Container Virtualization Overview

Benjamin Gutierrez

What is a Container?

- Emerging/mature virtualization lightweight cloud technology
- A container lets you run a Linux system within another Linux system.
- More precisely, it is a group of processes on a Linux box, put together in an isolated environment.
- Inside the box, it looks like a VM. Outside the box, it looks like normal processes.

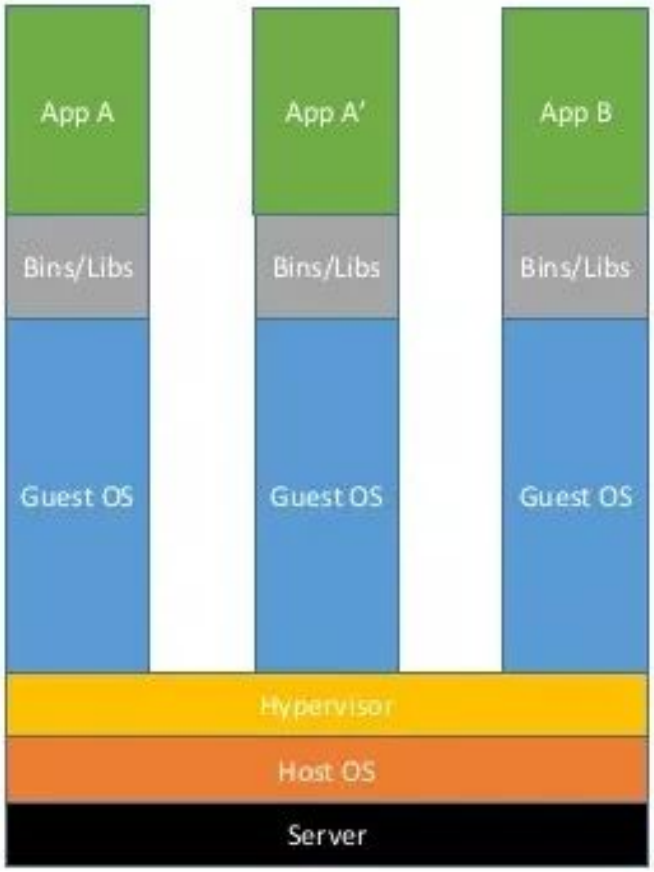
What is a Container?

- LXC (LinuX Containers) project provides a set of userspace tools and utilities to manage Linux containers.
- While a Virtual Machine is a whole other guest computer running on top of your host computer (sitting on top of a layer of virtualization), a container is an isolated portion of the host computer, sharing the host kernel (OS) and even its bin/libraries if appropriate.
- Higher densities on the same host

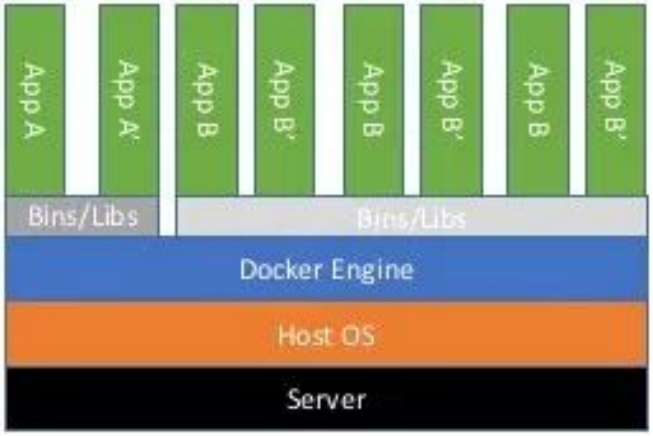
Why we want them?

- An application developer writes a program, then gives it to the IT department to deploy but it doesn't work because there are various dependencies missing (i.e. the developer had them in their environment but missing in the server environment).
- Containers solve this by creating a complete “Shipping unit” dependency for an application including middleware, runtimes, libraries and even the OS requirements.
- Additionally, each of these dependencies/layers are packaged up and run in their own user-mode container, isolating them from other applications avoiding problems with applications not being compatible with each other.
- These applications running in containers have their own view of the file system, registry and even networking addresses.

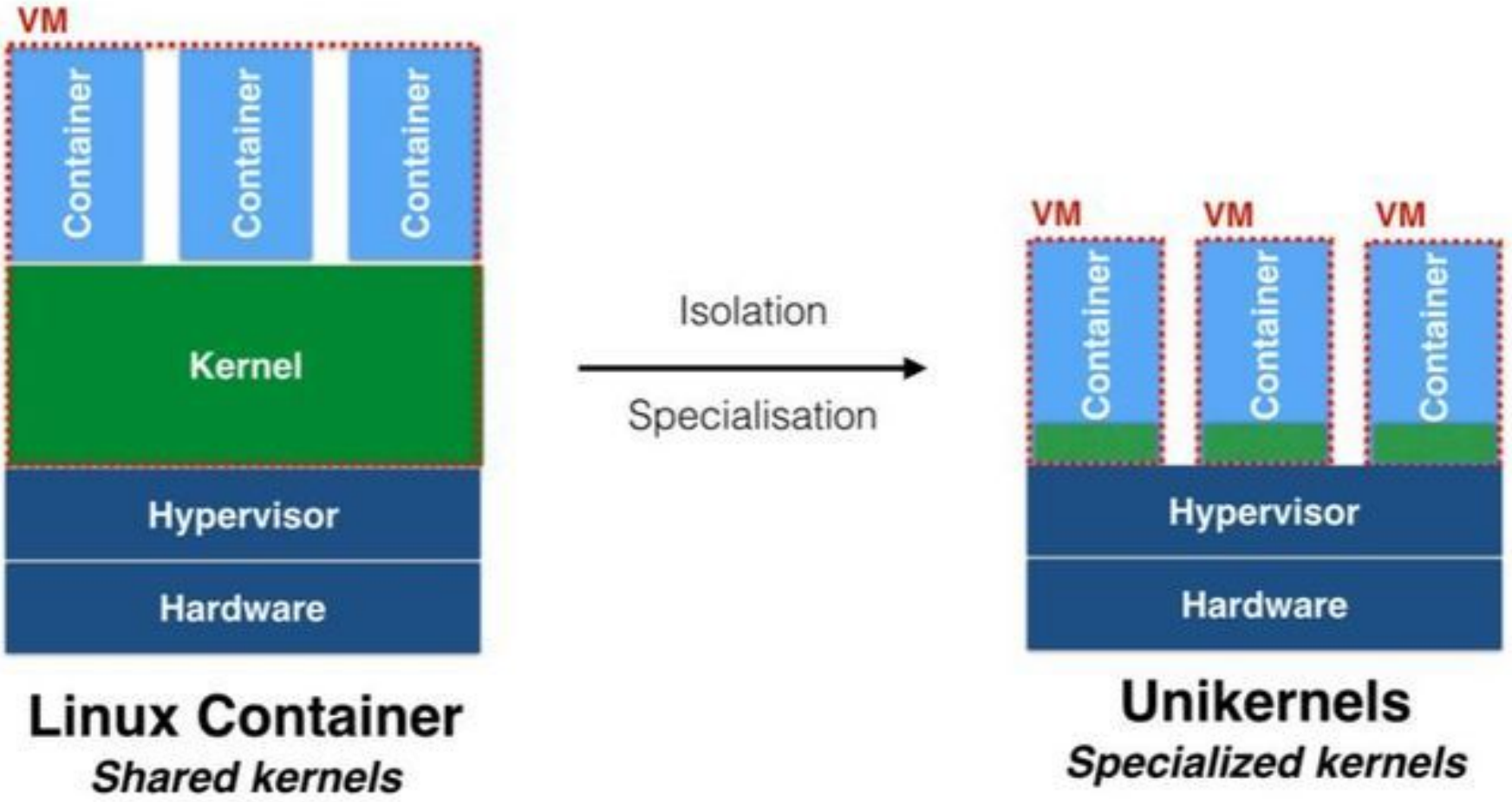
What is a Container?”.



