

# RAID types

RAID (Redundant Array of Inexpensive Disks or Drives, or Redundant Array of Independent Disks).

Purposes:

performance improvement (max. write read speed),  
increased data security against hard drive failure.

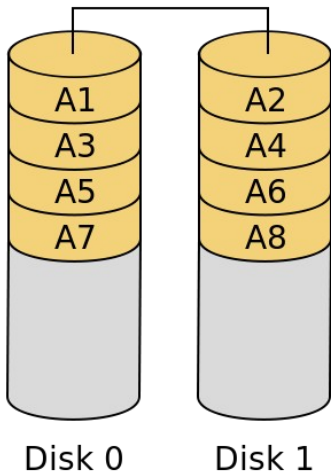
Hardware RAID = RAID controller

or

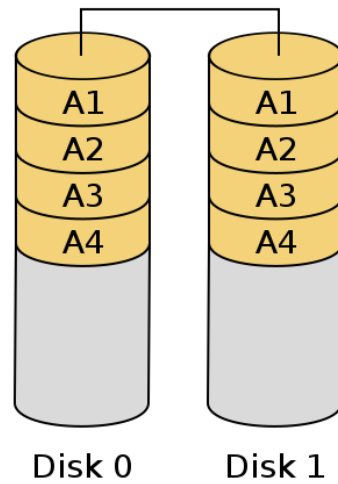
Software RAID

# RAID types

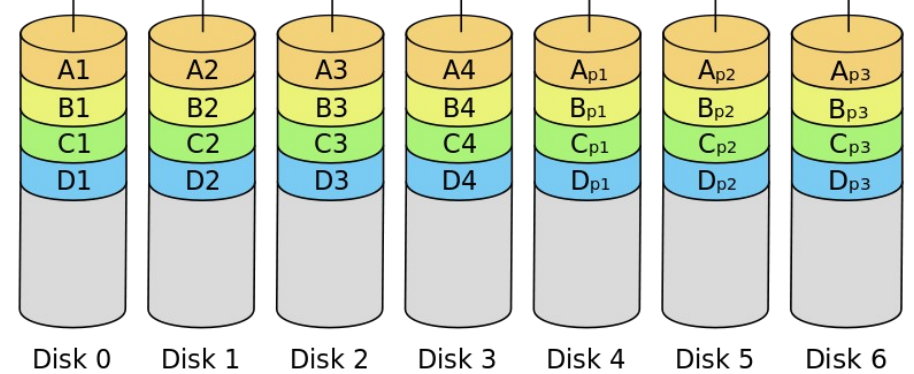
RAID 0



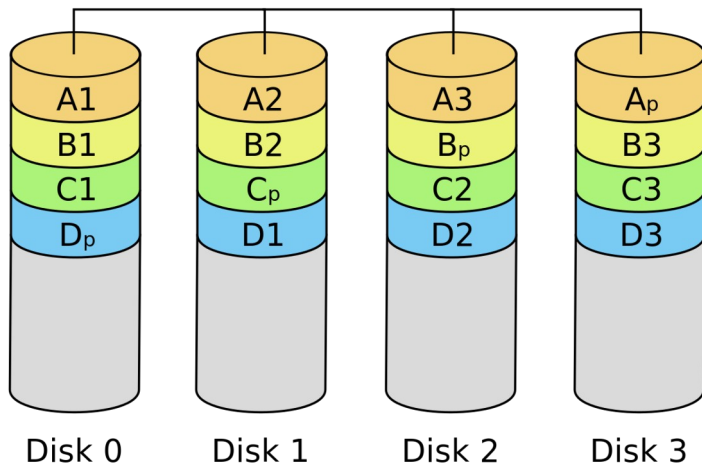
RAID 1



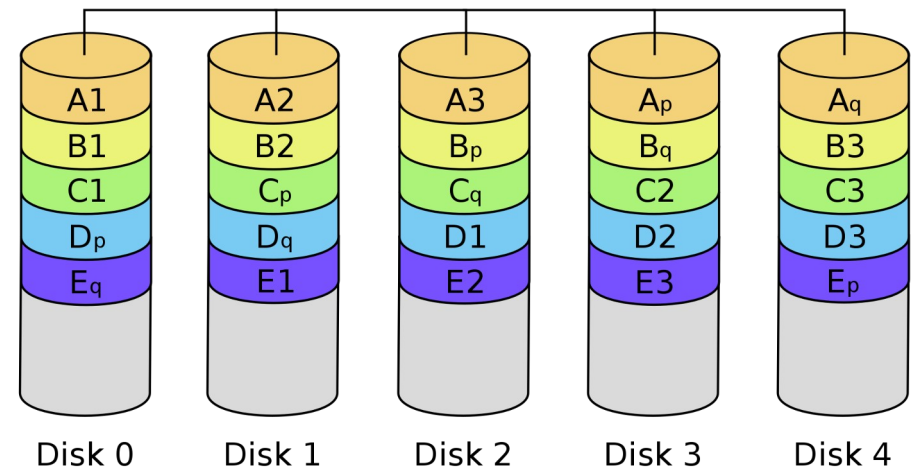
RAID 2 (Hamming code)



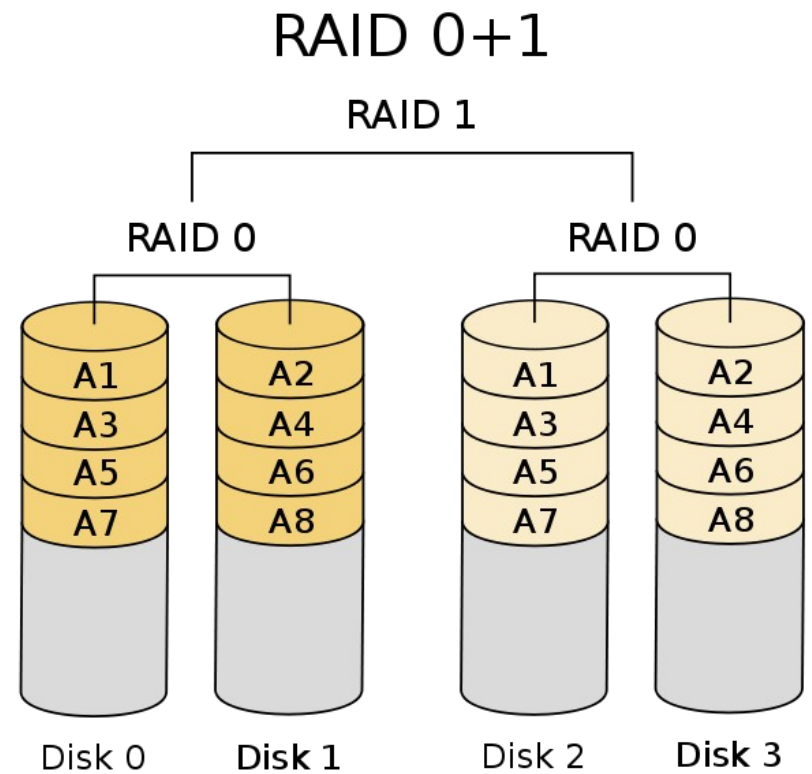
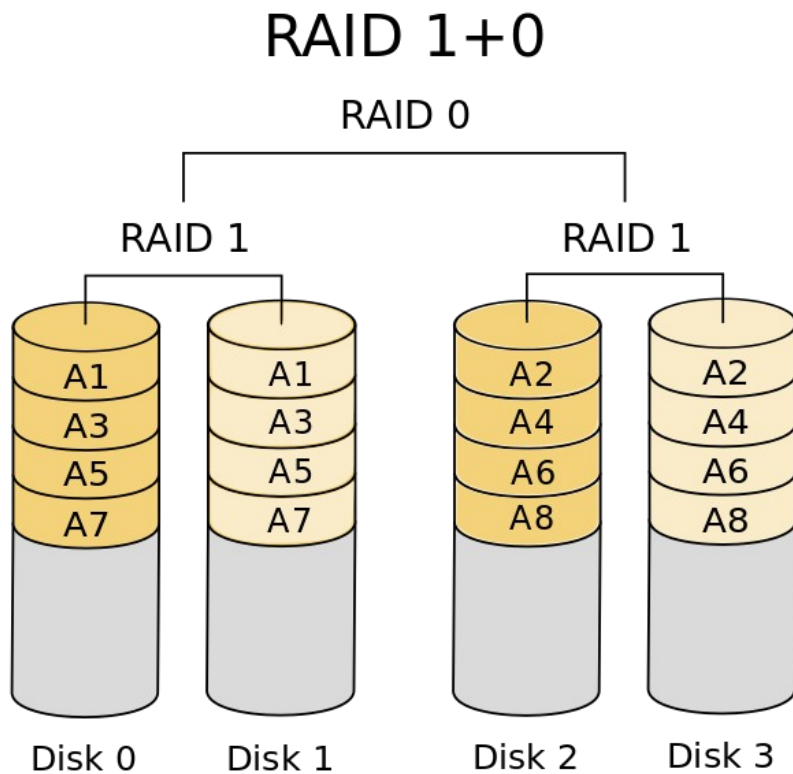
RAID 5



