Linux Containers

Darko Bozhinoski,
Ph.D. in Computer Science
Email: D.Bozhinoski@innopolis.ru

What is a container?

Containers

A set of processes that are isolated from the rest of the system.

What are containers?

Container “feels” like VM:

- Process space
- Network interface
- Independent filesystem
- ...

What are containers?

Container “feels” like VM:

- Process space
- Network interface
- Independent filesystem
- ...

Isn’t this just virtualization?

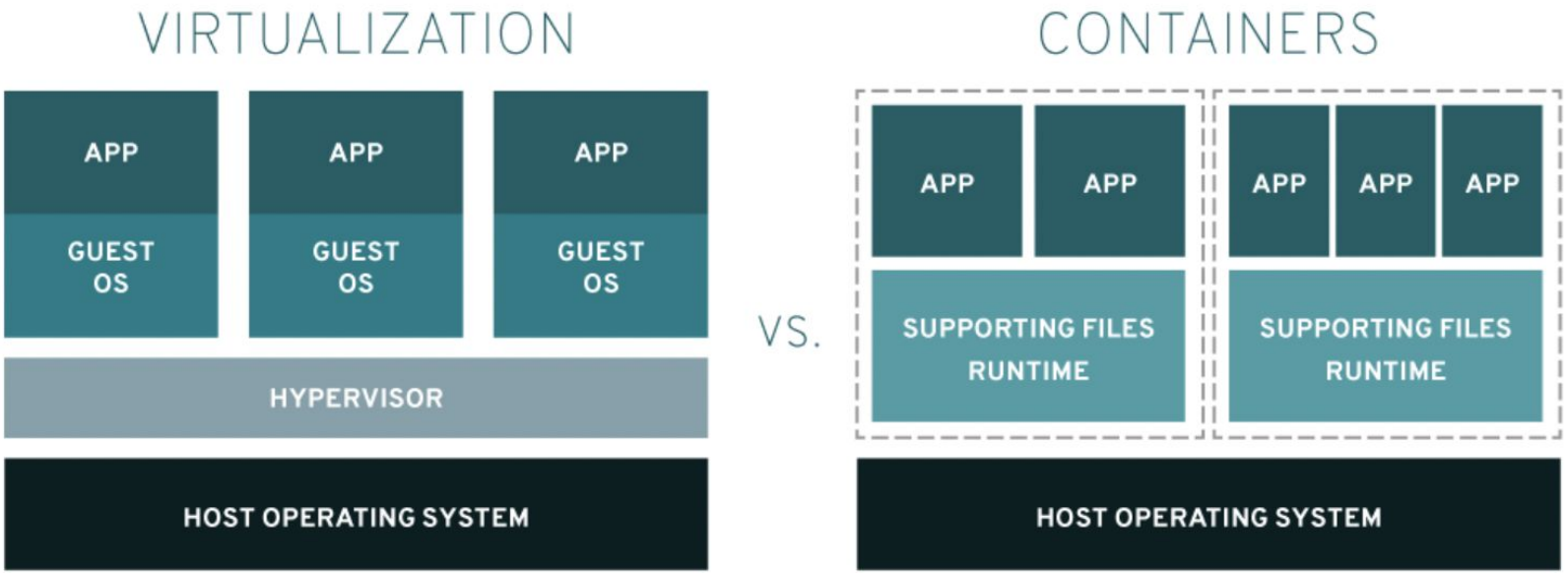
Containers vs. VMs?

NOT EXACTLY.

Think of them more as complementary of one another. Here's an easy way to think about the 2:

- Virtualization lets your operating systems run simultaneously on a single hardware system.
- Containers share the same operating system kernel and isolate the application processes from the rest of the system. For example: ARM Linux systems run ARM Linux containers, x86 Linux systems run x86 Linux containers, x86 Windows systems run x86 Windows containers. Linux containers are extremely portable, but they must be compatible with the underlying system.

Containers vs. VMs?