Containers are isolated, but share OS and, where appropriate, bins/libraries

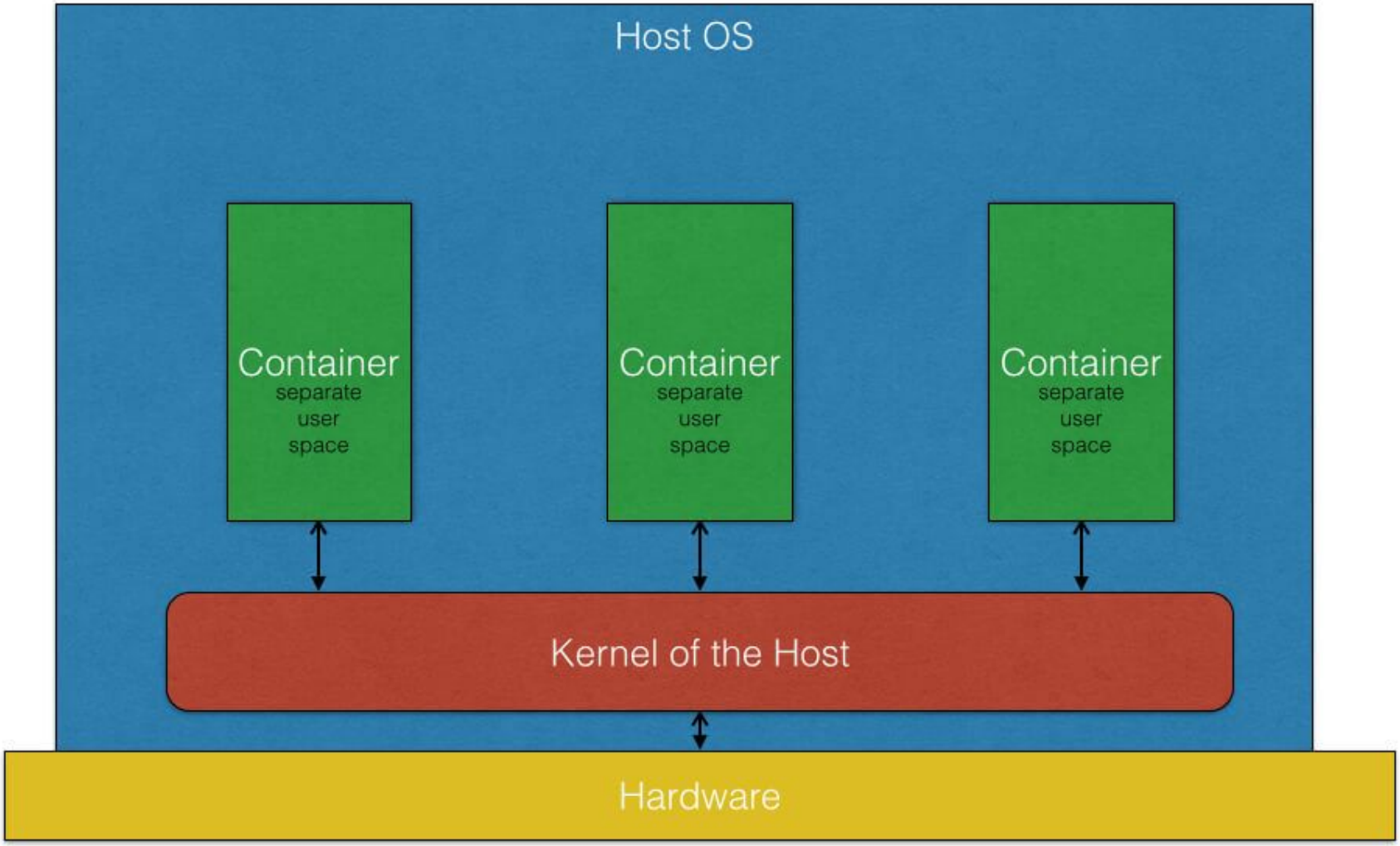


Share a Kernel



Process Isolation and Resource Control

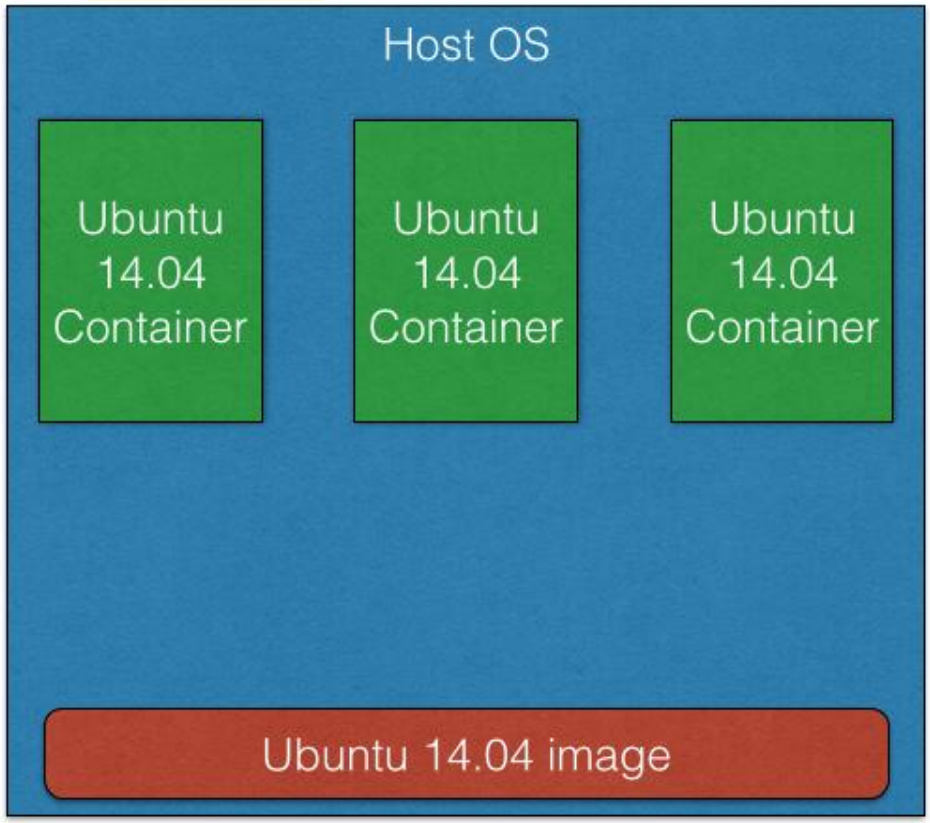
- Containers are the products of operating system virtualization.
- They provide a lightweight virtual environment that groups and isolates a set of processes and resources such as memory, CPU, disk, etc., from the host and any other containers.
- The isolation guarantees that any processes inside the container cannot see any processes or resources outside the container.



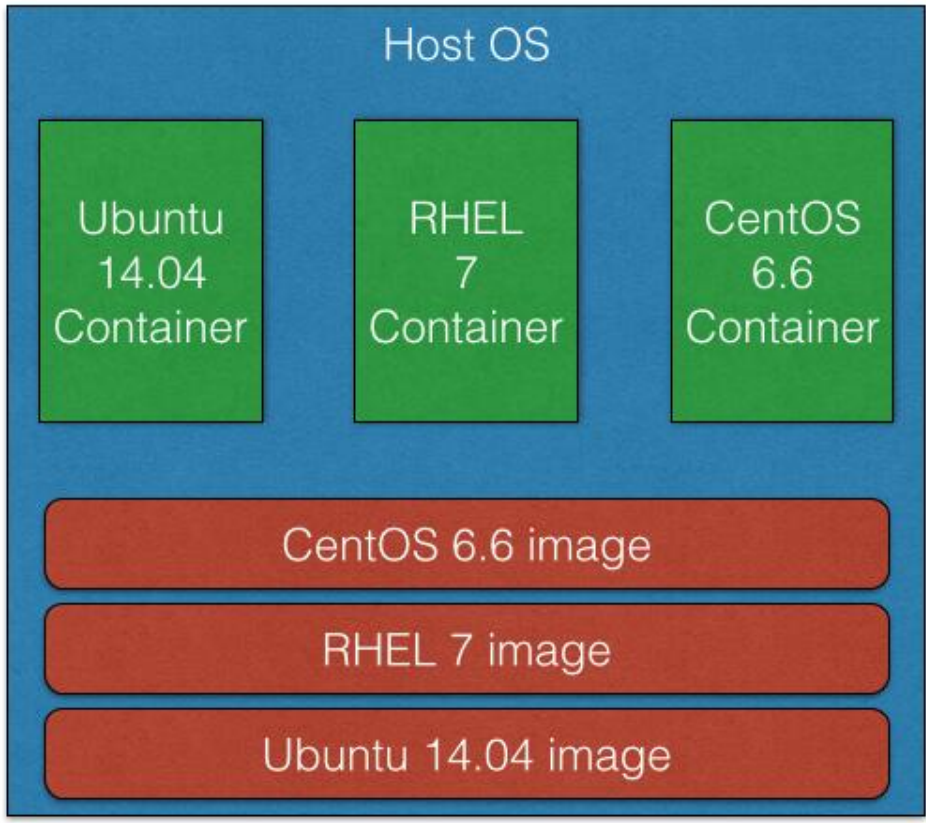
Operating System/Container Virtualization

Kernel sharing.