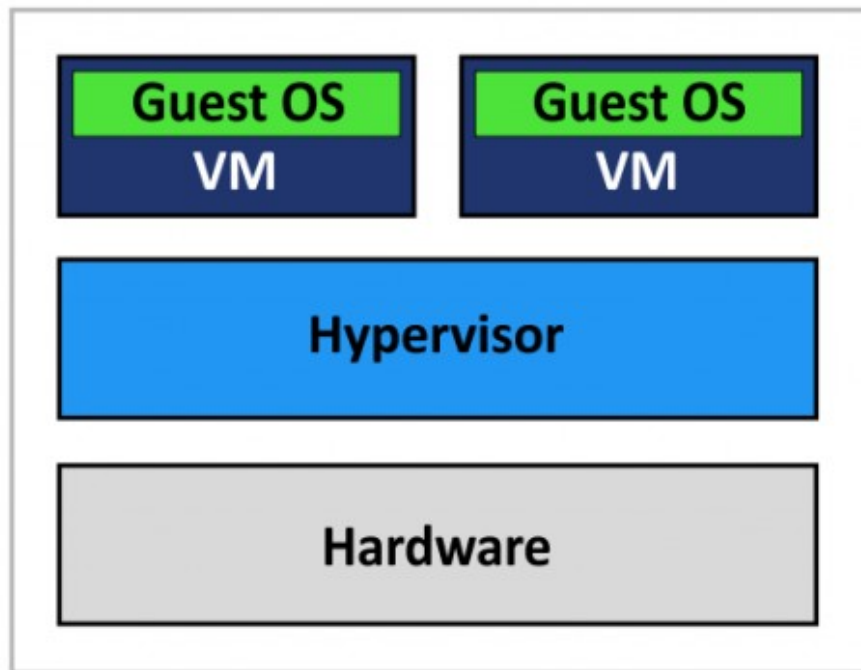
RAID 6



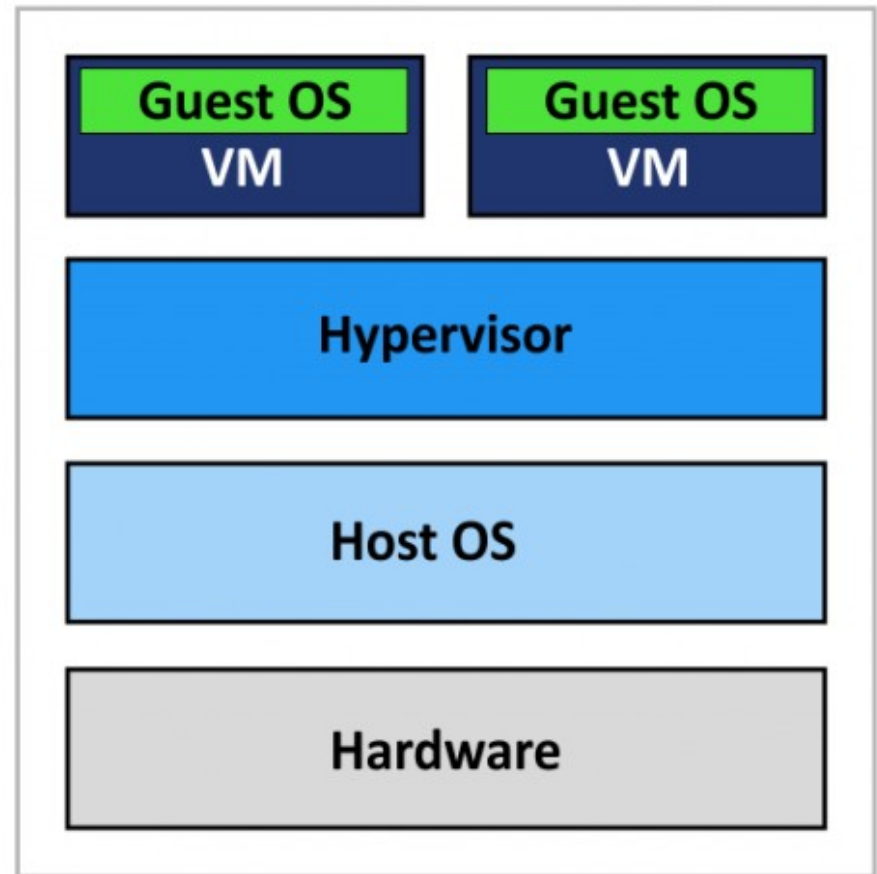
# Mixed RAID types principle



# Virtual machine approach

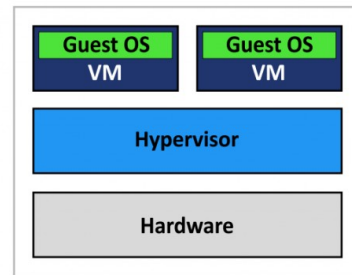


**Type 1 Hypervisor  
(Bare-Metal Architecture)**

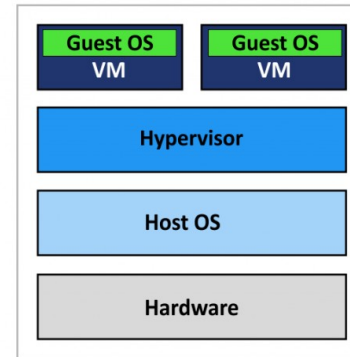