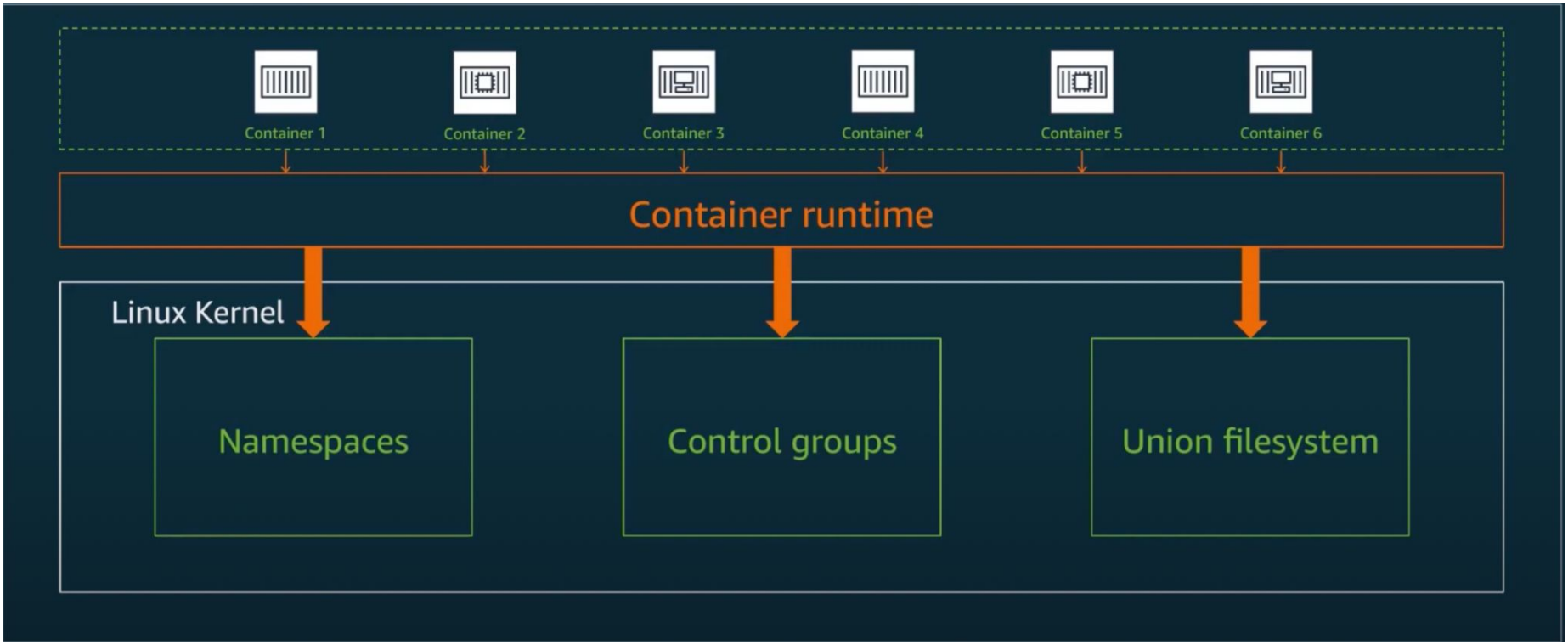


Linux Containers (LXC)

"Linux Containers" is a feature to contain a group of processes in an independent execution environment.

- Linux kernel provides an independent application execution environment for each container including:
  - Independent filesystem
  - Independent network interface and IP address.
  - Usage limit for memory and CPU time.
- Linux containers are realized with integrating many existing Linux features. There are multiple container management tools such as lxctools, libvirt and docker. They may use different parts of these features.

