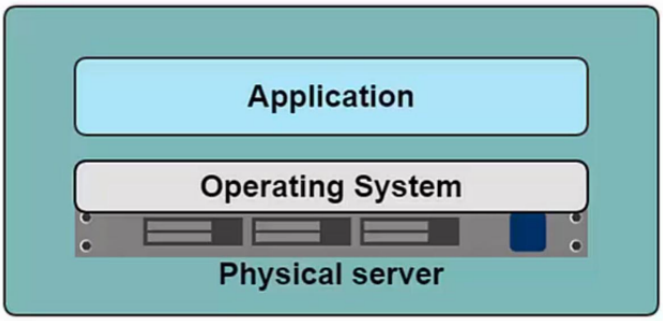
Exploring containerization :

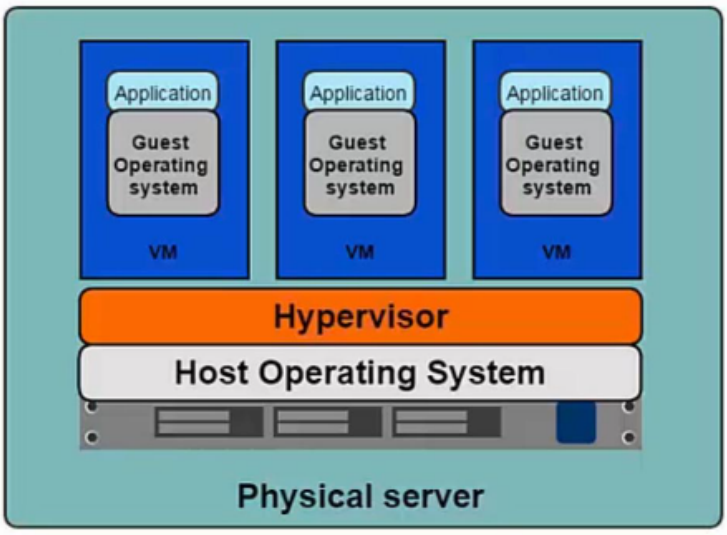
How we deployed application in PAST :

- Slow deployment times
- Huge costs
- Wasted resources
- Difficult to scale
- Vendor lock in



Hypervisor solve the problem of bare-metal :

- One physical server can contain multiple applications
- Each application runs in a virtual machine



Benefit of using VM :

- Better resource pooling
  - One physical machine divided into number of VMs
- Easier to scale
- VM's in the cloud
  - Rapid elasticity
  - Pay as you go model

Disadvantages of Vms :

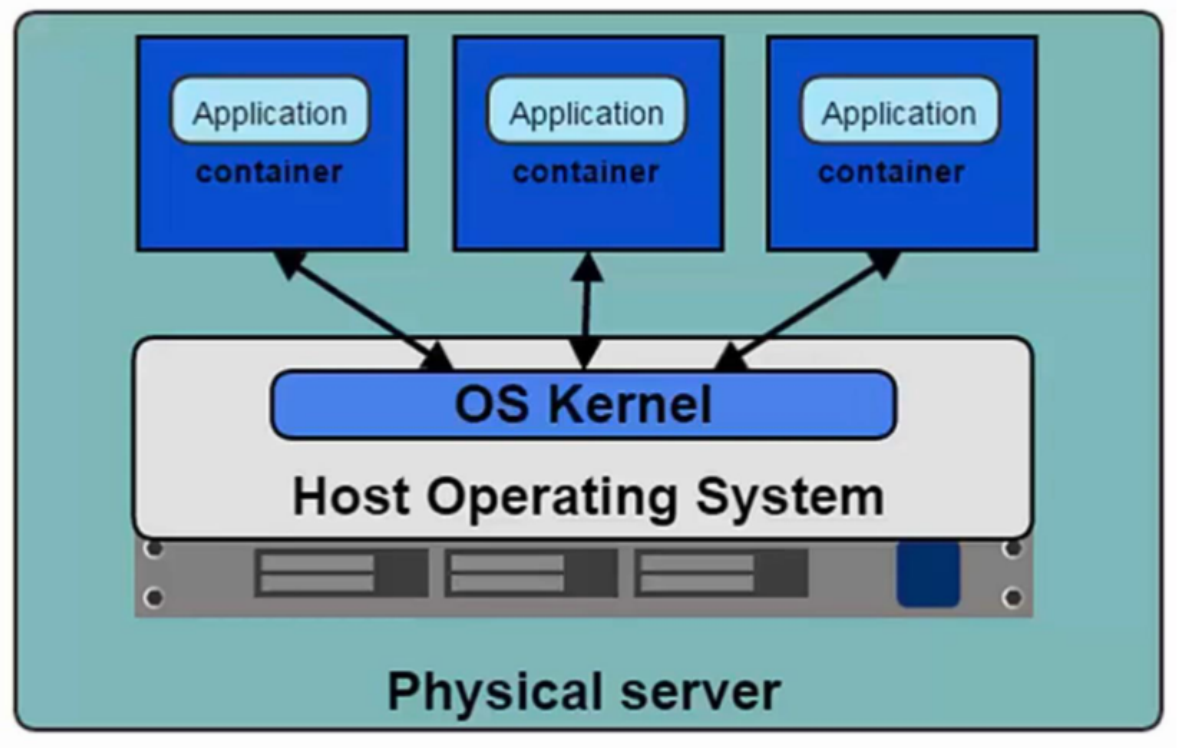
- Each VM still requires
  - CPU Allocation
  - Storage
  - RAM
  - An entire guest operation system
- The more VMs you run, the more resources you need
- Guest OS means wasted resources
- Application portability not guaranteed

INtroduction to containers :