**Type 2 Hypervisor  
(Hosted Architecture)**

# Virtual machine approach



Type 1 Hypervisor  
(Bare-Metal Architecture)

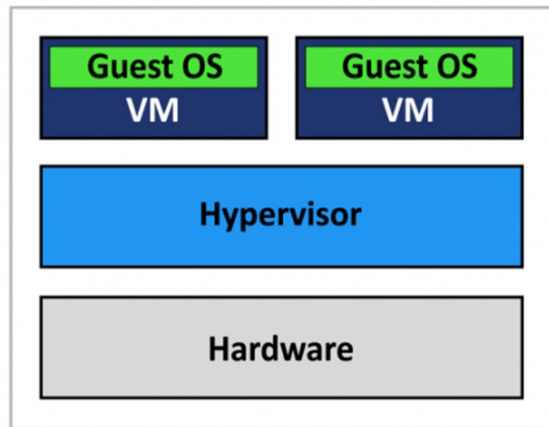


Type 2 Hypervisor  
(Hosted Architecture)

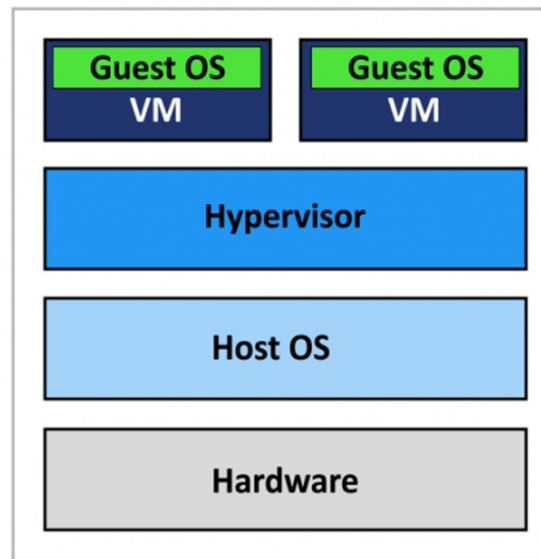
Vmware ESXi  
Microsoft Hyper-V Server  
Citrix XenServer  
KVM