LXC Primitives



Enabling Technology

Enabling Technology in Linux has been present for many years

- Namespaces
- cgroups - Control Groups

Namespaces - limits how much you can see

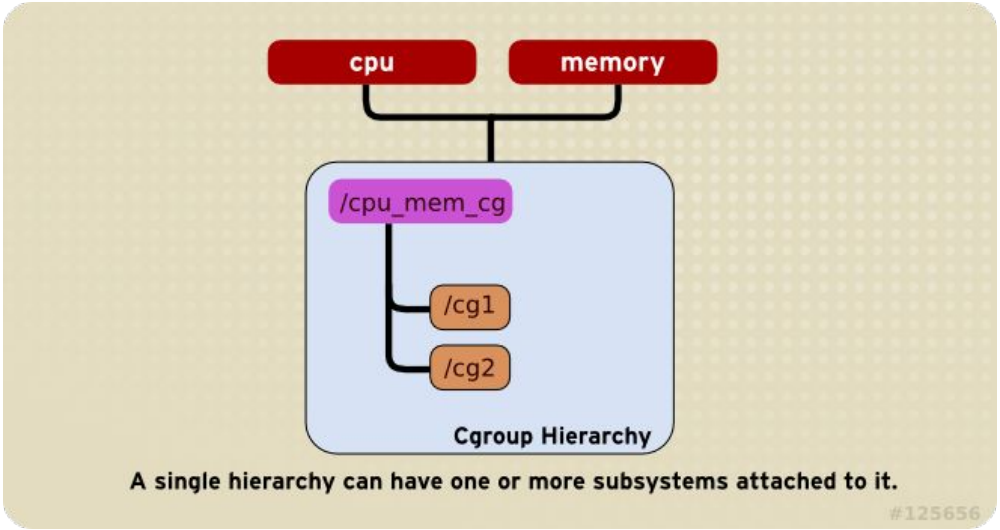
Cgroups - limits how much resources you can use

Control Groups (Cgroups)

- Organize all processes in the system
- Account for resource usage and gather utilization data
- Limit or prioritize resource utilization

Subsystems

- Cgroup system is an abstract framework
- Subsystems are concrete implementations
- Different subsystems can organize processes separately
- Most subsystems are resource controllers



Examples: CPU, Memory, Block I/O

Hierarchical representation

Independent subsystem hierarchies

Every **pid** is represented exactly once in each subsystem

Cgroup virtual filesystem

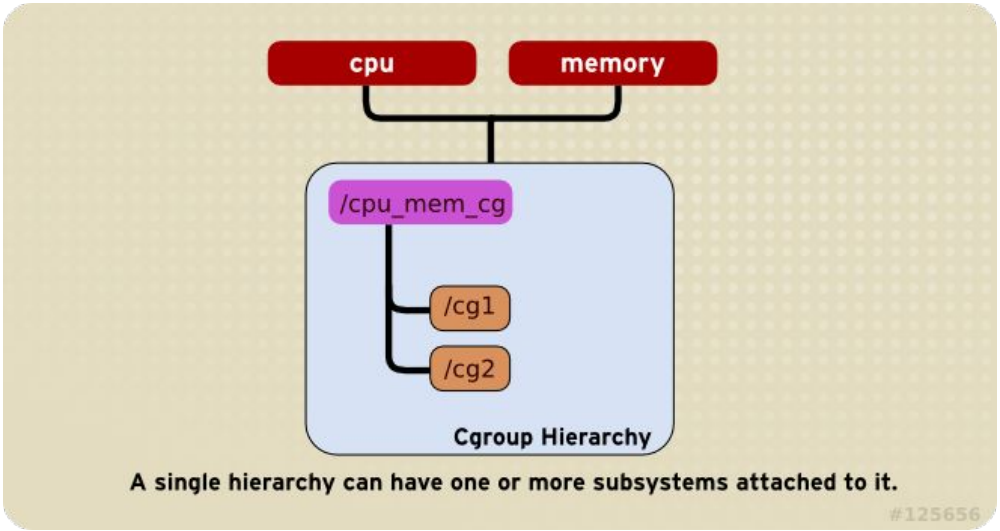
- Mounted at /sys/fs/cgroup
- Tasks virtual file holds all pids in the cgroup