OS containers are virtual environments **that share the kernel** of the host operating system but provide user space isolation



Identical OS containers



Different flavoured OS containers

Process Isolation and Resource Control

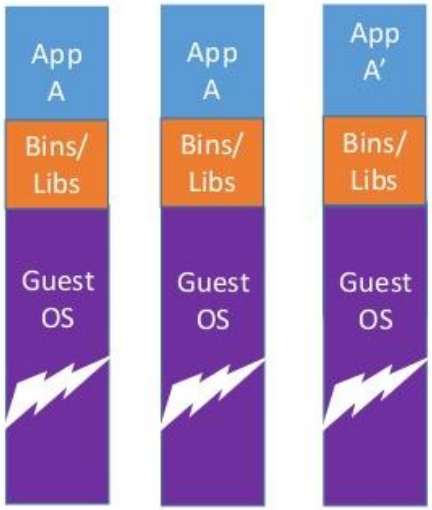
- Kernel features: namespaces and cgroups (defacto standard for creating linux containers)

- 6 types of namspaces for pre-process isolation: Network (NET), UTS(hostname), PROC(process id), MNT (mount), IPC and User (Security Seperation)
- Chroot, own root filesystem.

- NET own instance of network stack.
- cgroups subsystem provides resource management and accounting. (Google 2006 Paul Menage and Rohit Seth)

Why are Docker containers lightweight?