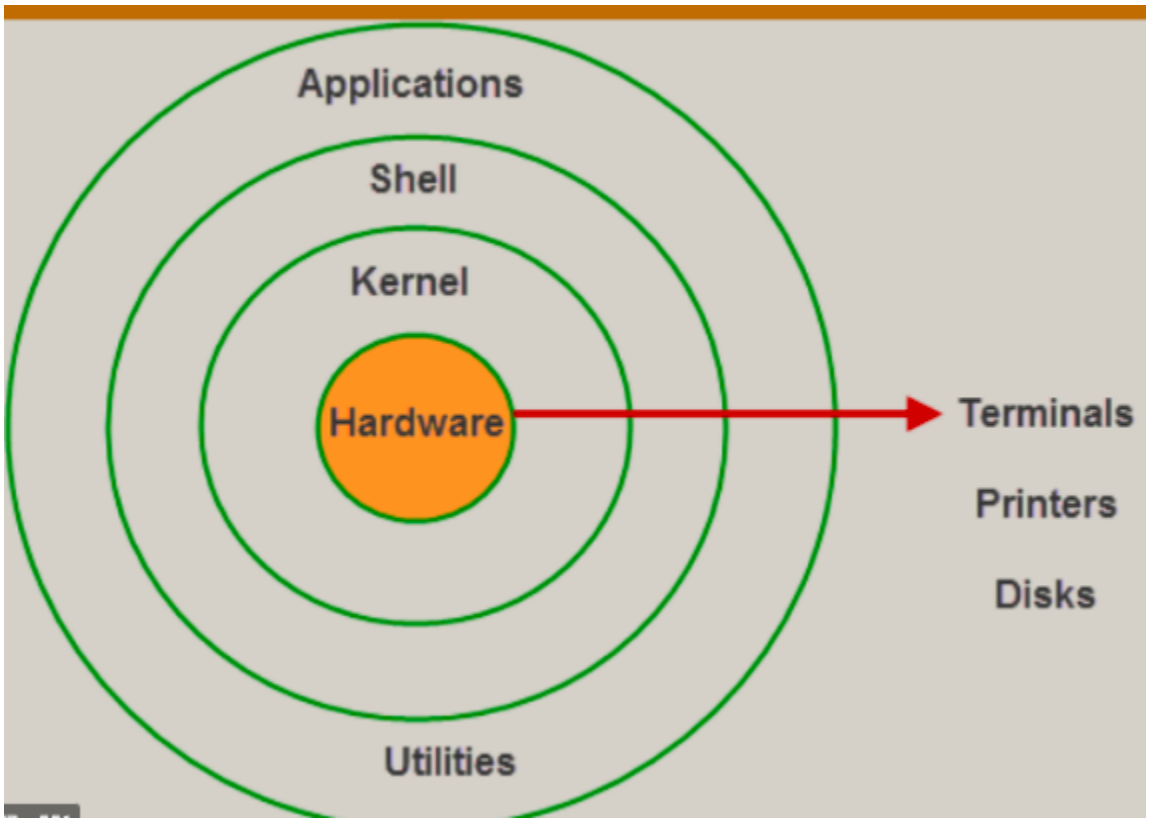
Container based virtualization uses the kernel on the host's operating system to run multiple guest instances

- Each guest instance is called a container
- Each container has its own
  - Root file system
  - Processes
  - Memory
  - Devices
  - Network ports

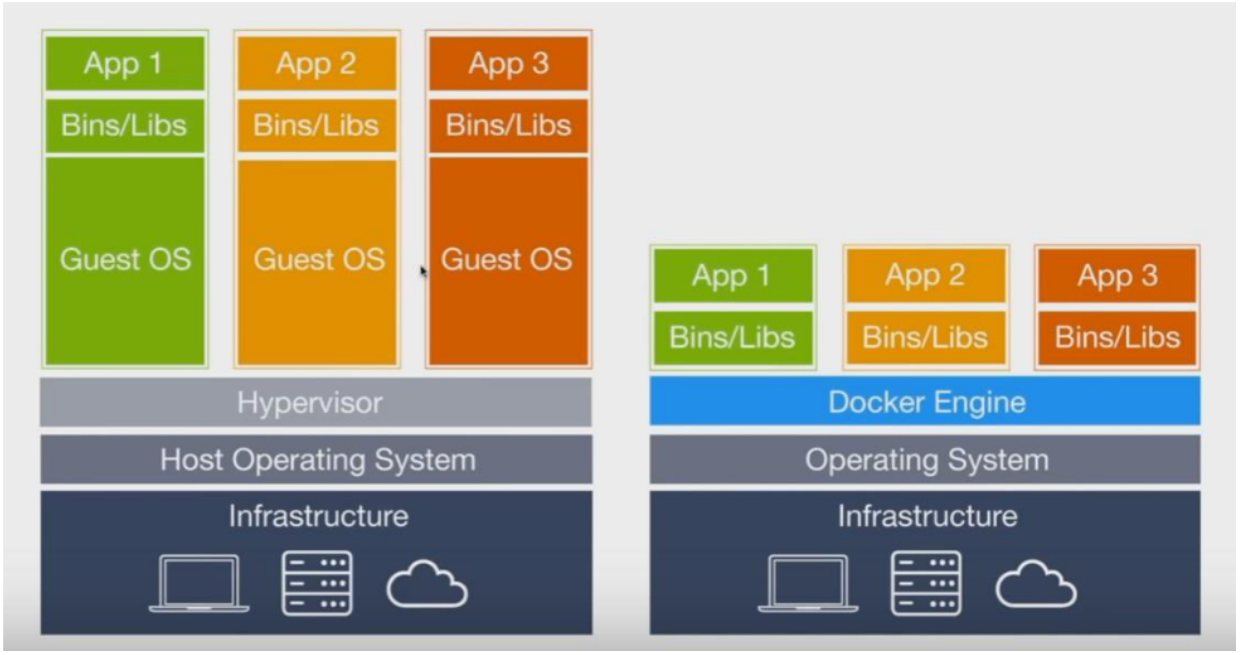
Container and host os relation :