Resource Controllers

- **Blkio** — this subsystem sets limits on input/output access to and from block devices such as physical drives (disk, solid state, USB, etc.).
- **CPU** — this subsystem uses the scheduler to provide cgroup tasks access to the CPU.
- **CPUACCT** — this subsystem generates automatic reports on CPU resources used by tasks in a cgroup.
- **CPUSET** — this subsystem assigns individual CPUs (on a multicore system) and memory nodes to tasks in a cgroup.
- **DEVICES** — this subsystem allows or denies access to devices by tasks in a cgroup.
- **FREEZER** — this subsystem suspends or resumes tasks in a cgroup.
- **MEMORY** — this subsystem sets limits on memory use by tasks in a cgroup, and generates automatic reports on memory resources used by those tasks.
- **NET_CLS** — this subsystem tags network packets with a class identifier (classid) that allows the Linux traffic controller (tc) to identify packets originating from a particular cgroup task.
- **NET_PRIO** — this subsystem provides a way to dynamically set the priority of network traffic per network interface.
- **NS** — the namespace subsystem.

Relationships Between Subsystems, Hierarchies, Control Groups and Tasks

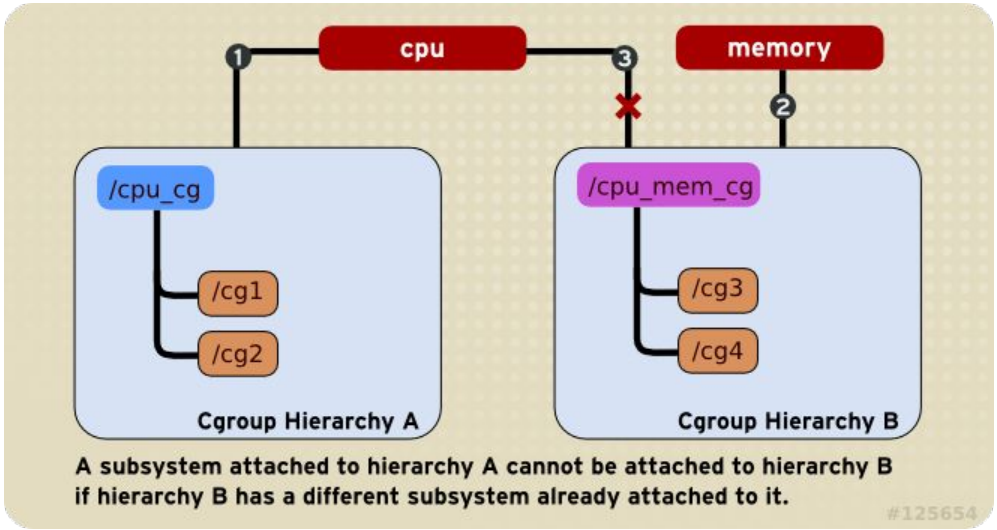
Rule 1. A single hierarchy can have one or more subsystems attached to it.
As a consequence, the `cpu` and `memory` subsystems (or any number of subsystems) can be attached to a single hierarchy, as long as each one is not attached to any other hierarchy which has any other subsystems attached to it already



Relationships Between Subsystems, Hierarchies, Control Groups and Tasks

Rule 2. Any single subsystem (such as `cpu`) cannot be attached to more than one hierarchy if one of those hierarchies has a different subsystem attached to it already.