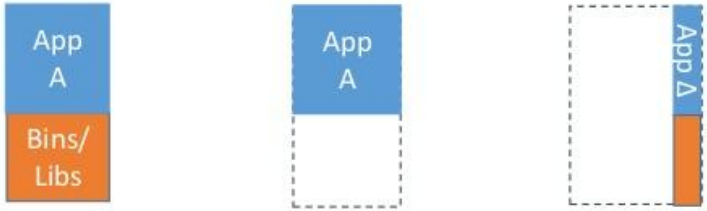
VMs



VMs

Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Containers



Original App

(No OS to take up space, resources, or require restart)

Copy of App

No OS. Can Share bins/libs

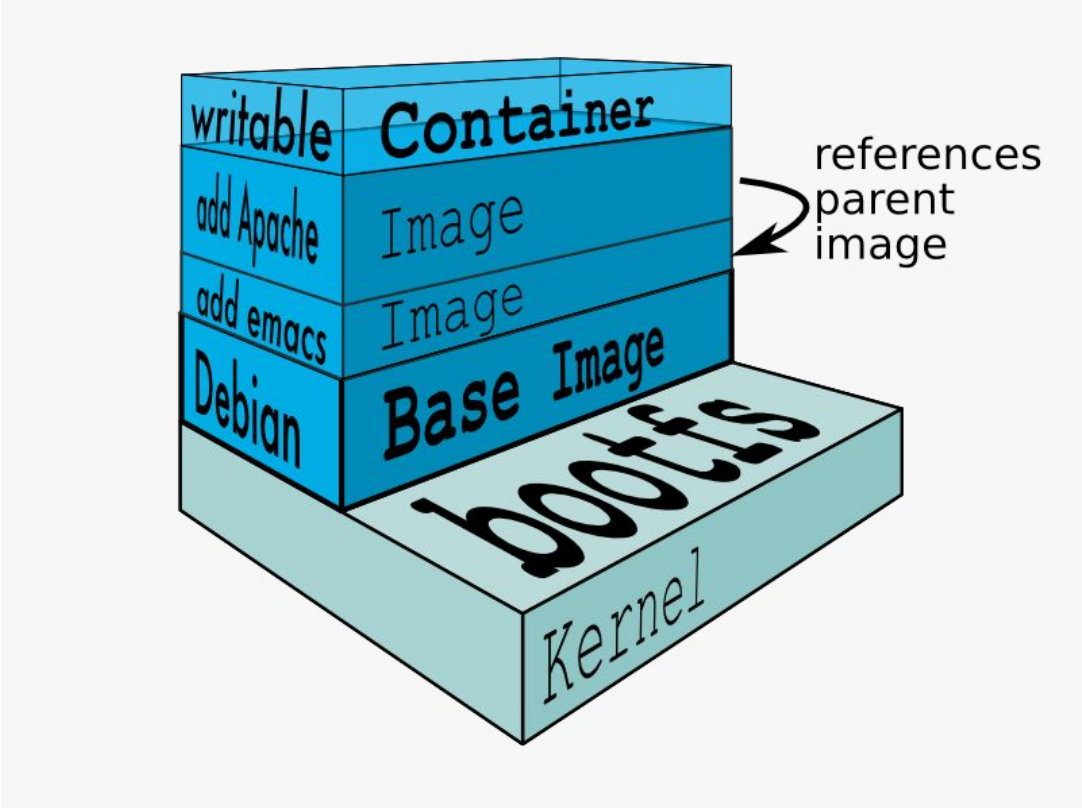
Modified App

Copy on write capabilities allow us to only save the diffs Between container A and container A'

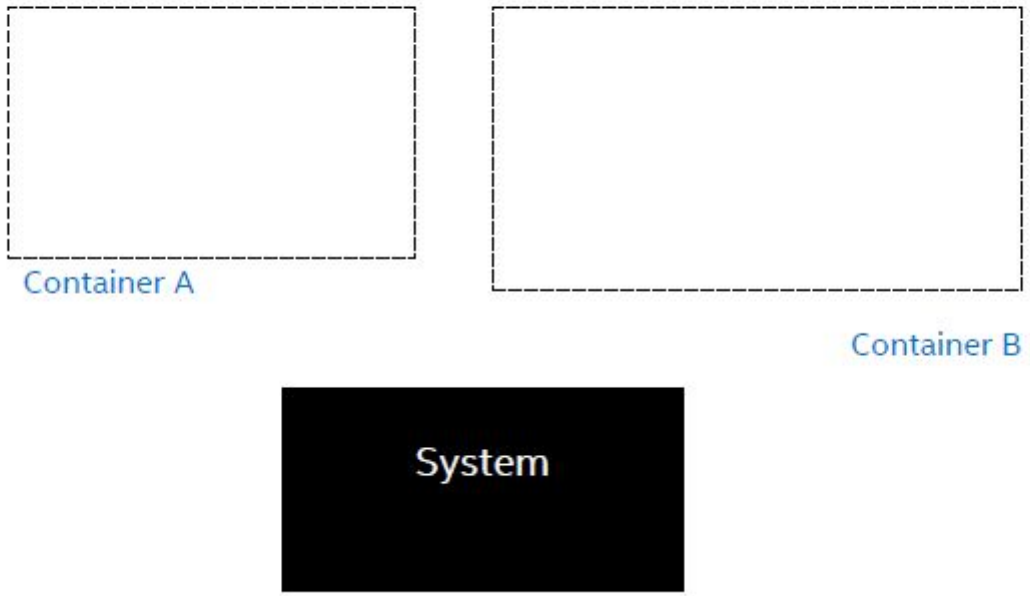
docker

Lightweight-> images system

Aufs (advanced multi layered unification filesystem) i.e. a system pointers/layers

