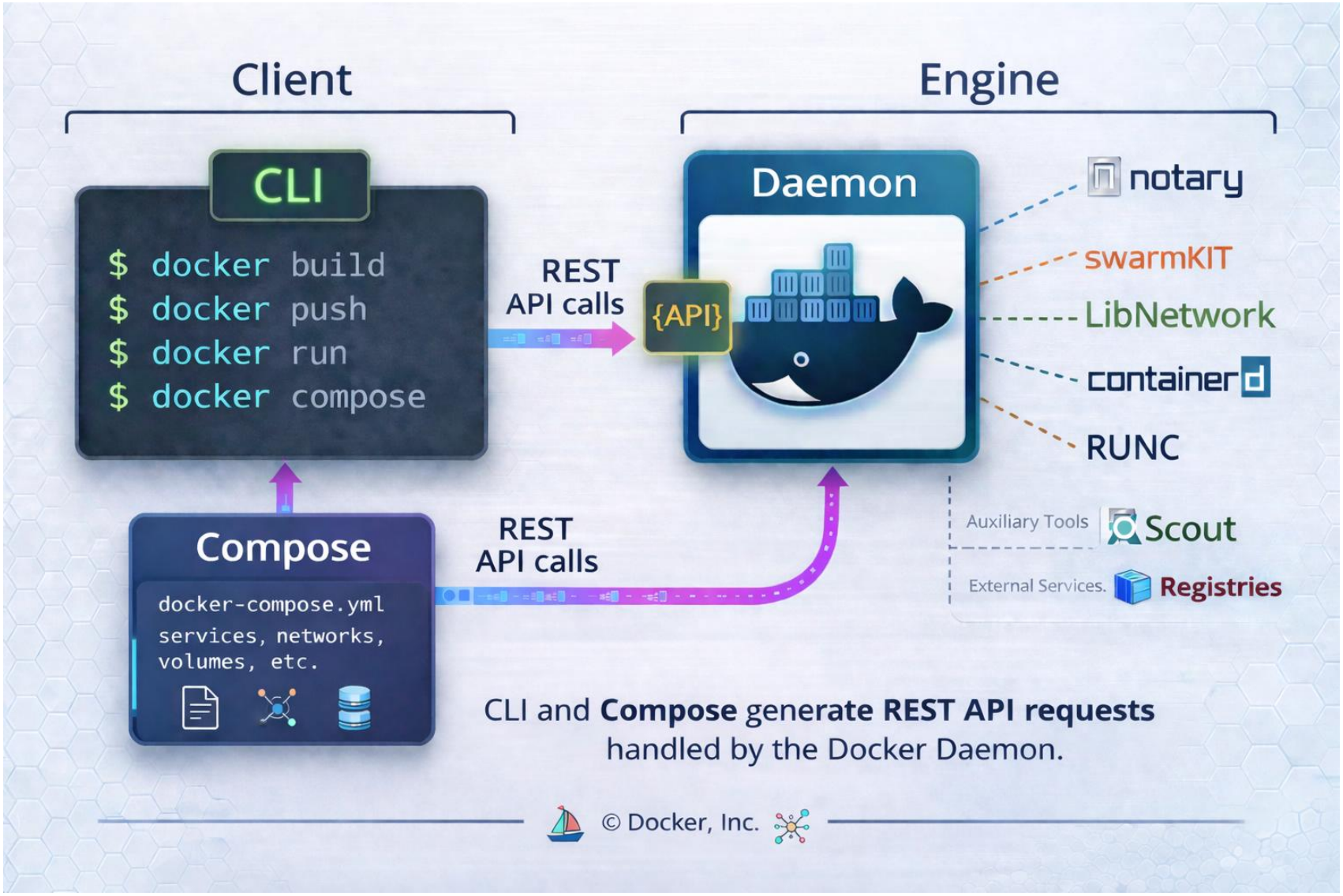
Why WSL 2

- Improved Performance and Compatibility:
  Compared to older virtualization methods (like Hyper-V), WSL 2 offers better I/O performance and a more seamless integration with Windows, making Docker operations faster and more efficient.
- WSL 2 Provides a Native Linux Kernel:
  WSL 2 includes a lightweight VM with a full Linux kernel. This allows Docker Desktop to run Linux containers on Windows as if they were running on a native Linux system.