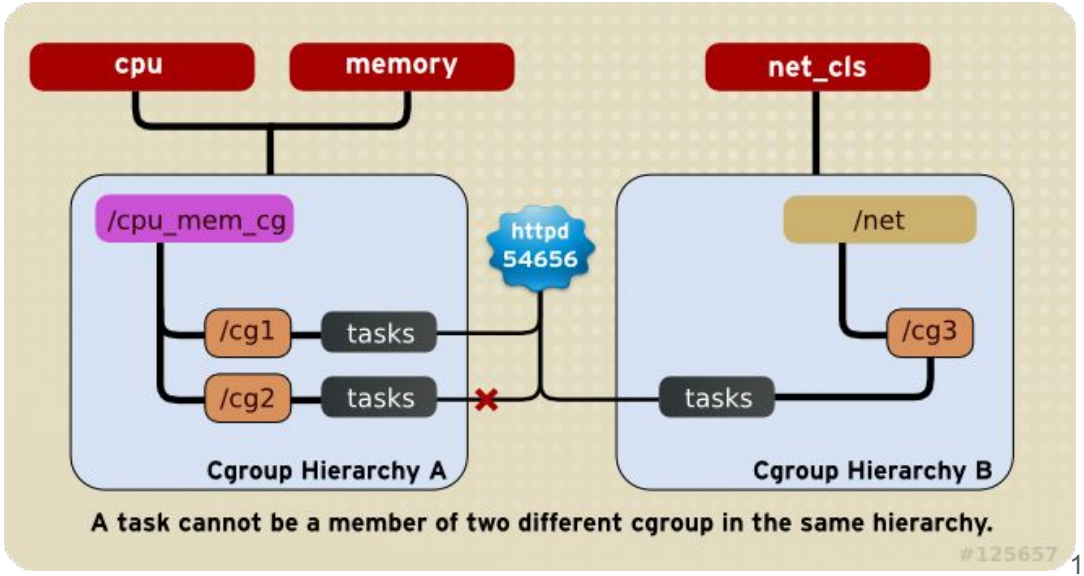
As a consequence, the `cpu` subsystem can never be attached to two different hierarchies if one of those hierarchies already has the `memory` subsystem attached to it.



Relationships Between Subsystems, Hierarchies, Control Groups and Tasks

Rule 3. Each time a new hierarchy is created on the systems, all tasks on the system are initially members of the default cgroup of that hierarchy, which is known as the *root cgroup.*

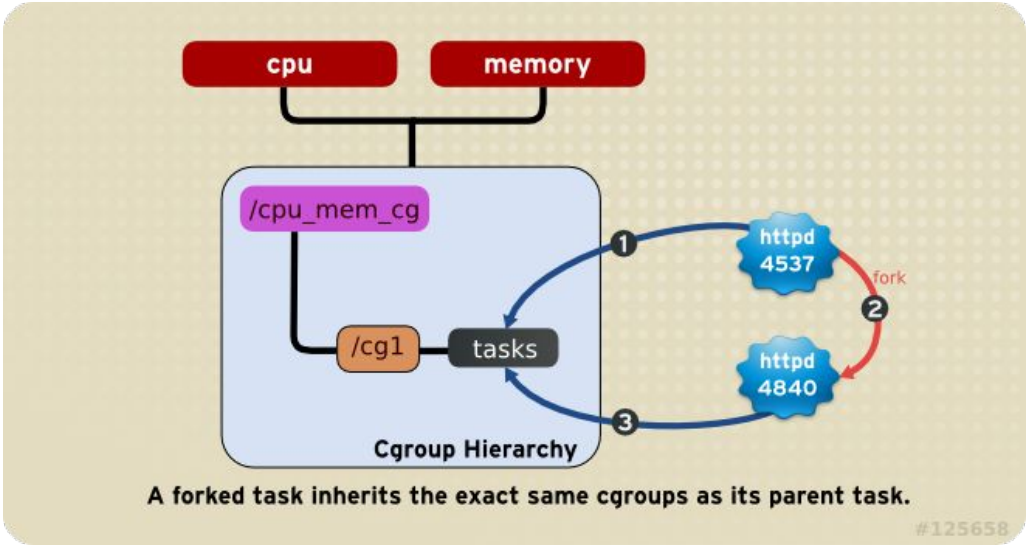
As a consequence, if the `cpu` and `memory` subsystems are attached to a hierarchy named `cpu_mem_cg`, and the `net_cls` subsystem is attached to a hierarchy named `net`, then a running `httpd` process could be a member of any one cgroup in `cpu_mem_cg`, and any one cgroup in `net`.



Relationships Between Subsystems, Hierarchies, Control Groups and Tasks

Rule 4. Any process (task) on the system which forks itself creates a child task. A child task automatically inherits the cgroup membership of its parent but can be moved to different cgroups as needed.

As a consequence, consider the `httpd` task that is a member of the cgroup named `half_cpu_1gb_max` in the `cpu_and_mem` hierarchy, and a member of the cgroup `trans_rate_30` in the `net` hierarchy. When that `httpd` process forks itself, its child process automatically becomes a member of the `half_cpu_1gb_max` cgroup, and the `trans_rate_30` cgroup. It inherits the exact same cgroups its parent task belongs to.



