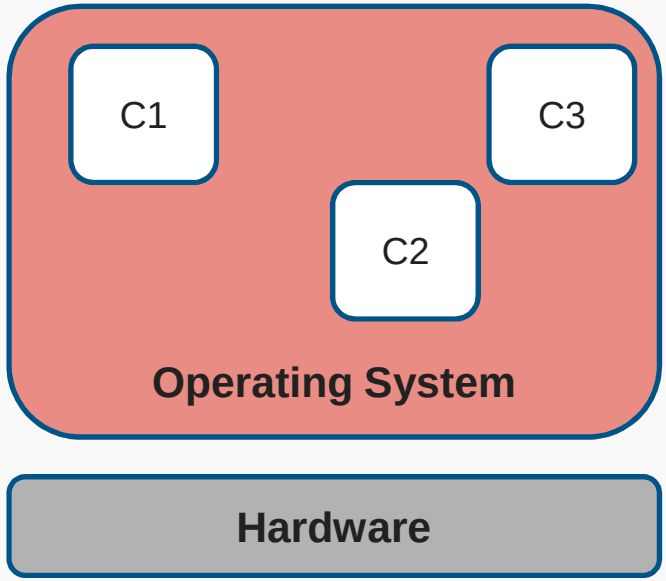
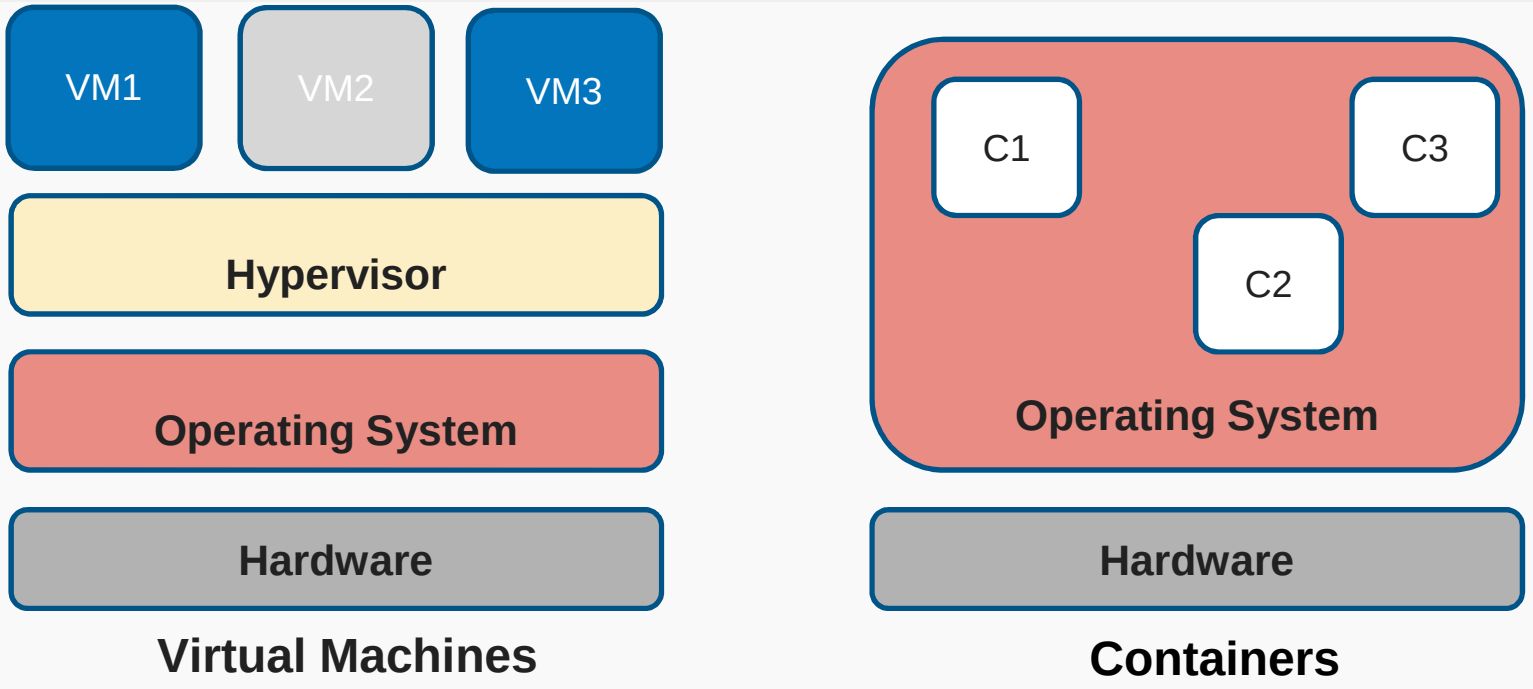
System-level of Operating Virtualization