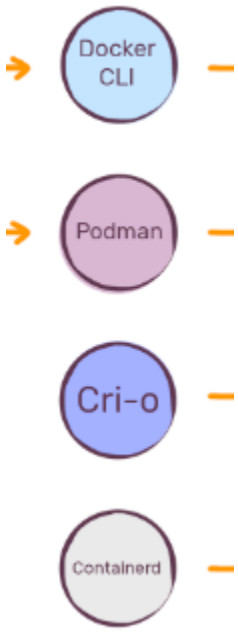
OS architecture :



VM vs containers:



Container runtimes:



Important Standard about containers:

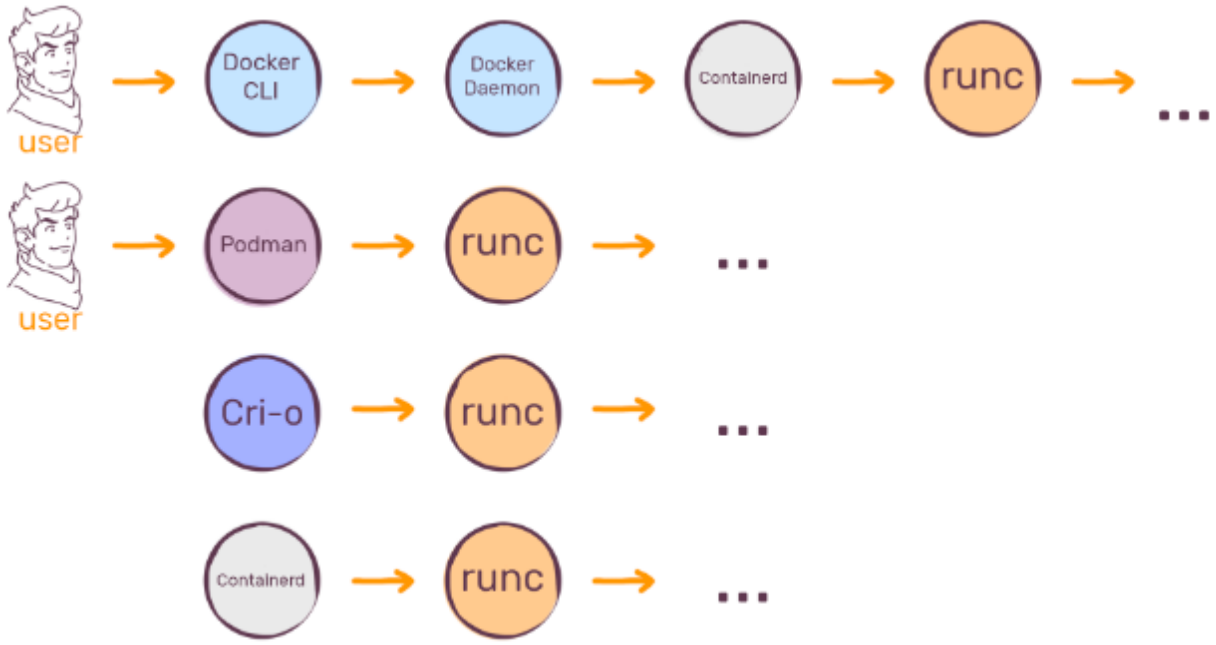
1. **OCI**

Open Container Initiative

The **Open Container Initiative** is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.

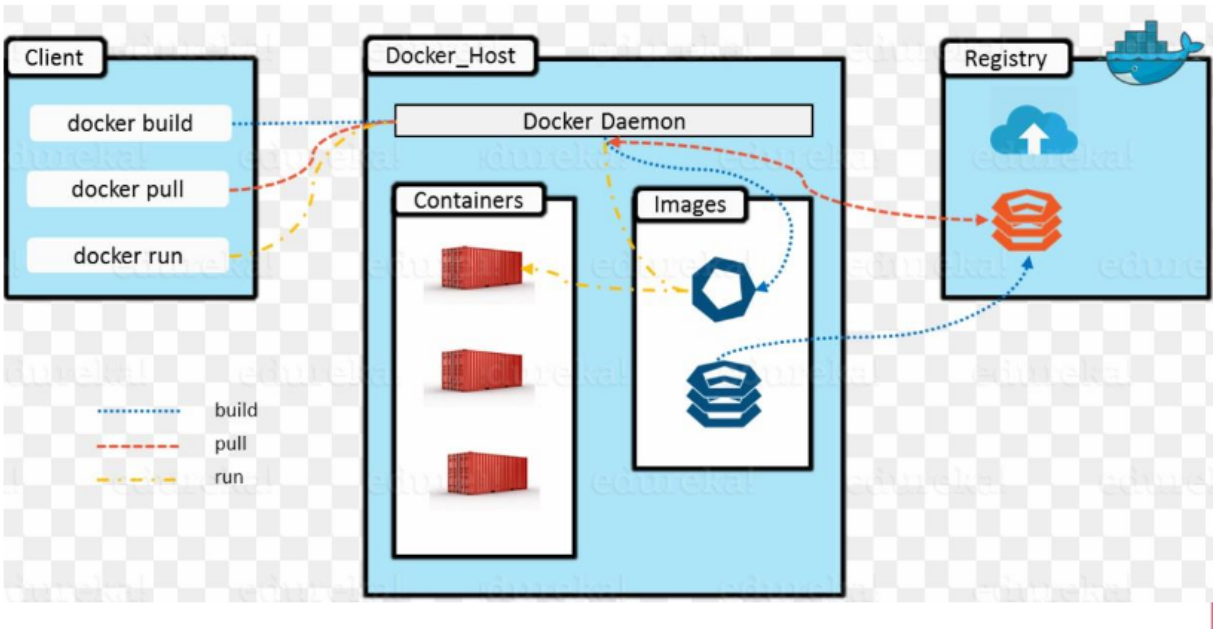
Established in June 2015 by Docker and other leaders in the container industry, the OCI currently contains two specifications: the Runtime Specification (runtime-spec) and the Image Specification (image-spec). The Runtime Specification outlines how to run a "filesystem bundle" that is unpacked on disk. At a high-level an OCI implementation would download an OCI Image then unpack that image into an OCI Runtime filesystem bundle. At this point the OCI Runtime Bundle would be run by an OCI Runtime.