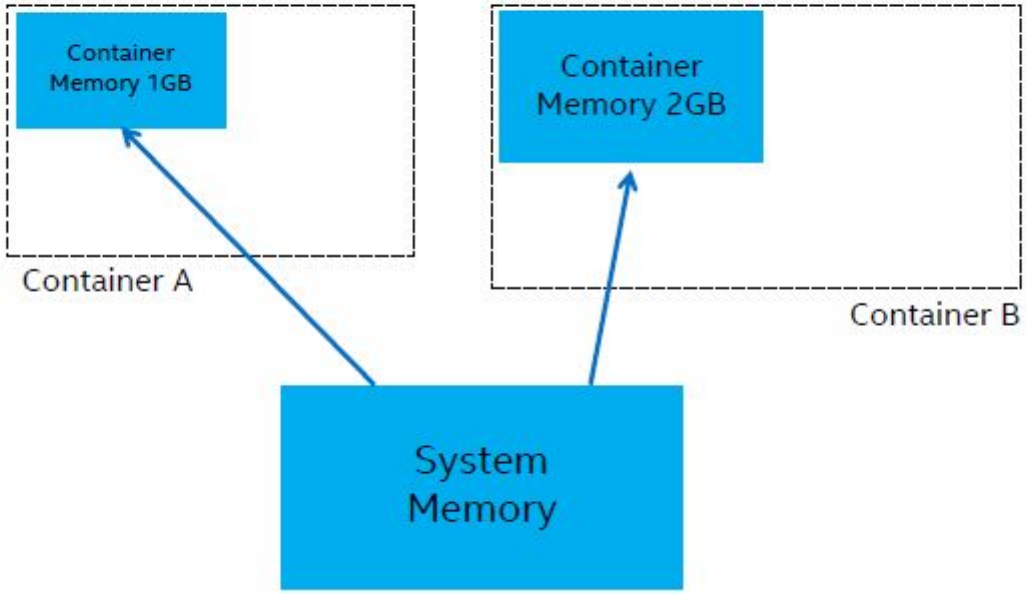


Process Isolation and Resource Control



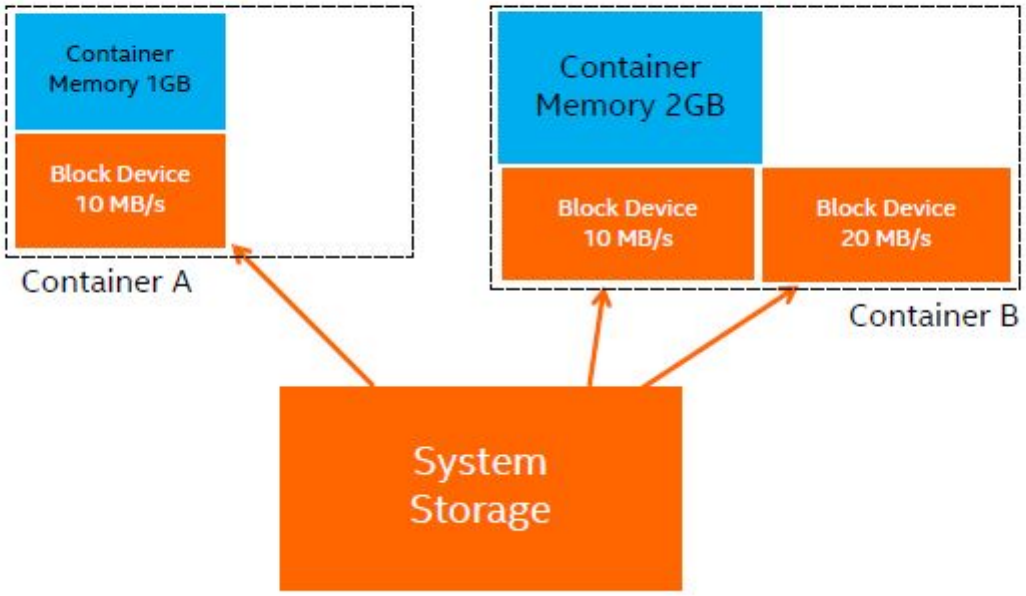
Process Isolation and Resource Control

Resource Limiting



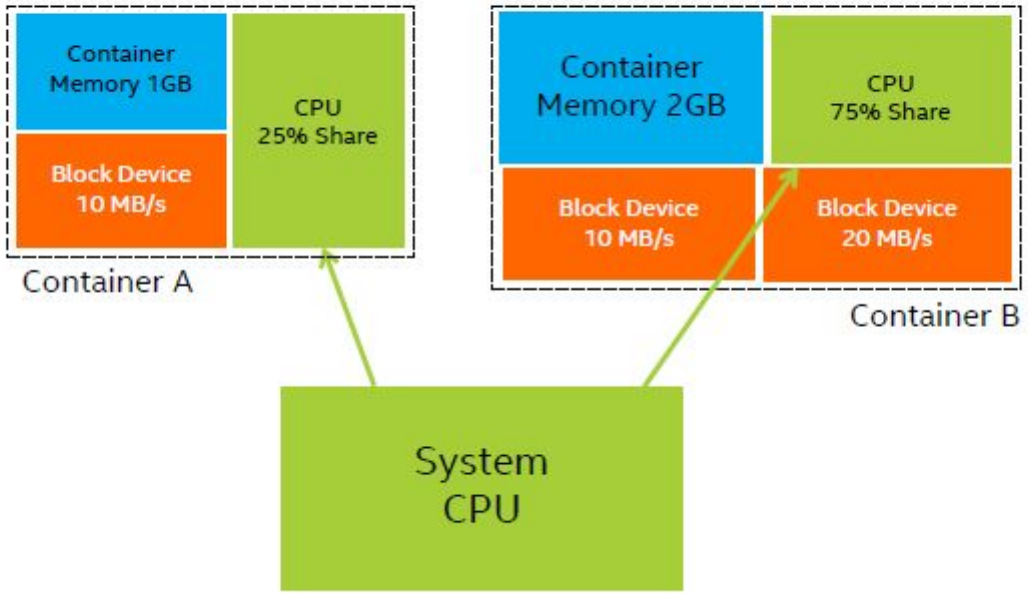
Process Isolation and Resource Control

Resource Limiting



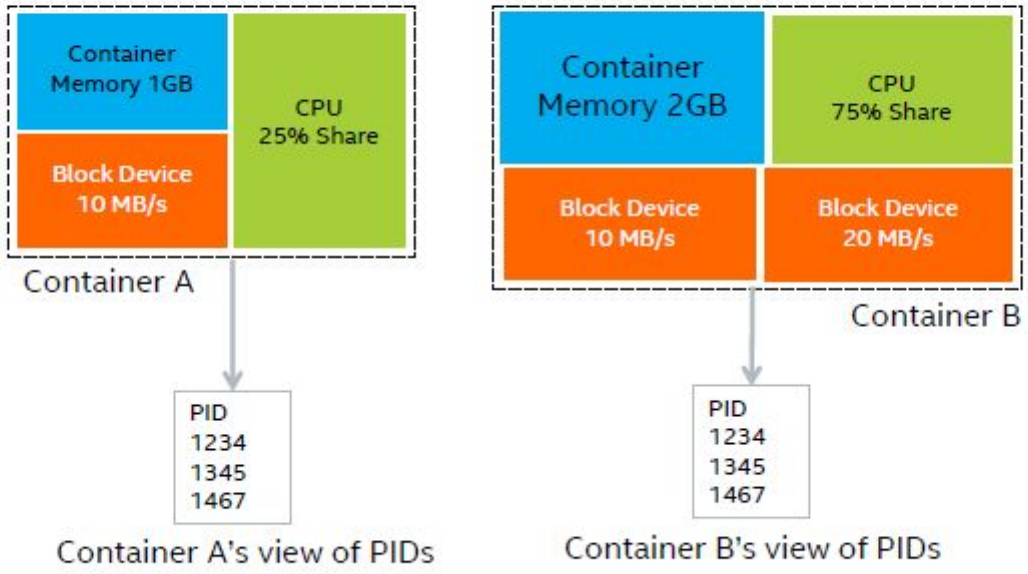