What are containers?

- Server-virtualization
- Multiple isolated user-space instances (instead of one) on a single kernel of the operating system
- Separate out a user instance from another
- Also called software containers, jails



How are they different from VMs?



How are they different from VMs?

Virtual Machines	Containers
• Abstracts the hardware	• Abstracts the operating system
• Different guest operating Systems	• Single operating system
• Heterogeneous (high versatility)	• Homogeneous
• Low Density	• High Density
• Overhead as it is not light weight	• Less overhead as it is light weight

Advantages over whole-systemvirtualization

As all containers use the same OS underneath, it leads to the following benefits:

- They use less CPU and Memory for running the same workloads as virtual machines.
- The time to initiate a container is smaller as compared to VMs.
- On a machine, we can have many have many more user containers (~100) as compared to a small number of user VMs (~10)

Limitations

- **Less Flexibility**: Cannot host a guest OS different from the host, or a different guest kernel
- **Security**: As containers run on top of same OS, security issues exist from adjacent containers

Use Cases

- Virtual hosting environments, where it is useful for securely allocating finite hardware resources amongst a large number of mutually-distrusting users.
- Operating-system-level virtualization implementations capable of live migration can also be used for dynamic load balancing of containers between nodes in a cluster.
- For consolidating server hardware by moving services on separate hosts into containers on the one server.

Features provided by OS-level virtualization

Isolation

- File system isolation
- Copy on Write
- Network isolation
- Root privilege isolation

Features provided by OS-level virtualization

Resource Management

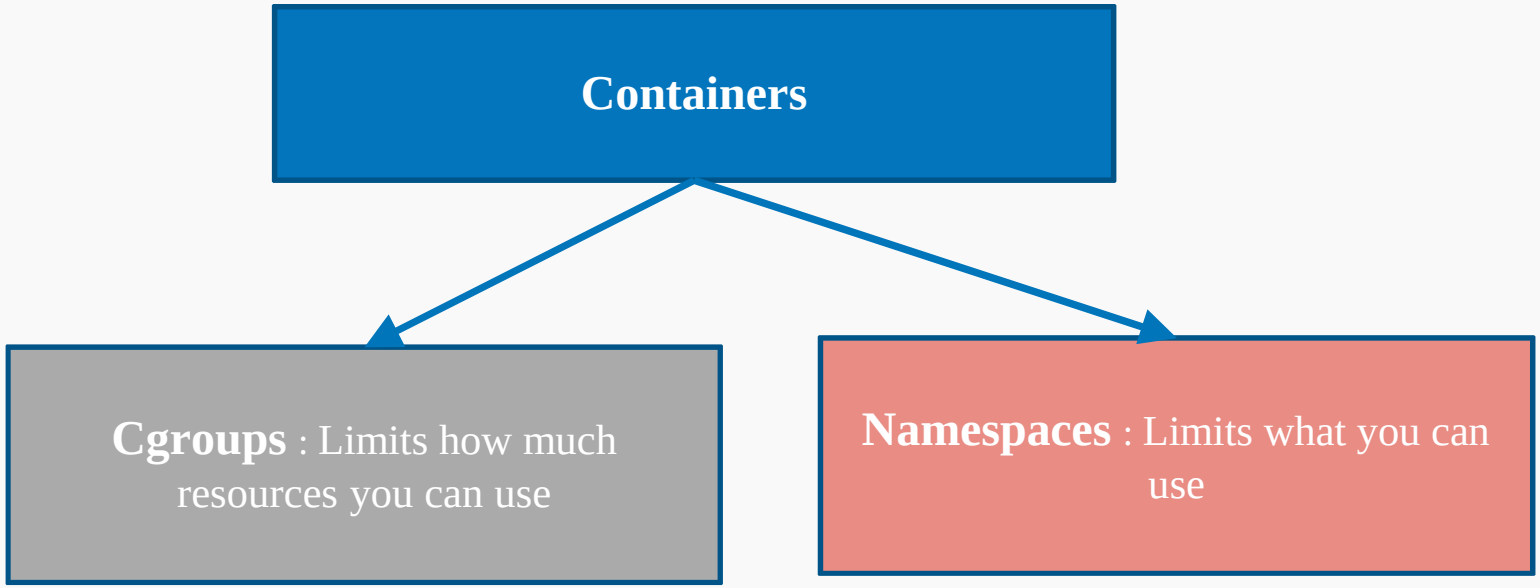
- Disk Quota
- I/O rate limiting
- Memory limits
- CPU quota