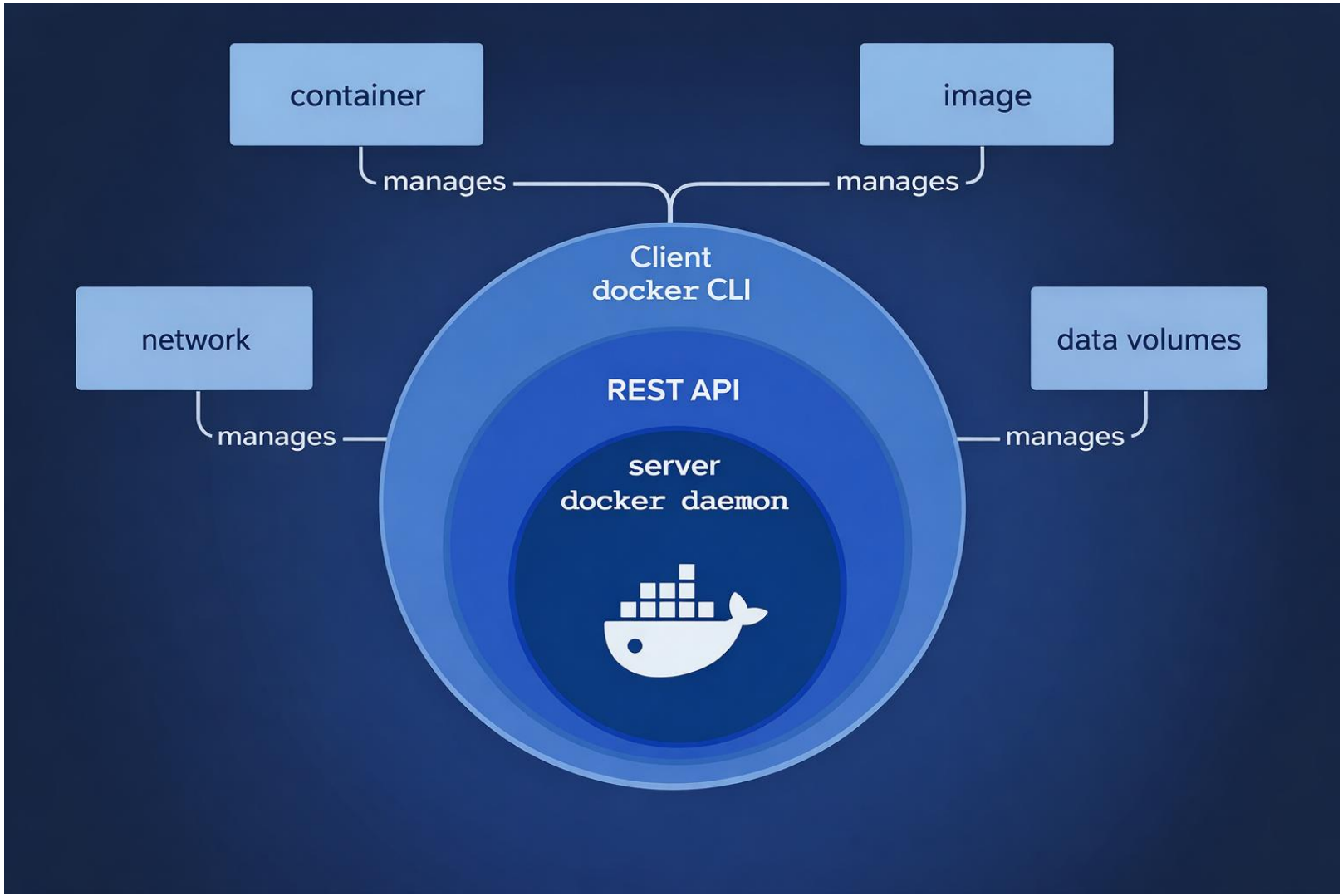
Docker Engine

- Docker Engine is a client-server application with these major components:
  - A server which is a type of long-running program called a daemon process
  - A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do
  - A command line interface (CLI) client.