Process Isolation and Resource Control

Resource Limiting



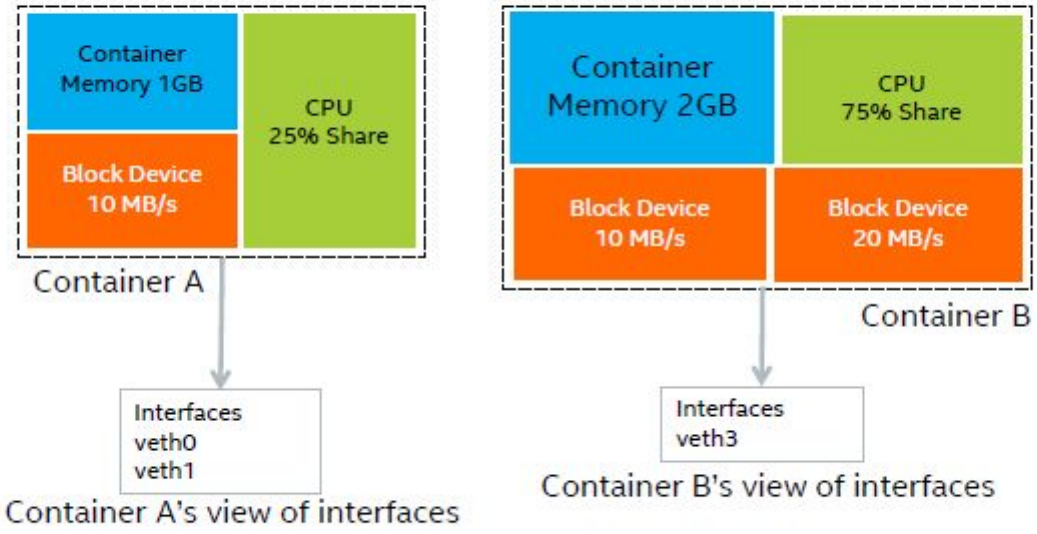
Process Isolation and Resource Control

Namespace Isolation – Processes



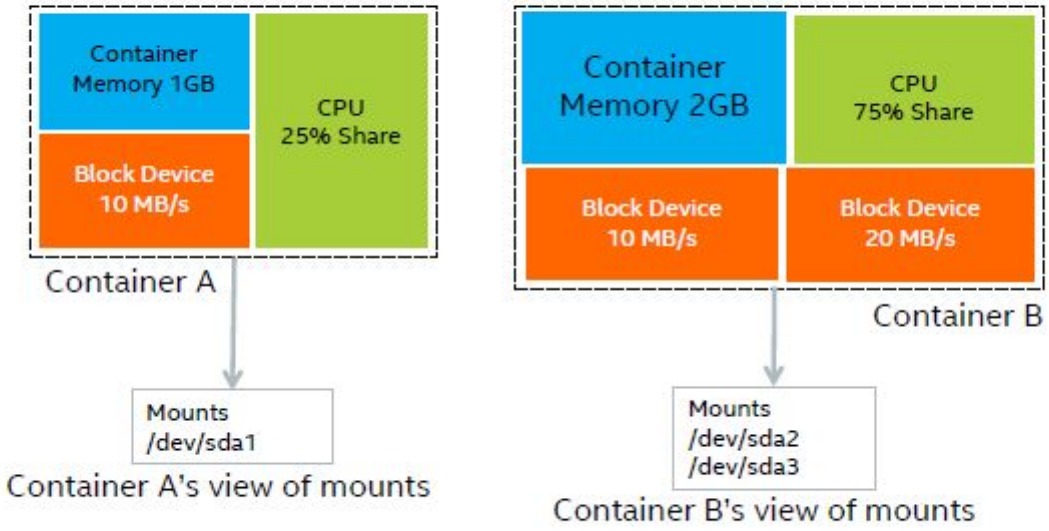
Process Isolation and Resource Control

Namespace Isolation - Networking



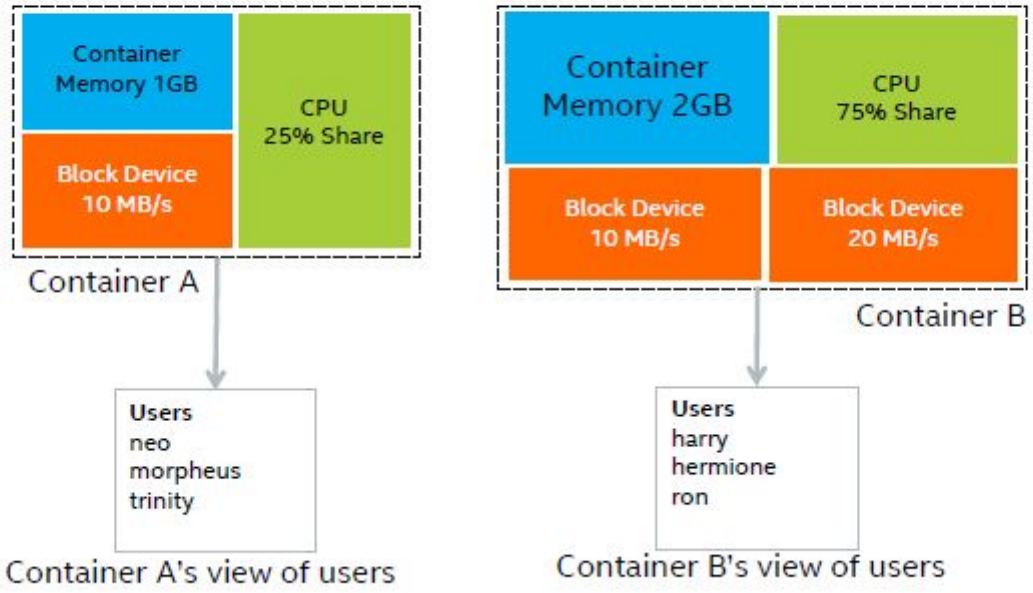
Process Isolation and Resource Control