Features provided by OS-level virtualization

Other features

- Nested Virtualization
- Partition Check pointing
- Live Migration

How are containers implemented?



Control Groups (cgroups)

- It is a Linux kernel feature that lets a group of processes be bound by the same criteria and associated with a set of limits.

Features

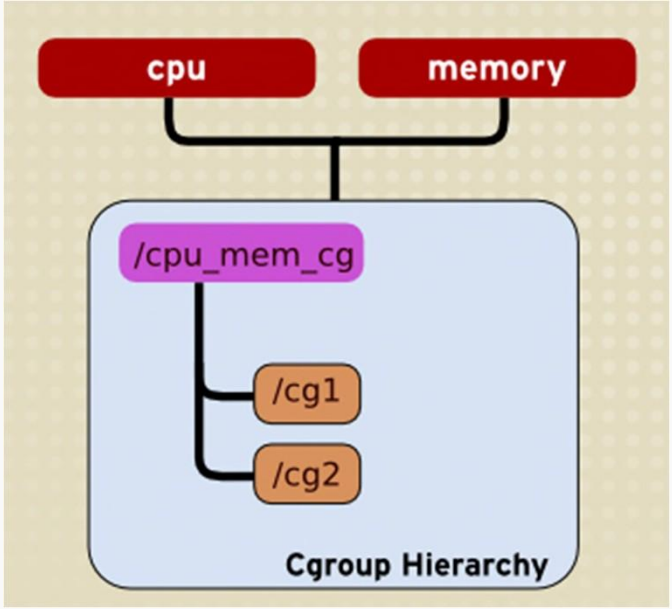
- Allocate and Limit resources
- Prioritize resources
- Monitor resource utilization
- Control groups of processes