VMware Workstation  
Virtualbox

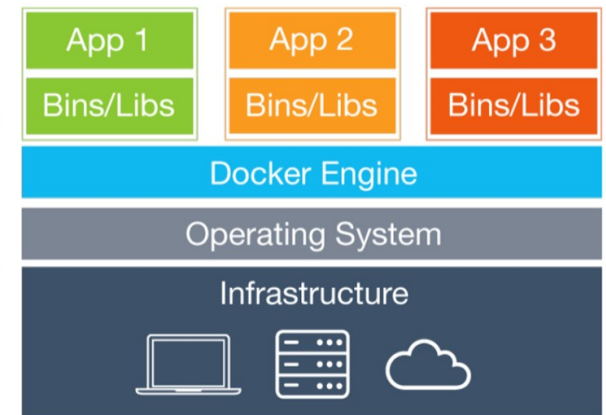
# VM vs Container



Type 1 Hypervisor  
(Bare-Metal Architecture)

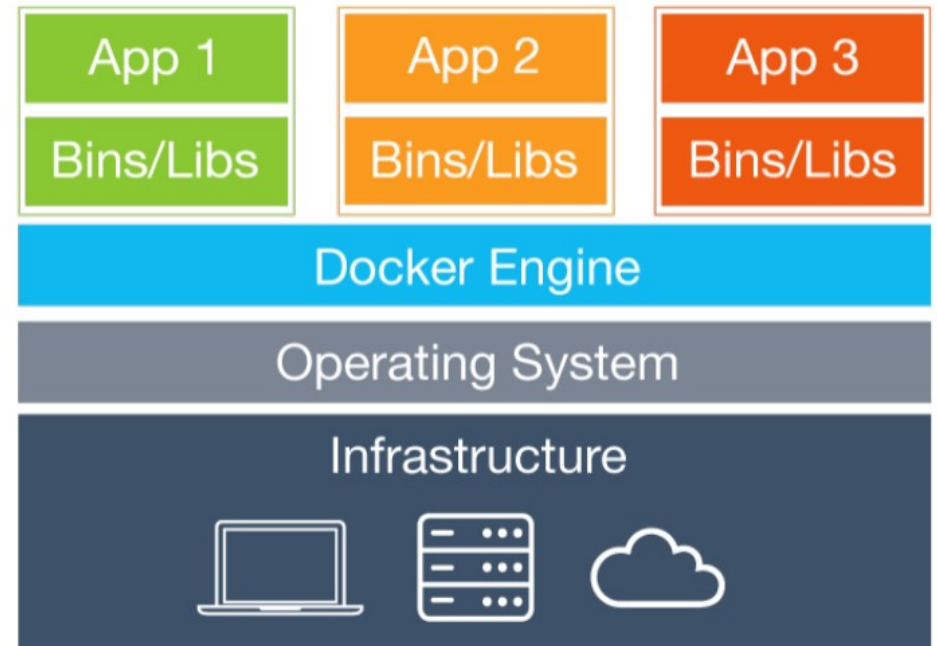
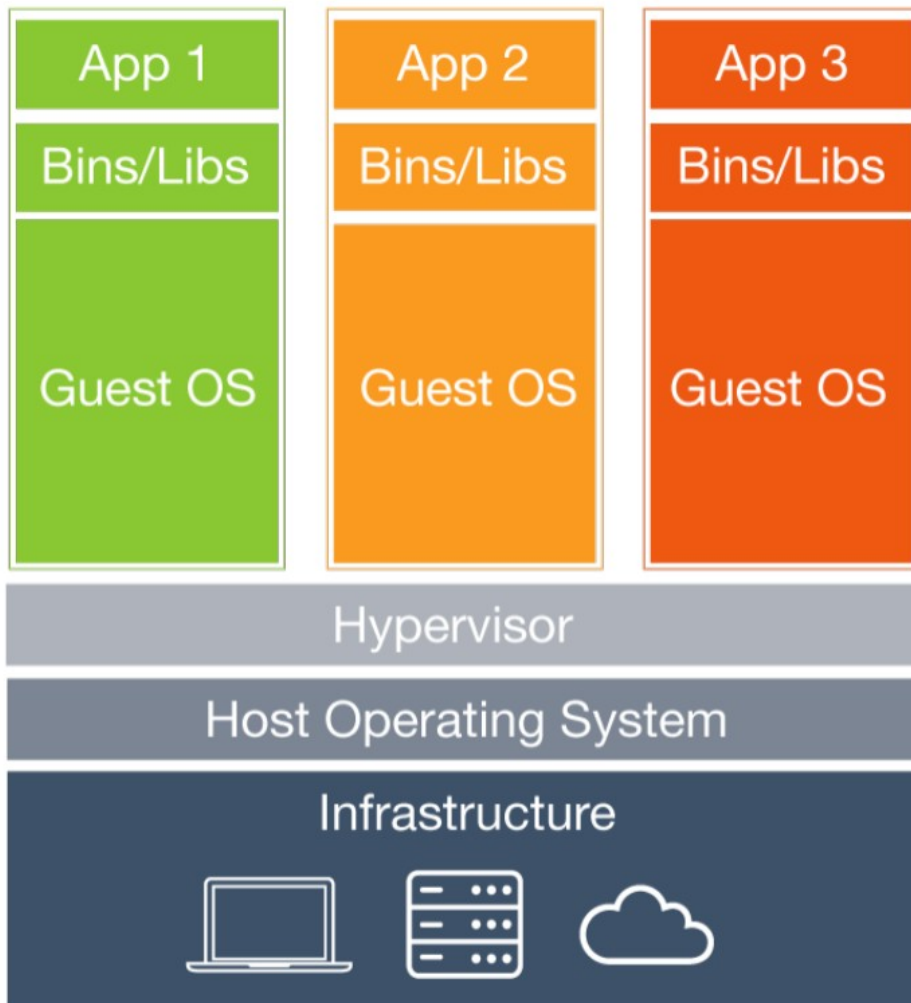