Interaction



Docker Engine



https://docs.docker.com/engine/docker-overview/#docker-engine

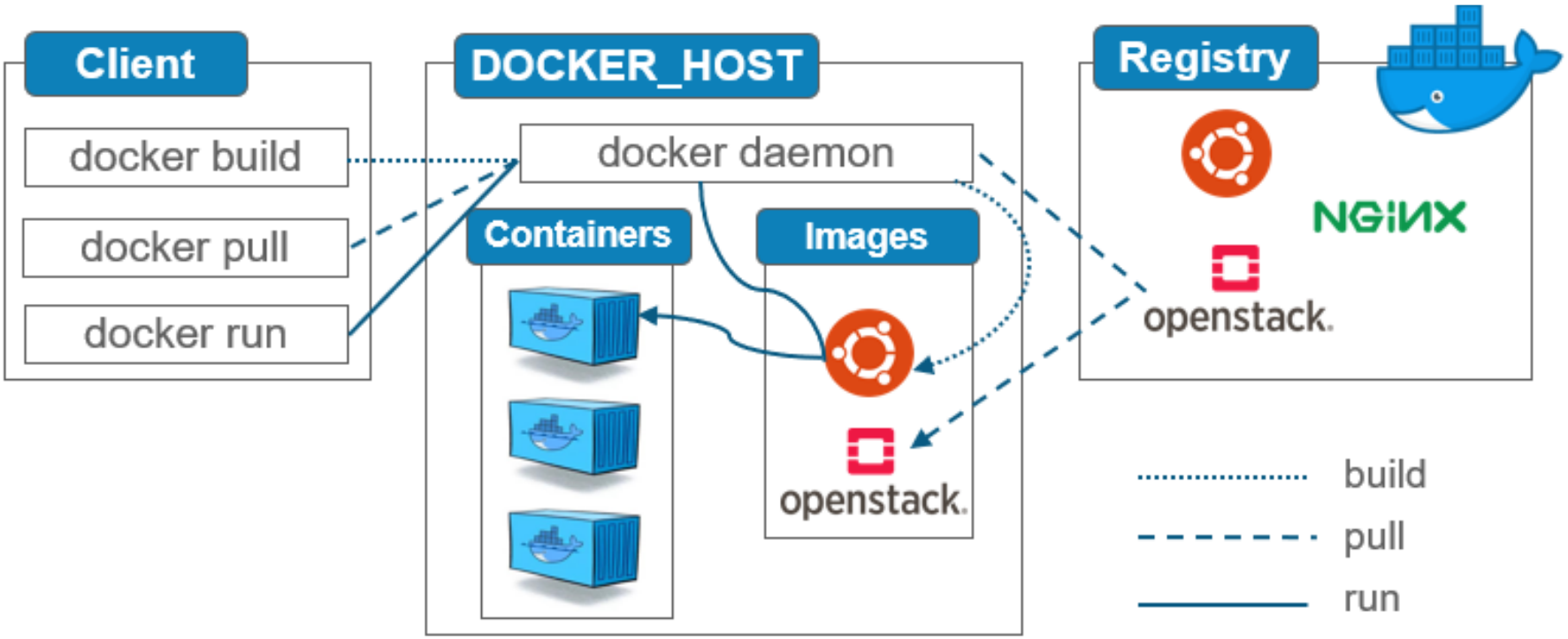
Docker Engine

- The **CLI** uses the Docker **REST API** to control or interact with the Docker daemon through scripting or direct CLI commands.
- Many other Docker applications use the underlying API and CLI.
- The **daemon** creates and manages Docker objects, such as images, containers, networks, and volumes.

Docker Architecture

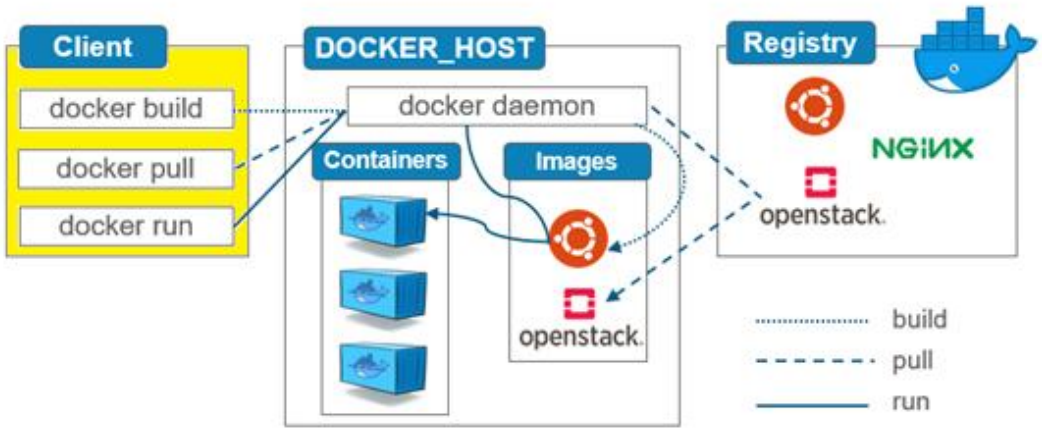
- Docker daemon does the building, running, and distributing Docker containers.
- The Docker client and daemon can run on the same system, or on a remote Docker daemon.
- The Docker client and daemon can communicate using a REST API:
  - over UNIX sockets or
  - a network interface.