How containers is getting created using any container client :



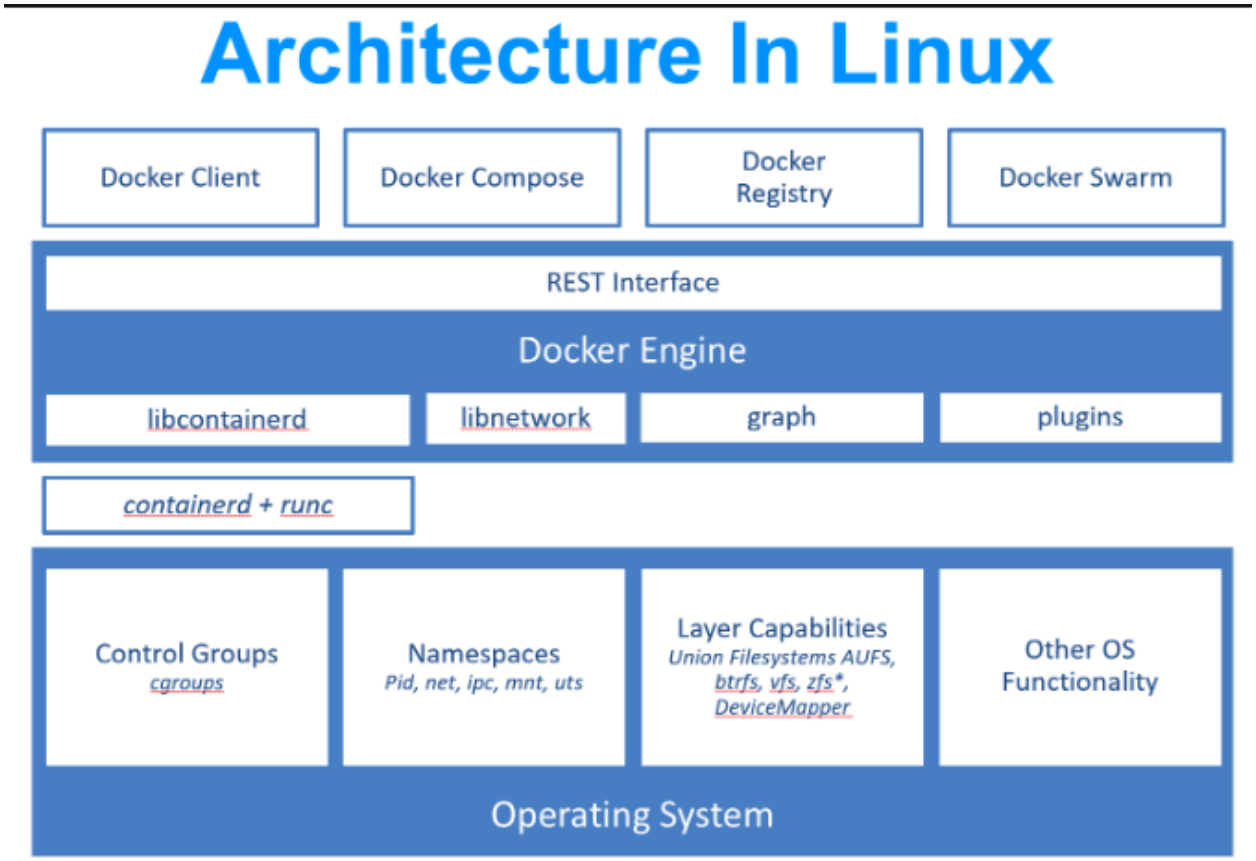
Introduction to Docker :

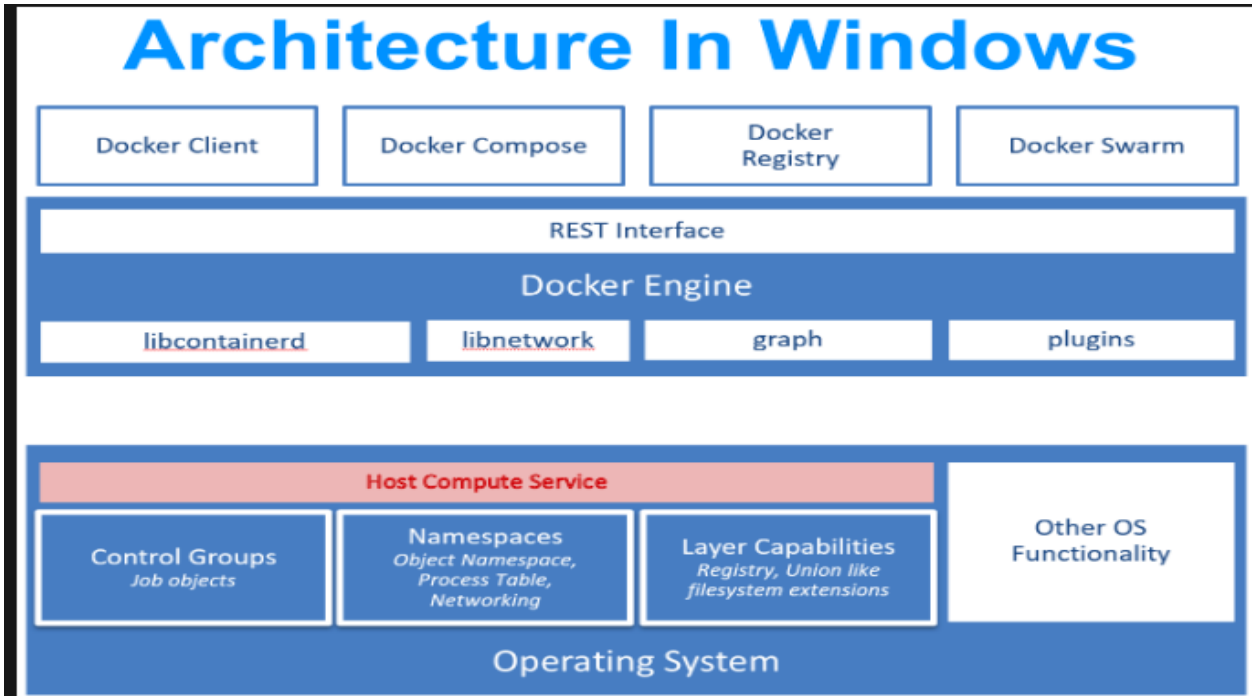
Docker architecture :