Namespace Isolation – Mounts



Process Isolation and Resource Control

Namespace Isolation – Users

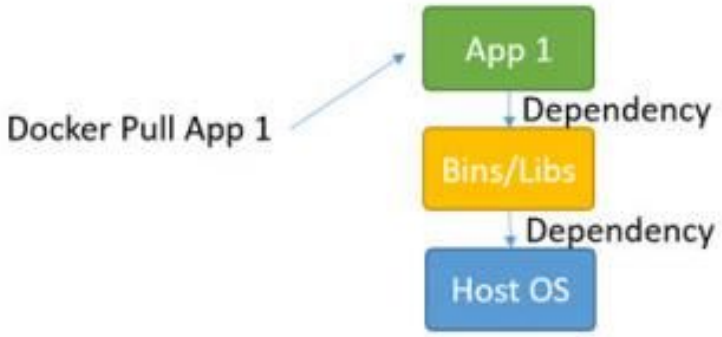


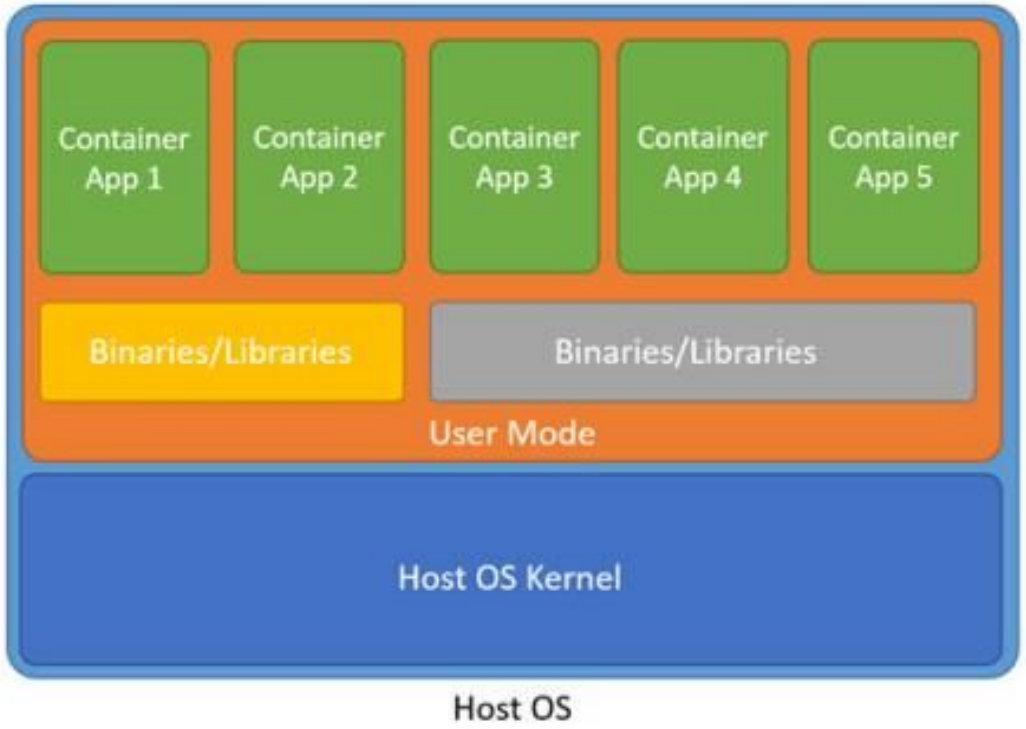
- **Affinity Linux/Windows**:
  - Can I run a windows container on a Linux kernel?

    No.
  - Can I run a Linux container on a windowd kernel?

