Lecture 6 : Containers

Dr. Bibhas Ghoshal

Assistant Professor

Department of Information Technology

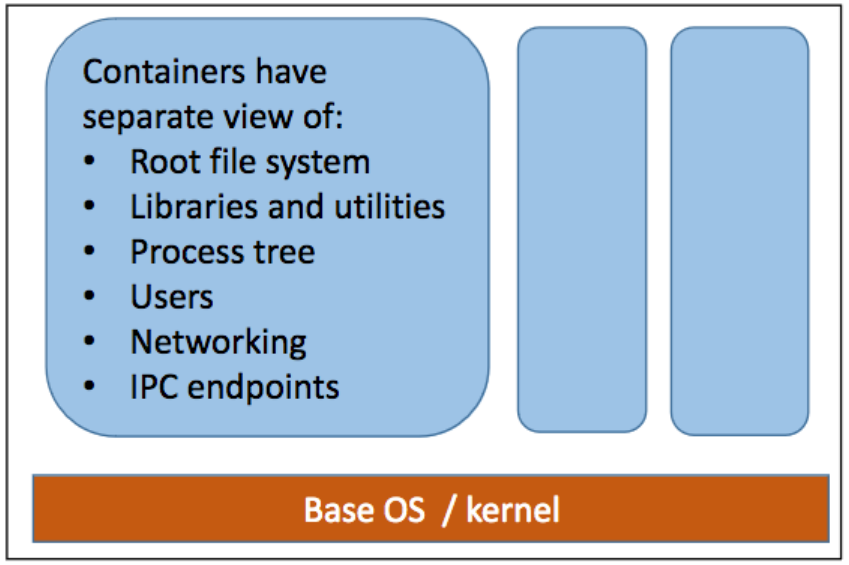
Indian Institute of Information Technology Allahabad

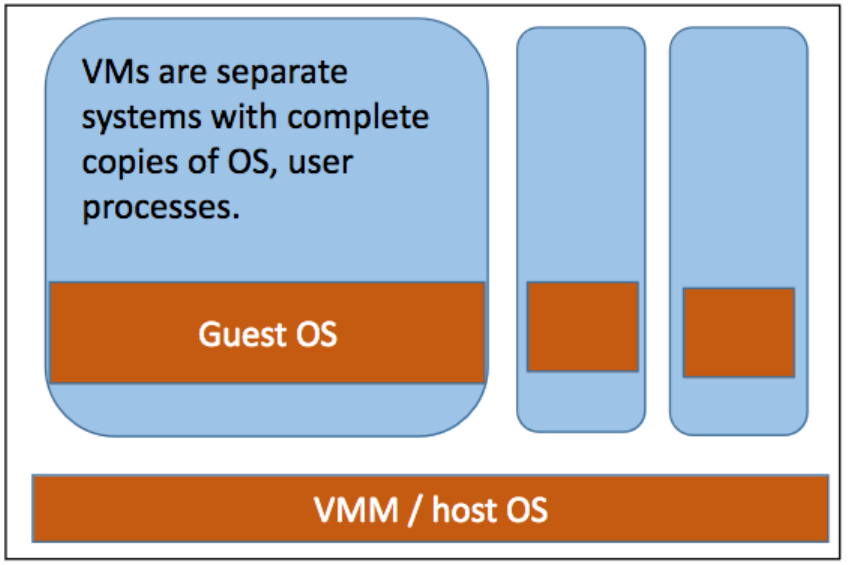
Slides Used in this Lecture have been adapted from slides of Professor Mythilli Vutukuru, Dept. Of CSE, IIT Bombay delivered for the course Virtualization and Cloud Computing

Containers: lightweight virtualization

Containers share base OS, have different set of libraries, utilities, root filesystem, view of process tree, networking, and so on.

- VMs have different copies of OS itself
- Containers have lesser overhead than VMs, but also lesser isolation





Slide Author : Mythilli Vutukuru

Namespace and Cgroups

Two mechanisms in Linux kernel over which containers are built:

• Namespaces: a way to provide isolated view of a certain global resource (e.g., root filesystem) to a set of processes. Processes within a namespace see only their slice of the global resource

• Cgroups: a way to set resource limits on a group of processes

• Together, namespaces and cgroups allow us to isolate a set of processes into a bubble and set resource limits

• Container implementations like LXC, Docker leverage these mechanisms to build the container abstractions

• LXC is general container while Docker optimized for single application

• Frameworks like Docker Swarm or Kubernetes help manage multiple containers across hosts, along with autoscaling, lifecycle management, and so on

Slide : https://lwn.net/Articles/531114/

Namespaces

Group of processes that have an isolated/sliced view of a global resource

- Default namespace for all processes in Linux, system calls to create new namespaces and place processes in them
- Which resources can be sliced?

  1. Mount namespace: isolates the filesystem mount points seen by a group of processes. The mount() and umount() system calls only affect the processes in that namespace.

  2. PID namespace: isolates the PID numberspace seen by processes. E.g., first process in anew PID namespace gets a PID of 1.

  3. Network namespace: isolates network resources like IP addresses, routing tables, port numbers and so on. E.g., processes in different network namespaces can reuse the same port numbers.