Type 2 Hypervisor  
(Hosted Architecture)

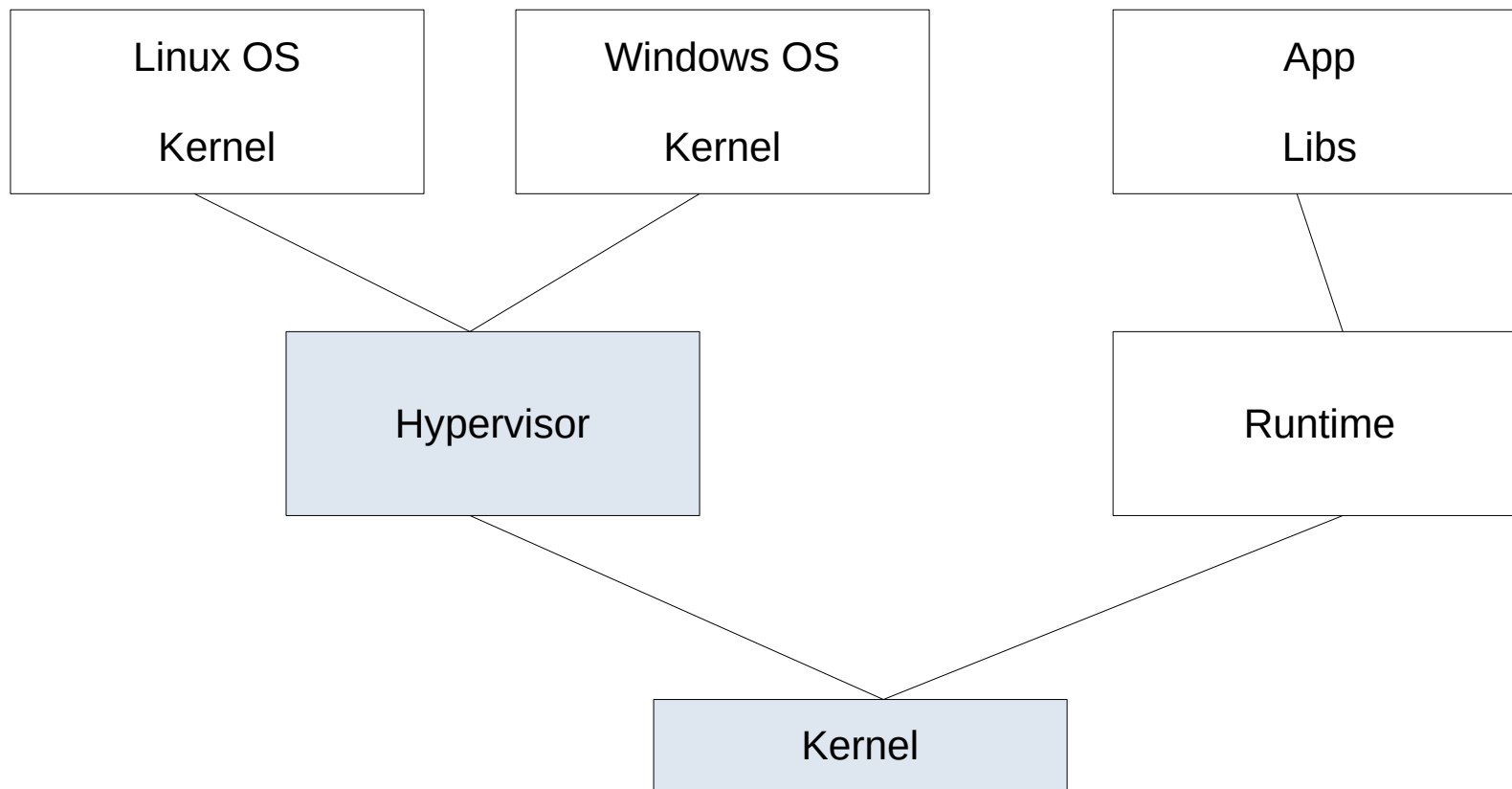


Docker

# VM vs Container



# VM vs Container





# Containers

Linux containers are ordinary processes, which isolated by Linux kernel tools:

- \* Namespaces for resource isolation
- \* Control Groups for resource constraints
- \* Permissions, Capabilities, SELinux, AppArmor and other Linux security constraints

All Linux processes are started as containers.

- \* `cat /proc/PID/cgroup` will see the Cgroups the process is in
- \* `cat /proc/self/attr/current` will show SELinux labels
- \* `ls /proc/PID/ns` will show a list of namespaces the process is in

Container different is that it is started from a container image. So, container runtime is the operating system part that manages the cgroups, SELinux labels and namespaces and starts the container from an image

# Container is

- \* A container is a running instance of an image
- \* The image contains the application code, language runtime and libraries
- \* External libraries such as libc are typically provided by the host operating system, but in a container is included in the images
- \* The container image is a read-only instance of the application that just needs to be started
- \* While starting a container, it adds a writable layer on the top to store any changes that are made while working with the container

# What is an Image?

- \* A container image is a TAR file that combines the following
  - \* The container root file system: a directory that looks like the standard root of the operating system, but presented as a mount namespace
  - \* Metadata: a JSON file that specifies how to run that root file system, including all settings required to get to a functional container (entrypoint, environment variables and more)
- \* Container images are layered: you can install additional content, add a new JSON file and store the differences in a new TAR file
- \* Images are standardized by the OCI

---

- \* Container images are typically shared through public registries, or by sharing mechanisms to build them easily, such as Dockerfile
- \* The Docker container image format has become the de facto standard image format
- \* Open Container Initiative (OCI) has standardized the Docker container image format

# Container Image Layers

- \* Docker images are made up of a series of filesystem layers
- \* Each layer adds, removes or modifies files from the preceding layer in the filesystem
- \* This is called an overlay filesystem, and different overlay filesystems exist, like aufs, overlay and overlay2
- \* By using these different image layers, and pointing to other image layers in a smart way, it's easy to build container images that have support for multiple versions of vital components
- \* Apart from the different layers, container images have a container configuration file that provides instructions on how to run the container

# Docker Essentials

<https://docs.docker.com/get-started/>