Namespaces

Namespaces are a kernel feature that allow processes a virtual view of isolated system resources. Provide process with their system view. In other words namespaces is the way to represent and do not collide with namings between processes within one host system.

What do namespaces do?

- Isolation mechanism for resources
- Changes to resources within the namespace can be invisible outside the namespace
- Resource mapping with permission changes

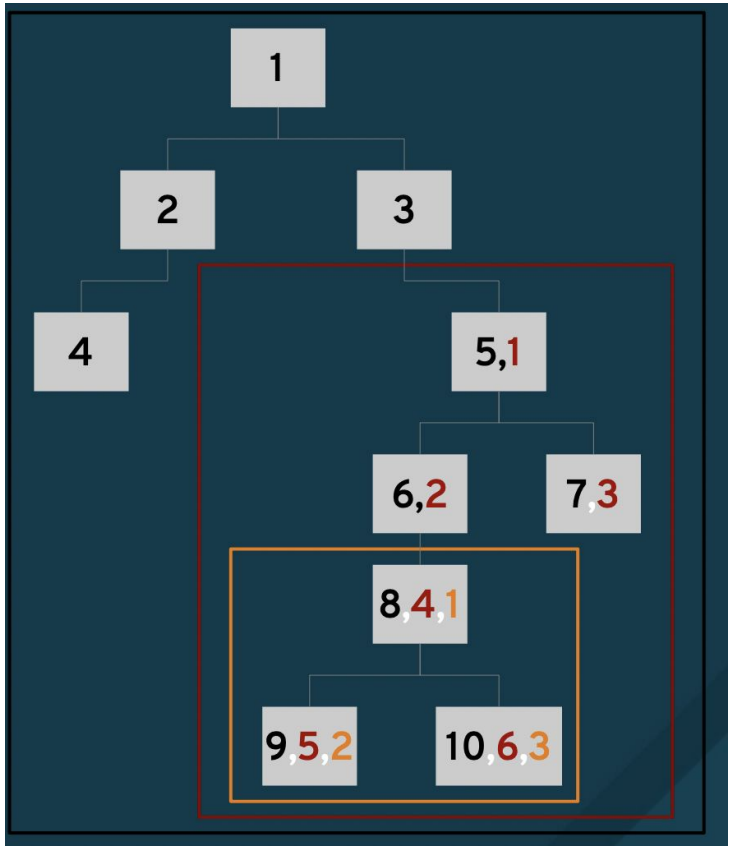
Available namespaces

- Process (pid)
- Network (net)
- Filesystem (mnt)
- User
- Interprocess (ipc)
- UNIX Technology Services (utc)

Process Namespaces

- Original UNIX Process Tree
  - First process is PID 1
  - Process tree rooted at PID 1
  - PIDs with appropriate privilege may inspect or kill other processes in the tree

- Linux Namespaces
  - Multiple, nested process trees
  - Nested trees cannot see parent tree
  - Process has multiple PIDs
  - One for each namespace it is a member of



Procfs virtual filesystem

- All of these processes are tracked in a special file system called procfs.
- mounted under /proc. If you do a listing of /proc, you will see a folder for every process currently running on your system.

Creating new PID Namespace

Fork() creates a new process by duplicating the calling process.

The new process is referred to as the child process.

The calling process is referred to as the parent process.
(https://man7.org/linux/man-pages/man2/fork.2.html)

Clone () is for new processes to create new namespaces. By contrast with fork(), it provides more precise control over what pieces of execution context are shared between the calling process and the child process. (https://man7.org/linux/man-pages/man2/clone.2.html)

Unshare() is for running processes to create and move itself into a new namespace. (https://man7.org/linux/man-pages/man1/unshare.1.html).

By default, a new namespace persists only as long as it has member processes. A new namespace can be made persistent even when it has no member processes by bind mounting /proc/pid/ns/type files to a filesystem path.

CLONEW_NEW* flags to specify which namespaces

Network Namespaces

Presents an entirely separate set of network interfaces to each namespace.