  4. UTS namespace: isolates the hostname seen by processes.

  5. User namespace: isolates the UID/GID numberspace. E.g., a process can get UID=0 (i.e., act as root) in one namespace, while being unprivileged in another namespace. Mappings to be specified between UIDs in parent namespace and UIDs in new namespace.

  6. IPC namespace: isolates IPC endpoints like POSIX message queues.

- More powerful than chroot() which only isolates root filesystem

Slide Author : Mythilli Vutukuru

Namespace API

- Three system calls related to namespaces:
  - clone() is used to create a new process and place it into a new namespace. More general version of fork().

```
childPID = clone(childFunc, childStack, flags, arg)
```

- Flags specify what should be shared with parent, and what should be created new for child (including virtual memory, file descriptors, namespaces etc.)
  - setns() lets a process join an existing namespace. Arguments specify which namespace, and which type.
  - unshare() creates a new namespace and places calling process into it. Flags indicate which namespace to create. Forking a process and calling unshare() is equivalent to clone().
- Once a process is in a namespace, it can open a shell and do other useful things in that namespace
- Forked children of a process belong to parent namespace by default

Slide Author : Mythilli Vutukuru

Namespace handles : Referring Namespaces

- /proc/PID/ns of a process has information on which namespace a process belongs to. Symbolic links pointing to the inode of that namespace (“handle”)

```
$ ls -l /proc/$$/ns          # $$ is replaced by shell's PID
total 0
lrwxrwxrwx. 1 mtk mtk 0 Jan  8 04:12 ipc -> ipc:[4026531839]
lrwxrwxrwx. 1 mtk mtk 0 Jan  8 04:12 mnt -> mnt:[4026531840]
lrwxrwxrwx. 1 mtk mtk 0 Jan  8 04:12 net -> net:[4026531956]
lrwxrwxrwx. 1 mtk mtk 0 Jan  8 04:12 pid -> pid:[4026531836]
lrwxrwxrwx. 1 mtk mtk 0 Jan  8 04:12 user -> user:[4026531837]
lrwxrwxrwx. 1 mtk mtk 0 Jan  8 04:12 uts -> uts:[4026531838]
```

- Namespace handle can be used in system calls (e.g., argument to setns)
- Processes in same namespace will have same handle, new handle created when new namespace created

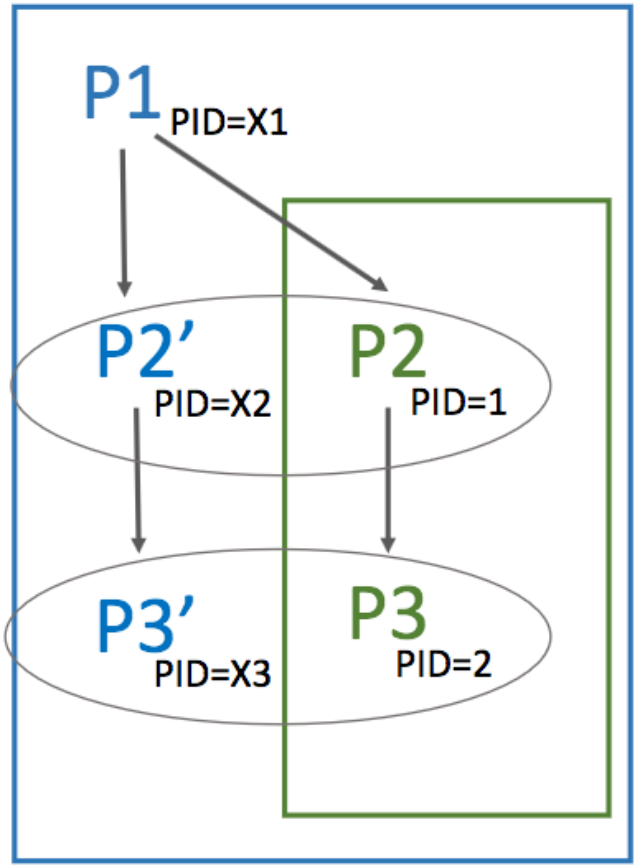
Slide : https://lwn.net/Articles/531114/

PID Namespaces

The first process to be created in a new PID namespace will have PID=1 and will act as init process in that namespace