Resources

- CPU
- Memory
- Network bandwidth
- Disk I/O

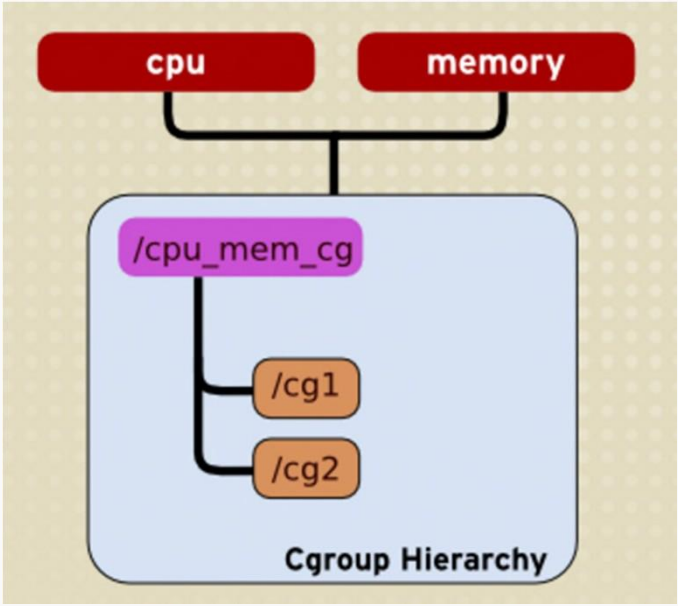
CGroup Model

- Multi-Hierarchy model
- Child cgroups inherit attributes from parent
- Each hierarchy attached to one or more subsystems or resources
- Subsystems → CPU, Memory, Disk I/O
- Config File → /etc/cgconfig.conf



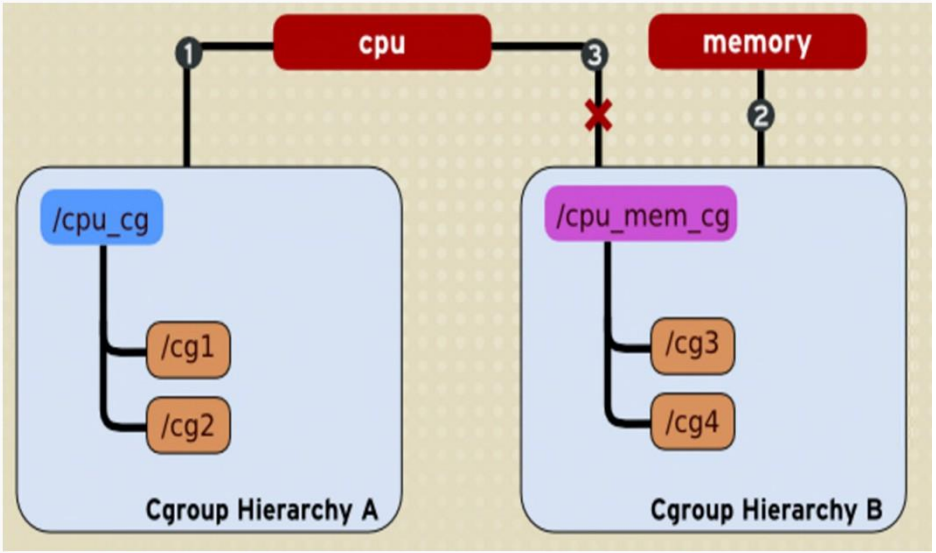
CGroup Hierarchy Rules (Rule 1)

- Multiple resources can be binded to single hierarchy
- Allows for consolidation of multi-resource specification under one control group
- Eg: CPU and Memory subsystems have same cgroup hierarchy



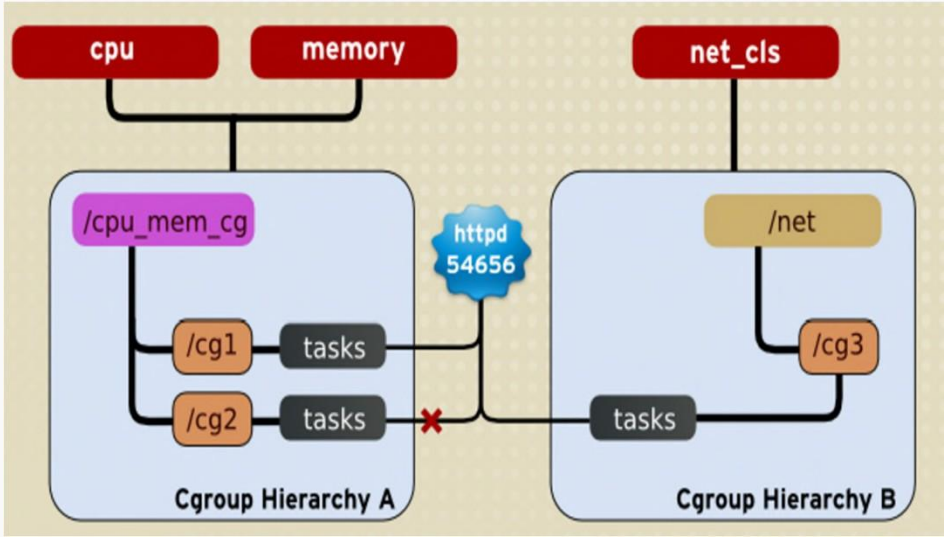
CGroup Hierarchy Rules (Rule 2)