Docker Architecture



https://docs.docker.com/engine/docker-overview/#docker-engine

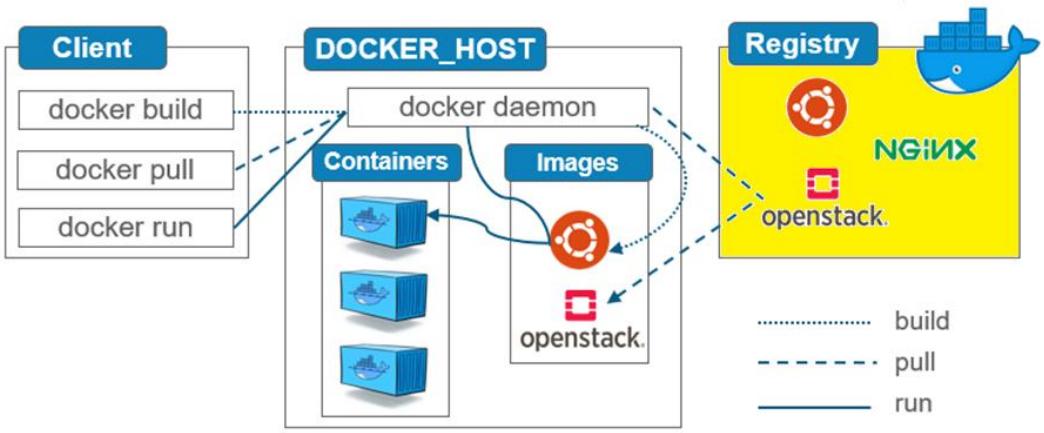
The Docker client



- The Docker client (docker) is the basic way to interact with Docker.
- Using commands such as docker run, the client sends these commands to dockerd.
- The Docker client can communicate with more than one daemon.

https://docs.docker.com/engine/docker-overview/#docker-engine

Docker registries



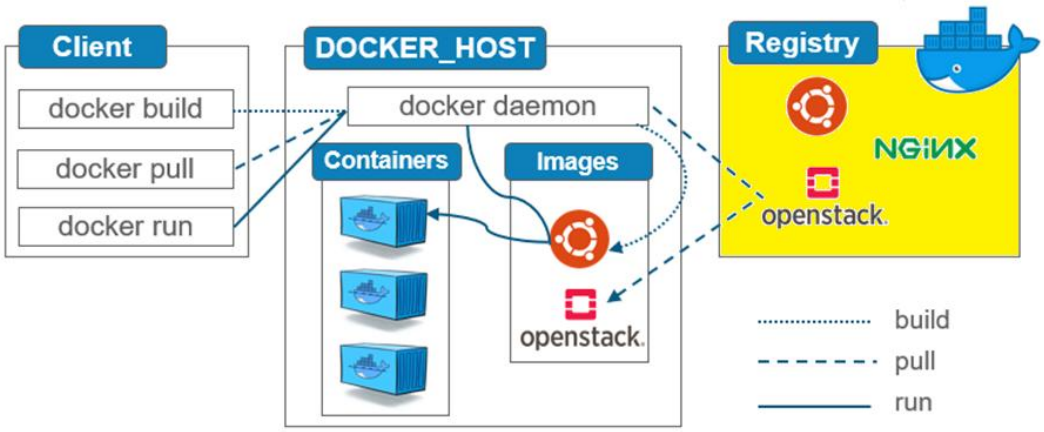
https://hub.docker.com/

- A Docker registry stores Docker images.
- Docker Hub is a public registry that anyone can use
- ***Docker looks for images on Docker Hub by default.***
- If one uses Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

IMPORTANT

https://docs.docker.com/engine/docker-overview/#docker-engine

Docker registries



- When we use the docker pull or docker run commands, the required images are pulled from our configured registry.
- When we use the docker push command, our image is pushed to configured registry.

https://docs.docker.com/engine/docker-overview/#docker-engine

Objects

- **Images** – It is a read-only template with necessary instructions used to create containers. Docker images are small and fast.

- **Containers** – Container is a runnable instance of an Image. One can use Docker API or CLI to create, start, stop, move, or delete a container.

Other Objects

- **Services** – The scalability of containers is managed by Services. The Services work together as **Swarm** and enables multi-host, multi-container deployment.
- One can define the desired state, such as the number of replicas of the service(containers) that must be available at any given time using services.

Other Objects

- **Volumes -**The **persisting data** generated by docker and used by Docker containers are stored in Volumes.
  - They are completely managed by docker through docker CLI or Docker API.

- **Networks-** Docker networking is a **passage** through which all the isolated container **communicate**.