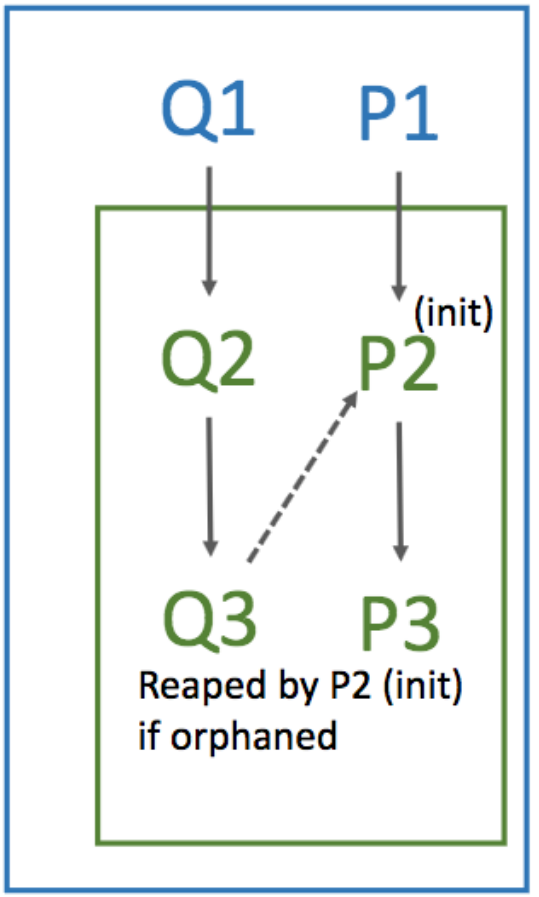
- Will reap orphans in this namespace

- Processes in PID namespace get separate PID Number space (child of init gets PID 2 onwards)
- A process can see all other processes in its own or nested namespaces, but not in its parent namespace P2, P3 not aware of P1 (parent PID of P2 = 0)
- P1 can see P2 and P3 in its namespace (with different PIDs)
- P2=P2' (just different PIDs in different namespaces



PID Namespaces

- First process in a namespace acts as init and has special privileges
  - Other processes in namespace cannot kill it
  - If init dies, namespace terminated
  - However, parent process can kill it in parent namespac
- Who reaps whom?
  - Init process reaped by parent in parent namespace
  - Other child processes reaped by parent in same namespace
  - Any orphan process in namespace reaped by init of that namespace



PID Namespace

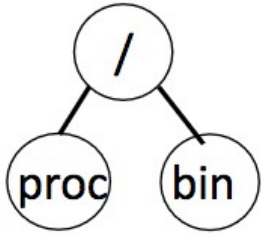
- Namespace related system calls have slightly different behavior with PID namespace alone
- clone() creates a new namespace for child as expected
- However, setns() and unshare() do not change PID namespace of calling process. Instead, the child processes will begin in a new PID namespace
- Why this difference? If namespace changes, PID returned by getpid() will also change. But many programs make assumption that getpid() returns same value throughout life of process.
    - getpid() returns the PID in the namespace the process resides in

Mount Namespace

- Root filesystem seen by a process is constructed from a set of mount points (mount() and umount() syscalls)
- New mount namespace can have new set of mount points
  - New view of root filesystem
- •Mount point can be shared or private
  - Shared mount points propagated to all namespaces, private is not
  - If parent makes all its mount points private and clones child in new mount namespace, child starts with empty root filesystem
- Container frameworks use mount namespaces to create a custom root filesystem for each container using a base rootfs image

Mount Namespace and ps

- How does “ps” work?
  - Linux has a special procfs, in which kernel populates info on processes
  - Reading /proc/PID/.. does not read file from disk, but fetches info from OS
  - procfs mounted on root as a special type of filesystem
- P1 clones P2 to be in new PID namespace but uses old mount namespace. We open shell in new PID namespace and run ps. We will still see all processes of parent namespace. Why?
  - ps command is still using procfs of parent’s mount namespace
- How to make ps work correctly within a PID namespace?
  - Place P2 in new mount namespace, mount a new procfs at root
  - New procfs at new mount point is different from parent’s procfs
  - ps will not show only processes in this PID+mount namespace



Network Namespace

- Network namespace can be created by cloning a process into a new namespace, or simply via command line

```
# ip netns add netns1
```

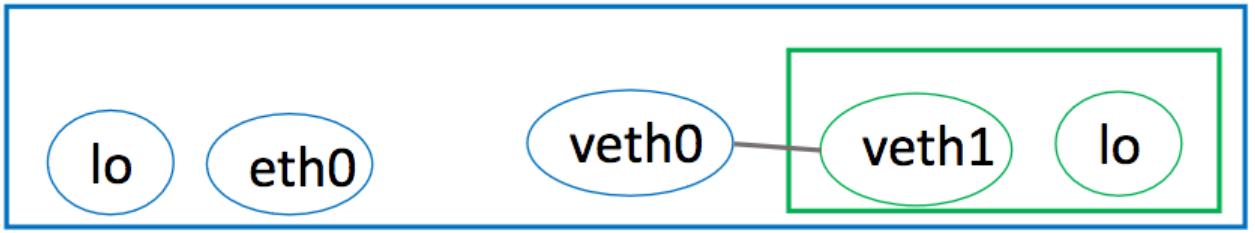
- List of network namespaces can be viewed at /var/run/netns, can use setns() to join existing namespace
- Command “ip netns exec” can be used to execute commands within network namespace, for example, to view all IP links:

```
# ip netns exec netns1 ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Network Namespace

- Any new network namespace only has loopback interface. How to communicate with rest of network?

- Create a virtual Ethernet link (veth pair) to connect parent namespace to new child namespace
  - Assign endpoints to two different namespaces
  - Assign IP addresses to both endpoints
  - Can communicate over this link to parent namespace
  - Can configure bridging/NAT to connect to wider internet



Slide : https://lwn.net/Articles/531114/