Each container has its own private net stack:

- Interface
- Routing table
- Iptable
- Sockets

VETH devices can connect different namespaces (one in a container, one in the host machine).

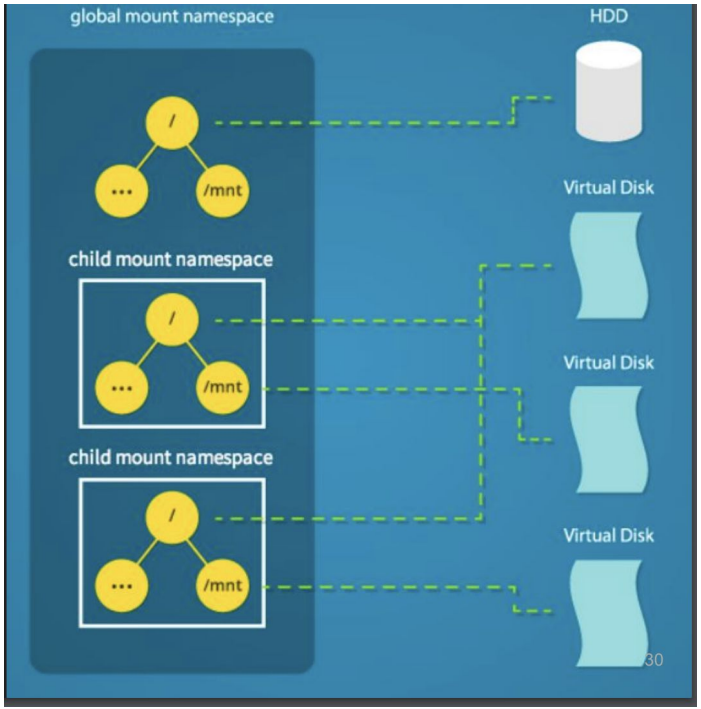
- VETH devices are virtual ethernet devices. They can act as tunnels between network namespaces to create a bridge to a physical network device in another namespace, but can also be used as standalone network devices.

Docker run uses a separate network namespace per container

CLONE_NEWPID | CLONE_NEWNET | CLONE_NEWNS | SIGCHLD (**check these flags: https://man7.org/linux/man-pages/man2/clone.2.html**)

Filesystem Namespaces

- Allows isolation of all mount points, not just root
- Similar to chroot, however provides proper isolation
- Atrributes can be changed between namespaces (read only, for instance)
- Used properly, avoids exposing anything about underlying system



Filesystem Namespaces

The main idea - to create virtual storage device using file as image, which will contain file system information and all contents.

Order of usage:
1. Create a zeroed image file (hint: man dd)
2. Setup loop mechanism to use this file as image for virtual storage device
(hint: man losetup)
3. Make filesystem on virtual storage device (hint: man mkfs)
4. Mount virtual storage device inside guest namespace for specific mount point (e.g. /home)
5. Create file inside mounted path in guest environment and check in host environment that this file is not exist

Hints: man dd, man mkfs, man mount

Loop

(losetup)

mount -t ext4 /dev/loop

Create a container

Creating a process, that is isolated, has its own namespaces, has its own root directory in file system, networking, can save its filesystem as file, and can even limit its own resources.

Lab Assignment

1. Create your own container. Isolate PID, mnt, net
2. Put container filesystem in a file by using loop and try to create file in your new mount point.
3. Benchmark your container vs your host (host, LXC) - use sysbench
   a. Attributes: CPU total time, FileIO read, FileIO write, Memory access, Thread execution
4. Write report where you explain the container you created, the steps you have performed and why and you provide insights into the benchmarking.

Expected Deliverables

1. zip archive with your *.c *.h *.sh files which are relevant to your container and make it work. Adding "readme" file is necessary.

2. pdf file with your report. Use the following template to create the report: https://docs.google.com/document/d/1p1E_wBcWOO4wcPrry_6K4hZvr2hwl85l/edit?usp=sharing&ouid=104103188587314732632&rtpof=true&sd=true