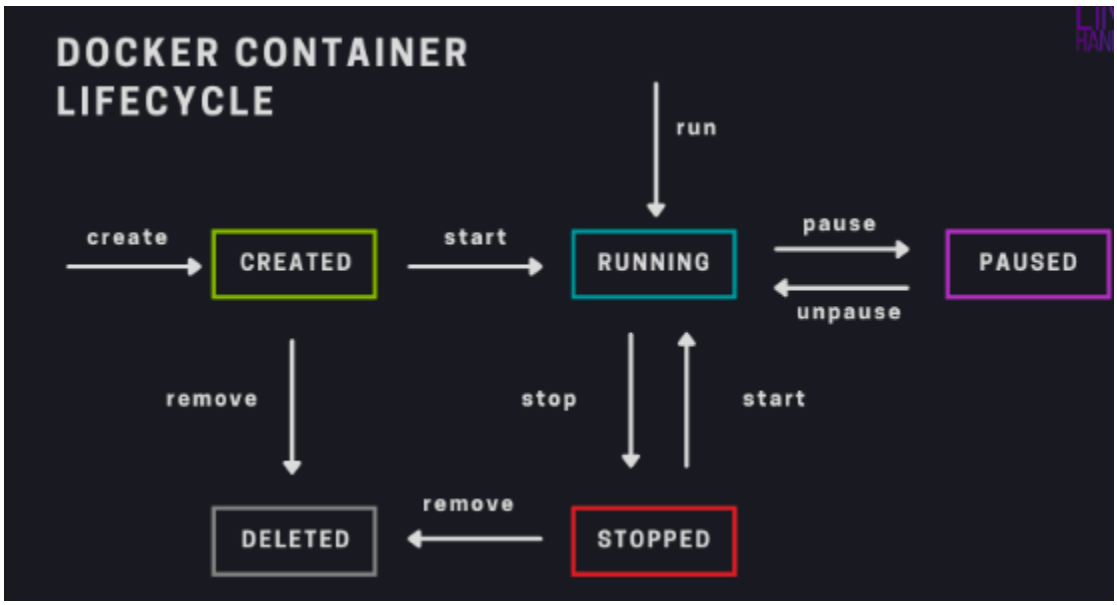
Installation of Linux based containers :

Linux container architecture :





Docker container life cycle :



Docker image Building :

Tools:

1. **Dockerfile**
2. **Buildah**
3. **Kaniko**

Docker Networking :

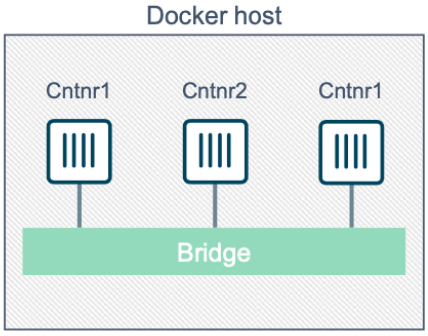
1. **Networking models for containers**

Container Networking Specifications

Container Networking Model CNM	Container Networking Interface CNI
• Specification proposed by Docker, adopted by projects such as libnetwork	• Specification proposed by CoreOS and adopted by projects such as Kubernetes, Cloud Foundry and Apache Mesos
• Plugins built by projects such as Weave, Project Calico and Kuryr	• Plugins built by projects such as Weave, Project Calico, Contiv Networking
• Supports only Docker runtime	• Supports any container runtime

2. **CNM look likes**

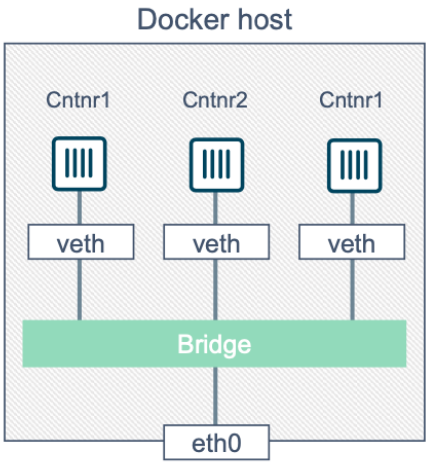
- The bridge driver creates a bridge (virtual switch) on a single Docker host
- Containers get plumbed into this bridge
- All containers on this bridge can communicate
- The bridge is a private network restricted to a single Docker host