- Single subsystem cannot be attached to more than one hierarchy,if one of hierarchy is shared
- Eg: cpu subsystem cannot be attached to cpu_mem_cg hierarchy



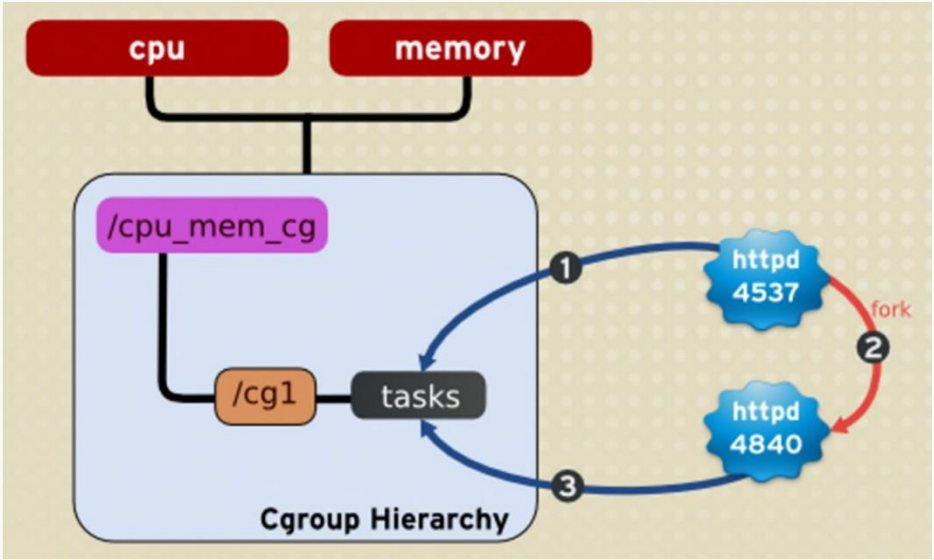
CGroup Hierarchy Rules (Rule 3)

- Tasks cannot be a member of two different cgroups in the same hierarchy
- Single task can be member of cgroups in multiple hierarchy
- Eg: “httpd 54656” cannot exist in different cgroups under same hierarchy.



CGroup Hierarchy Rules (Rule 4)

- Child tasks inherit cgroup membership of parent task
- Child tasks can be moved to different cgroups independent of parent
- Eg: Task “httpd 4537” forks child task “httpd 4840” which inherits its membership of /cpu_mem_cg/cg1



Namespaces

- Wraps system resources and presents it to the processes within the namespace
- Provides processes their own view of the system
- Multiple Namespaces: pid, net, mnt, uts, ipc, user

chroot

- Operation that allows users to change the apparent root of the file system.
- Restricts the program to access files outside the root directory.

Pros:

- Isolating insecure or unstable applications.
- For supporting legacy software.
- Testing new packages before deployment in production setting.
- Dependency control for new software development.

Limitations:

- Users with root permissions can escape by second chroot.
- Do not intend to restrict the resources like CPU, disk space or I/O.