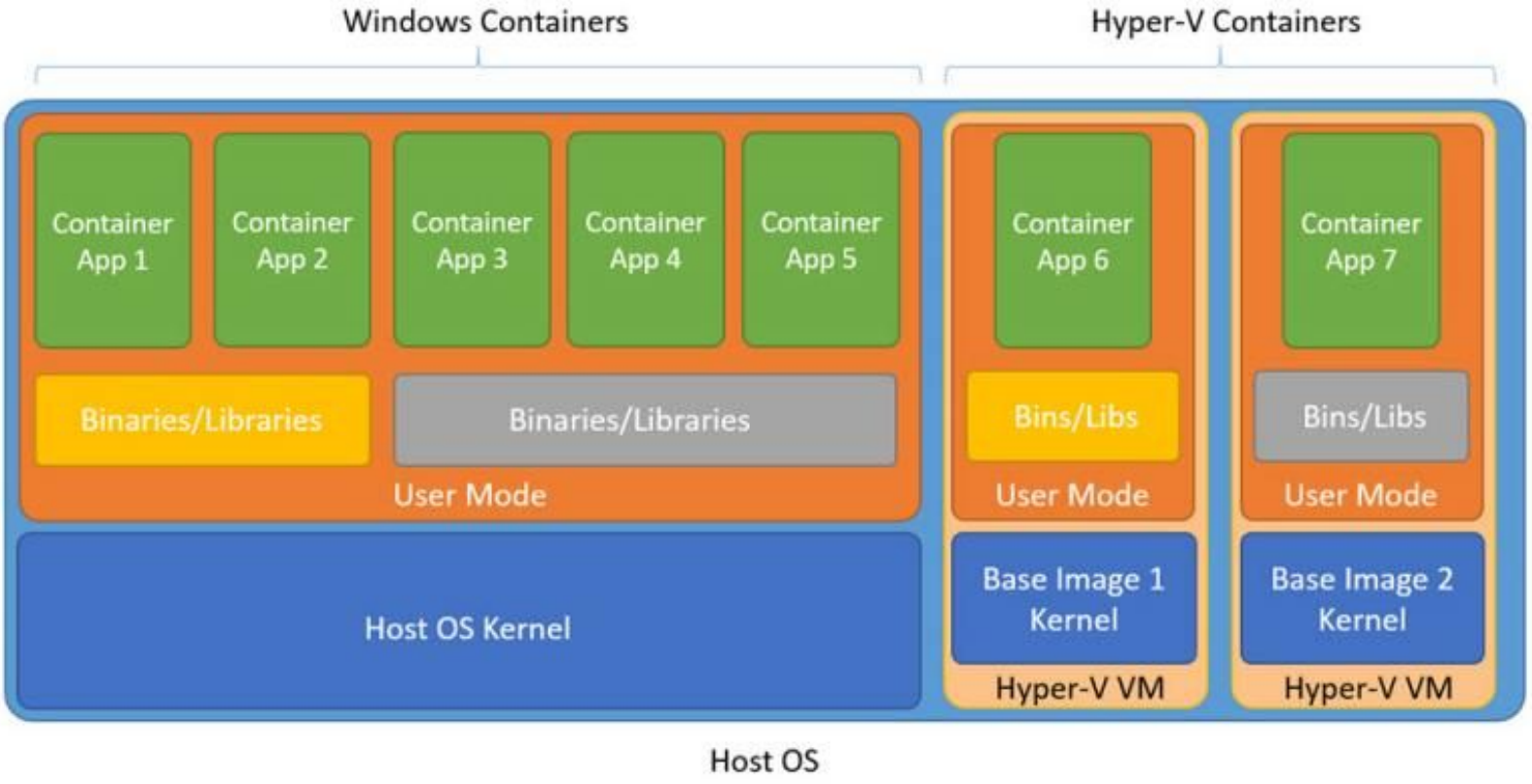
    no

- **Windows containers** work the same was as Linux containers. Each containerized application runs in its own user-mode, isolated container on a shared host operating system.
- The various dependencies are pulled for a Docker application.
- Note that different containers may use the same libraries.
- Multiple OSs on top of each other **are not supported** since they share a common kernel





- **Hyper-V containers:** Hyper-V isolation first creates a traditional VM, and then one or more containers can be deployed atop the operating system installed in the VM.
- The VM can be a different OS than the one used on the host or management VM.
- This approach provides more integrity by ensuring security between the VM and the other VMs, containers and the rest of the system. Organizations can use Hyper-V isolation in multi-tenant environments or in scenarios where containers are untested or untrustworthy.



Pros of containers

- **Pros**:
  - faster lifecycle vs. virtual machines;
  - ideal for homogenous application stacks on Linux;
  - faster boot;
  - Containers do not have the overhead of a hypervisor layer and because of this they gain the performance of the host it is running on.

Pros and Cons

- A container offers a virtualized environment without the overhead of a VM (shares OS resources)
- A Container offers an isolated unit to deliver a service
- A full virtualized system gets its own set of resources allocated to it, and does minimal sharing. You get more isolation, but it is much heavier (requires more resources).

Cons

- Currently much weaker security isolation than VMs
- Complexity?
- Not enough isolation since the isolation is at user-mode meaning a shared kernel. In a single tenant environment where applications can be trusted this is not a problem but in a multi-tenant environment a bad tenant may try to use the shared kernel to attack other containers.
- There is a dependency on the host OS version which may cause problems if a patch is deployed to the host which then breaks the application.

Past of Container Projects

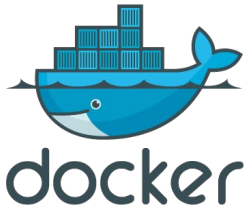
- OpenVZ: Virtuozzo (Swsoft 1997); opensource (2005); Parallels(2008); Basis of Parallels Cloud Server: CL tool available; runs on o a modified linux kernel; runs on generic kernel albeit reduced feature set; templates several distros.
- Google Containers (2013): opensource version of container stack lmctfy (Let Me Contain That For You); beta stage; only cgroups no namespaces;
- Linux-VServer: (2001) opensource; way to partition resources securely on a host. The host should run a modified kernel.

Container Management

- Warden: Developed by CloudFoundry as an orchestration layer to create application containers; controlling subsystems like linux cgroups, namespaces and security.

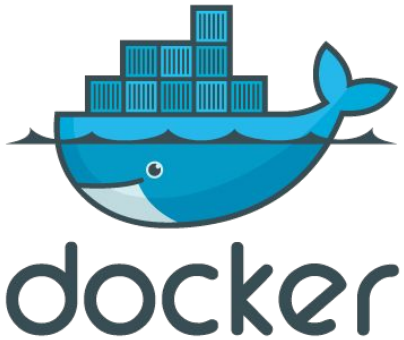
- Libcontainer: Written in Go programming language and developed by dotCloud/Docker it is a native Go implementation of “lxc-like” control over cgroups and namespaces
- Libvirt-lxc driver

Docker



- Docker is a very popular in the world of containers as a standard repository and management layer for the native container functionality found in Linux.
  - Open Source project that automates the creation and deployment of containers, Released March 2013.
  - PaaS project at (then) dotCloud (Docker Inc).
  - Python then Go
  - Kernel 3.8 or above
  - RHEL backported patches to 2.6.32
  - Frontend for the LXC toolkit (docker.io or –io)

Docker



- From Windows Server 2016 brings containers to Windows Server and integrates with Docker for the repository and management.
- Starting Windows Server 2016 comes with two types of containers:Windows containers and Hyper-V containers.

Docker

- Docker images can be stored on a public repository and can be downloaded
- Changes you make in a container are lost if you destroy the container, unless you commit your changes (much like you do in git)
- These images can be uploaded to a public registry
- Just like a Makefile will compile code into a binary executable, a Dockerfile will build a ready-to-run container image from simple instructions.

Why we need Docker?

- Deploying a consistent production environment
- Even with tools (chef and puppet)
- OS updates etc that change between hosts and environments.
- Docker does is it gives you the ability to snapshot the OS into a common image, and makes it easy to deploy on other docker hosts.
- Locally, dev, qa, prod, etc, all the same image.

Demos

- Virtualenv
- AWS+Docker container+Jupyter Notebook

The differences between Windows Containers and Hyper-V Containers in Windows Server 2016

ITProToday John Savill | Aug 04, 2015