3. **More details about container networking with host connection**

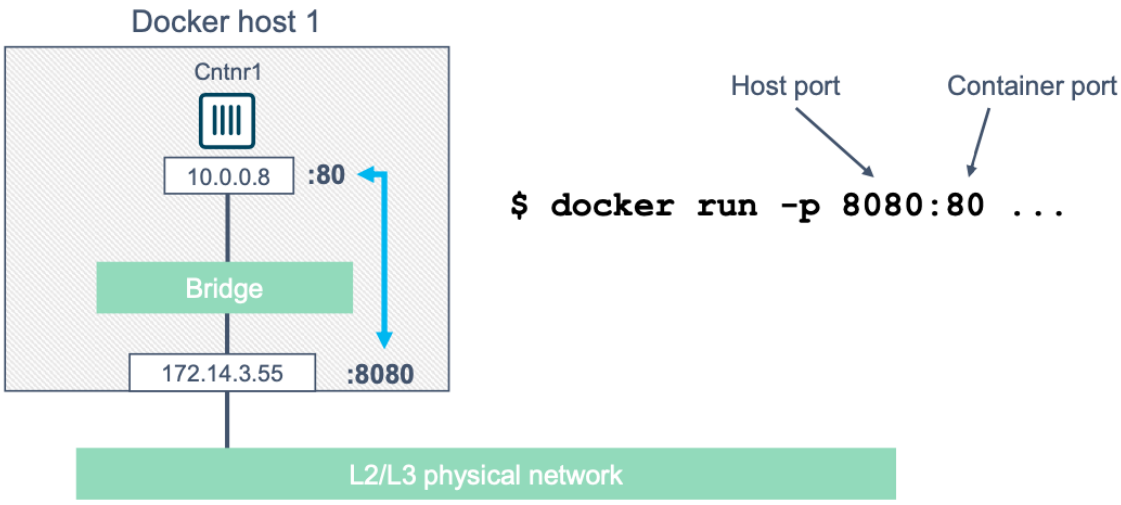
Bridge Networking in a Bit More Detail

- The bridge created by the bridge driver for the pre-built bridge network is called docker0
- Each container is connected to a bridge network via a veth pair
- Provides single-host networking
- External access requires port mapping



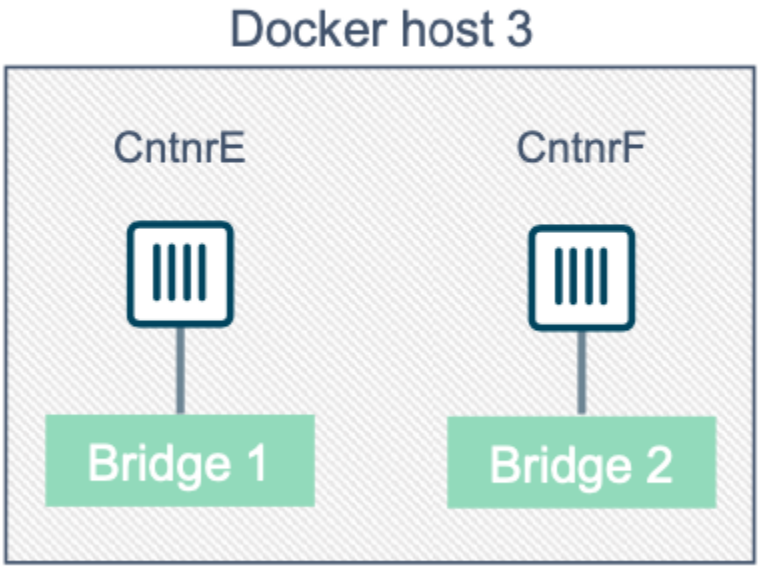
4. **NAT and port mapping**

Docker Bridge Networking and Port Mapping



Custom bridges and drivers :

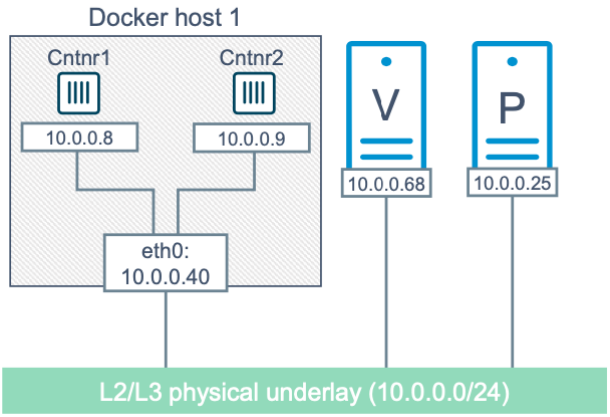
Driver bridge :



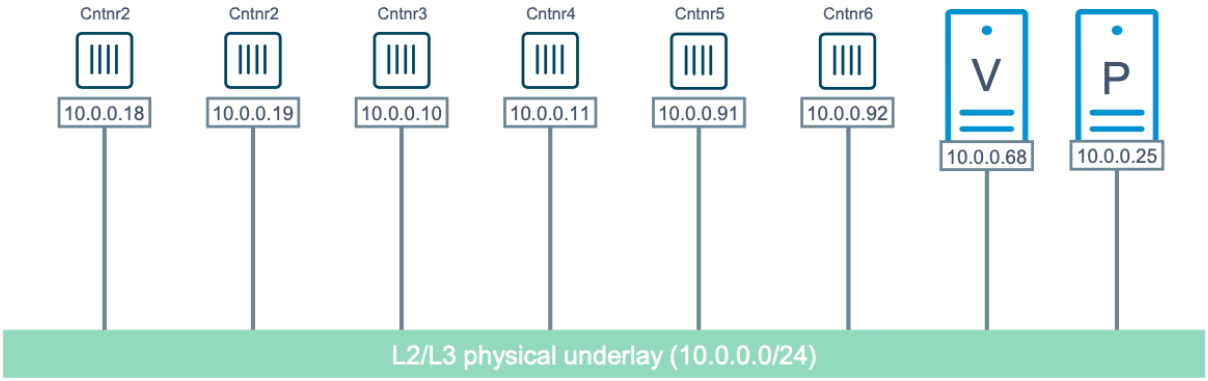
Driver MacVLAN :

What is MACVLAN?