Types of Containers

OS containers:

- Meant to be used as OS - run multiple services
- Built on cgroups, namespaces, native process resource isolation
- Examples - LXC, BSD Jails, OpenVZ

Application containers

- Meant to run for a single service
- Built on top of OS container technologies

Examples - Docker, Rocket

FreeBSD Jail (2000)

- Jails built upon chroot.
- Jail is a virtualization with its own files, processes, users and superusers.
- Jails are isolated from each other and rest of the system.
- Enables running different versions of softwares in isolation.

Limitations:

- Processes cannot interact across the jail boundaries.
- Kernel cannot be modified by direct access and loading modules.
- Jail is bounded to access a set of IP addresses.
- Mounting and unmounting of file systems not possible.

OpenVz(2005)

- Each container is a separate entity and contains its own files, users and groups, process tree, network, devices.
- Four resources: Two level disk quota, CPU Scheduler, I/O Scheduler, User Bean counter
- Check pointing and Live migration feature

Limitations:

- OpenVz restricts container access to real physical devices and therefore container is hardware dependent

Linux Containers (LXC) (2008)

- LXC is a userspace interface for the Linux kernel containment features.
- Allow processes to have their own private view of the operating system with their own process ID (PID) space, file system structure and network interfaces.
- Based on Linux resource management and isolation features:
  - Linux Kernel Cgroups.
  - Namespace Isolation.
  - Mandatory access control.

Linux Containers (LXC) (2008)

- Motivation : User space isolation as close to VM with much less overhead. (Lightweight alternative to virtual machines.)
- Method for running multiple Linux systems (containers) on a single host.
- Provides powerful APIs to allow Linux users to easily create and manage system or application containers.
- Provides tools to manage containers, advanced networking and storage support.

Linux Containers (LXC) (2008)

- Installation : `apt-get install lxc`
- Create a container : `lxc-create -t <template> -n <container name>`
- List of containers: `lxc-list`
- Start a container : `lxc-start -n <container name>`
- Launch container in background: `lxc-start -d -n <container name>`
- Attach to container console: `lxc-console -n <container name>`

Linux Containers (LXC) (2008)

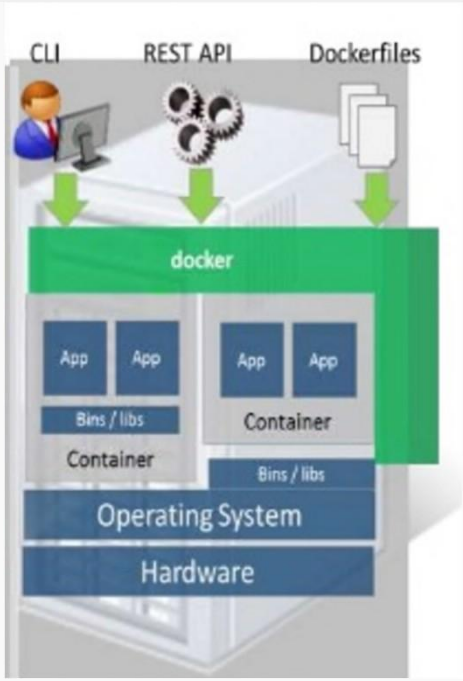
Limitations

- All LXC containers run inside host system's kernel.
- Allows only Linux "guest" operating systems.
- No secure barrier for the processes from the host system, nor from other containers.

LXD (2015)

- More advanced and secured system built on top of LXC.
- Maximum density of guests per host with virtually no overhead
- Containers controllable over network through REST APIs
- Features:
  - Secure
  - Scalable
  - Intuitive
  - Image Based
  - Live Migration

User Containers vs OS containers



Docker (2013)

- A very popular application container.
- Docker provides lightweight containers to run applications in isolation.
- Completely simplifies the process to manage and create new containers in distributed systems.
- Rapid deployment of distributed multi-component system
- Reduce the container to a single process and manage that through an API.

Rocket (2013)

- Consists of two tools:
- Actool – Administers the building of containers, handles container validation
- Rkt – helps in fetching and running container images

- Unlike Docker (which has a central daemon), Rocket starts execution under the process that started it.

Thank you