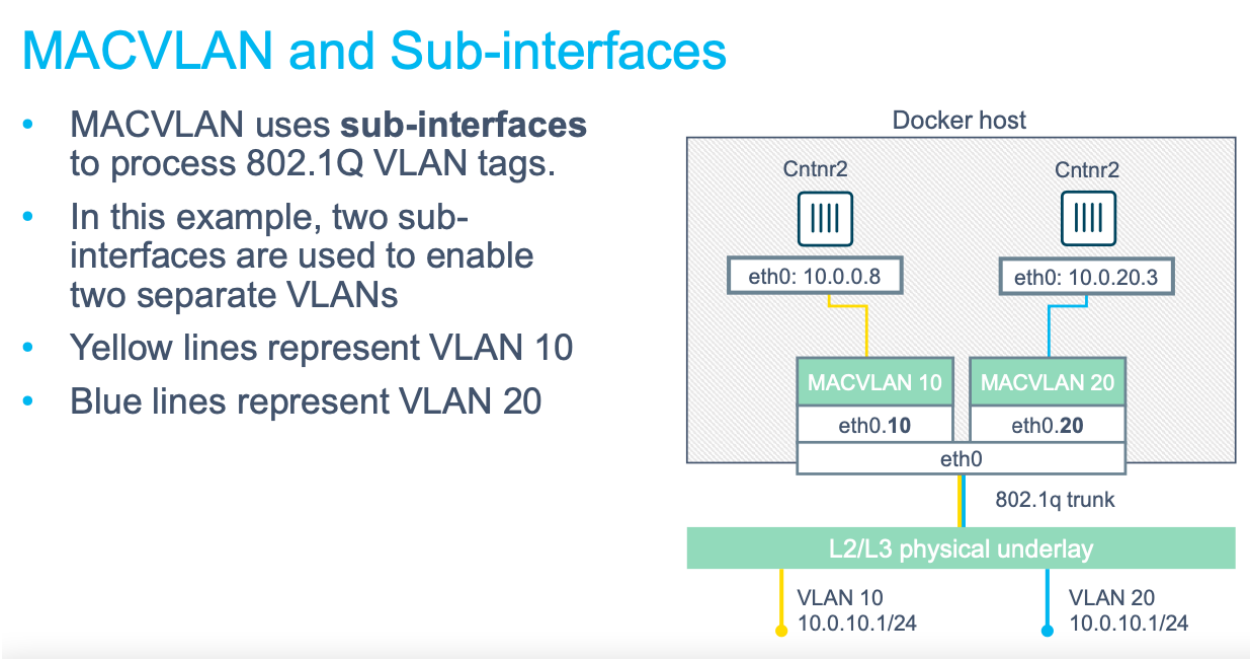
- A way to attach containers to existing networks and VLANs
- Good for mixing containers with VMs and physical machines
- Ideal for apps that are not ready to be fully containerized
- Uses the well known MACVLAN Linux network type
- Nothing to do with Mac OS!



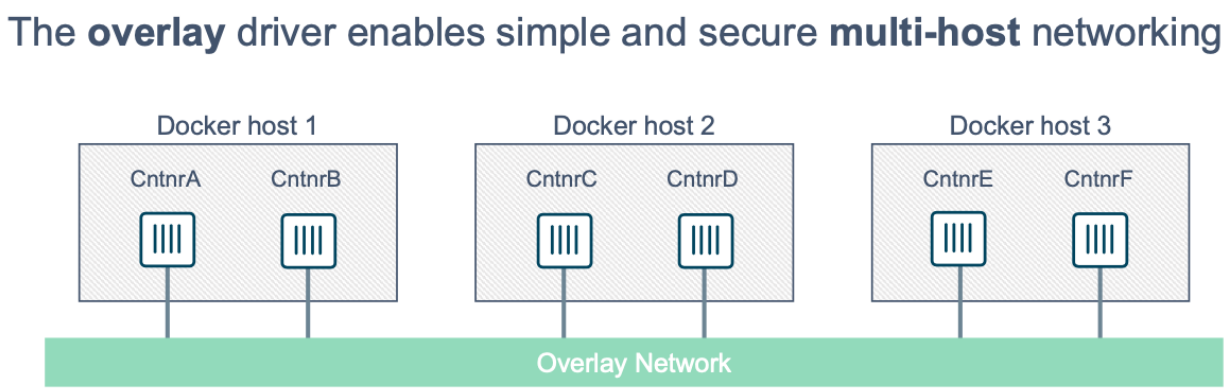
View after mac vlan :



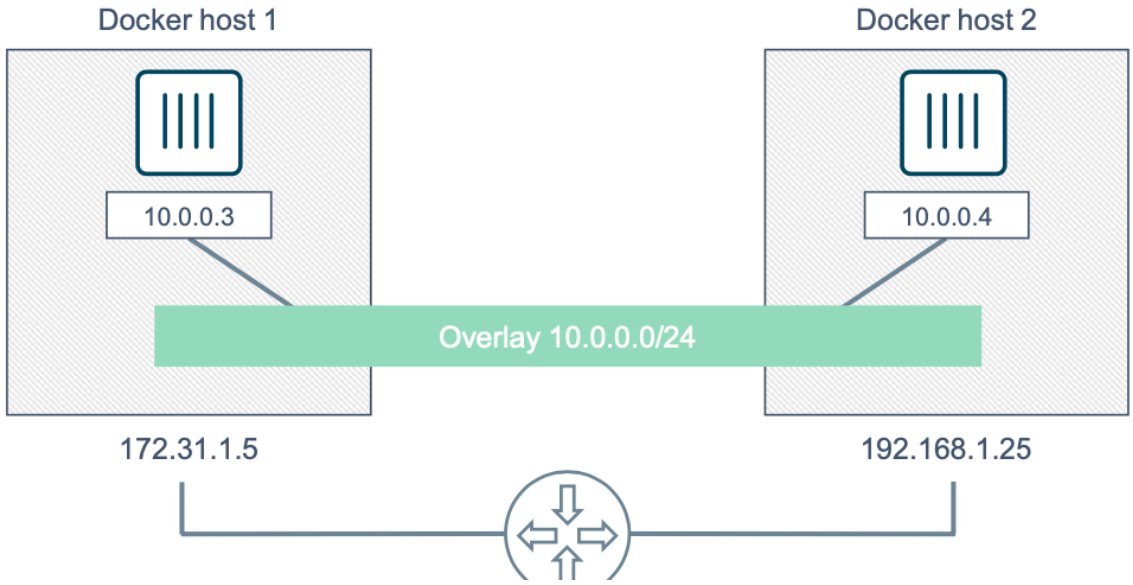
MAC vlan with trunking :



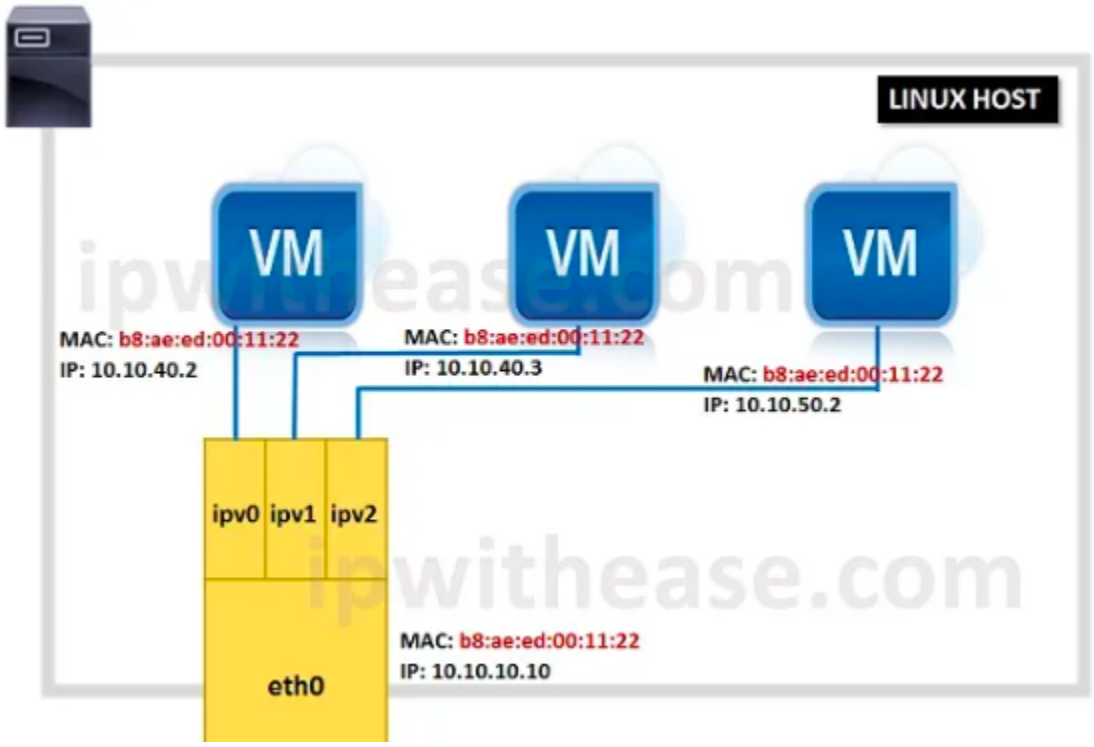
Overlay Network driver :



One more view :



IPvlan Network driver :


LINUX HOST
VM
VM
VM
MAC: b8:ae:ed:00:11:22
IP: 10.10.40.2
MAC: b8:ae:ed:00:11:22
IP: 10.10.40.3
MAC: b8:ae:ed:00:11:22
IP: 10.10.50.2
ipv0
ipv1
ipv2
eth0
MAC: b8:ae:ed:00:11:22
IP: 10.10.10.10

Docker Storage :

VOlume concept :

- This is where persistent data lives
- Extremely pluggable
- Network attached storage is extremely useful here

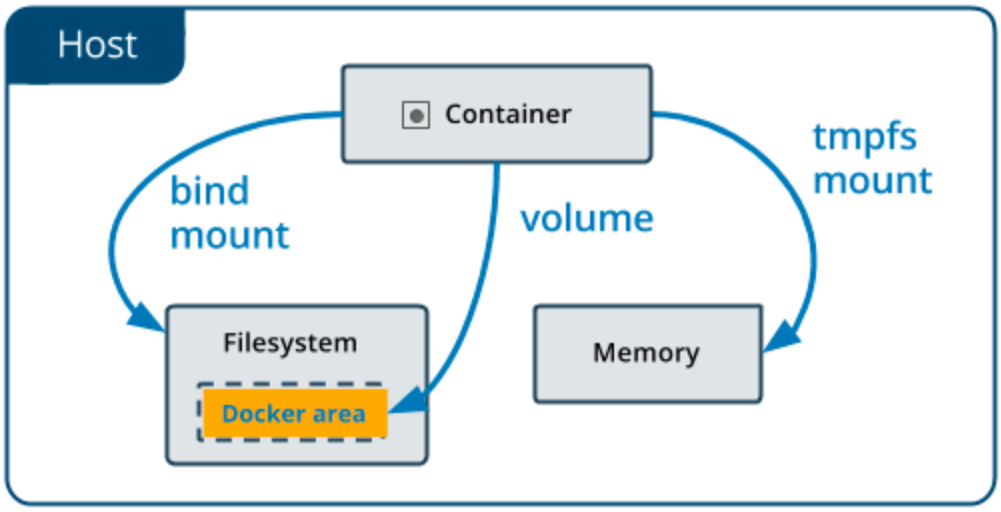
VOlume benefits :

Working with Volumes

Docker volumes can be used to achieve many things, including:

- Bypassing the copy-on-write system to obtain native disk I/O performance.
- Bypassing copy-on-write to leave some files out of `docker commit`.
- Sharing a directory between multiple containers.
- Sharing a directory between the host and a container.
- Sharing a *single file* between the host and a container.

Docker host and volumes :



Docker compose :