Container Primitives

Namespaces & CGroups

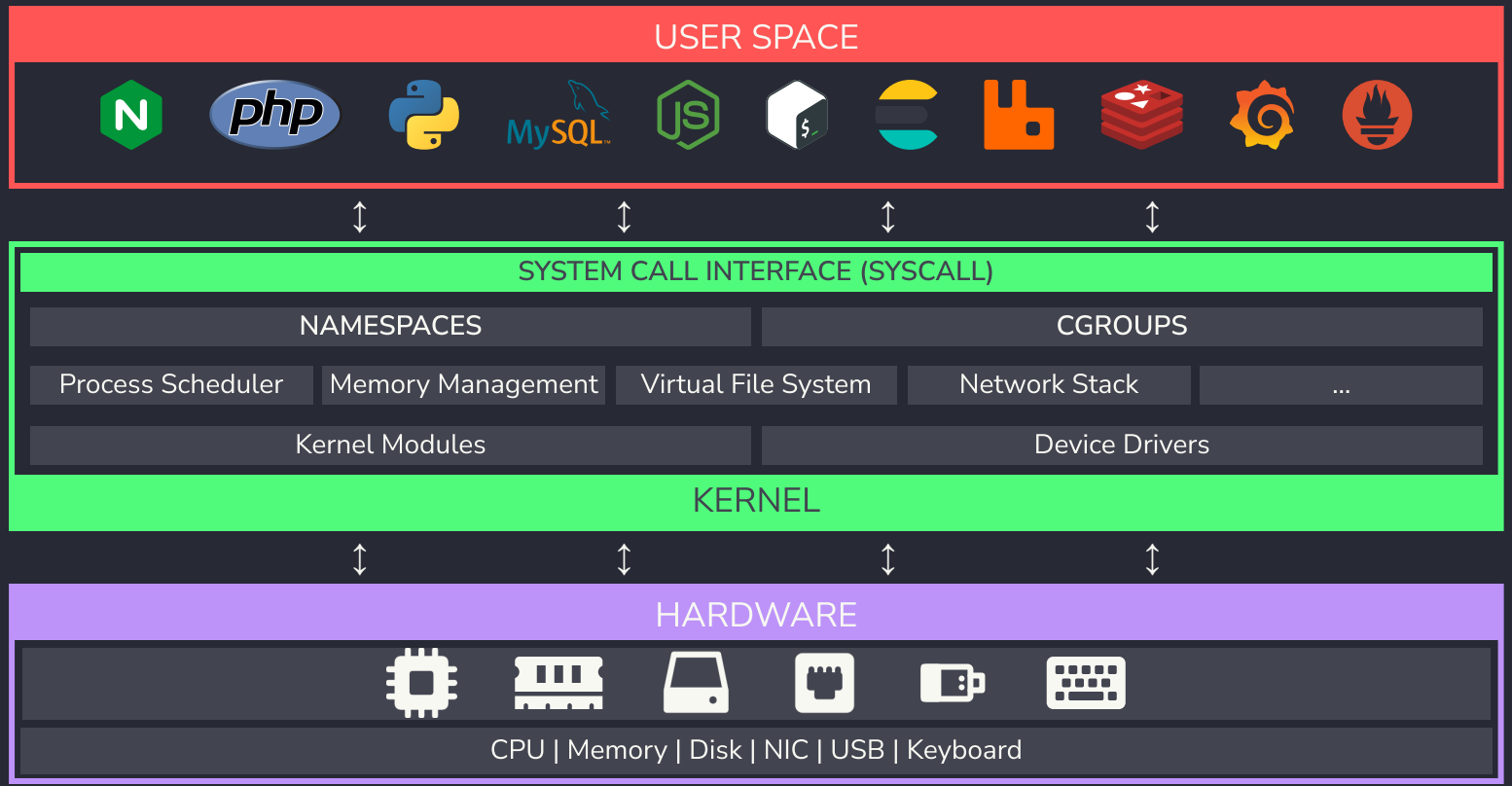
$ whoami

```
Name = Abhishek Singh Okheda
Organization = Worldlink Communications Ltd.
Designation = Senior System Architect
Description = A tech enthusiast
```

About Presentation

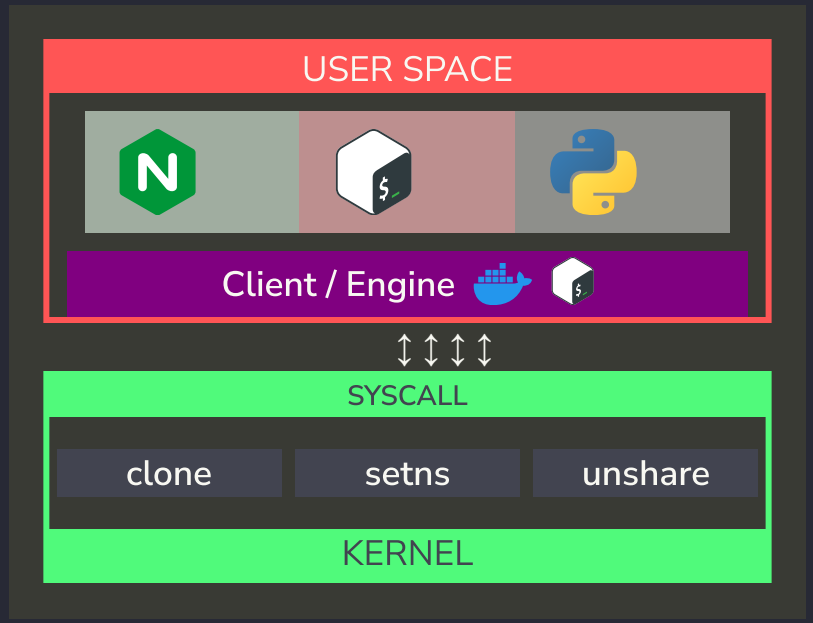
- Understand what containers are from the perspective of Operating System
- Take a look into Linux kernel features that make containerization a possibility
  - Namespaces
  - CGroups

Processes >> KERNEL >> Hardware



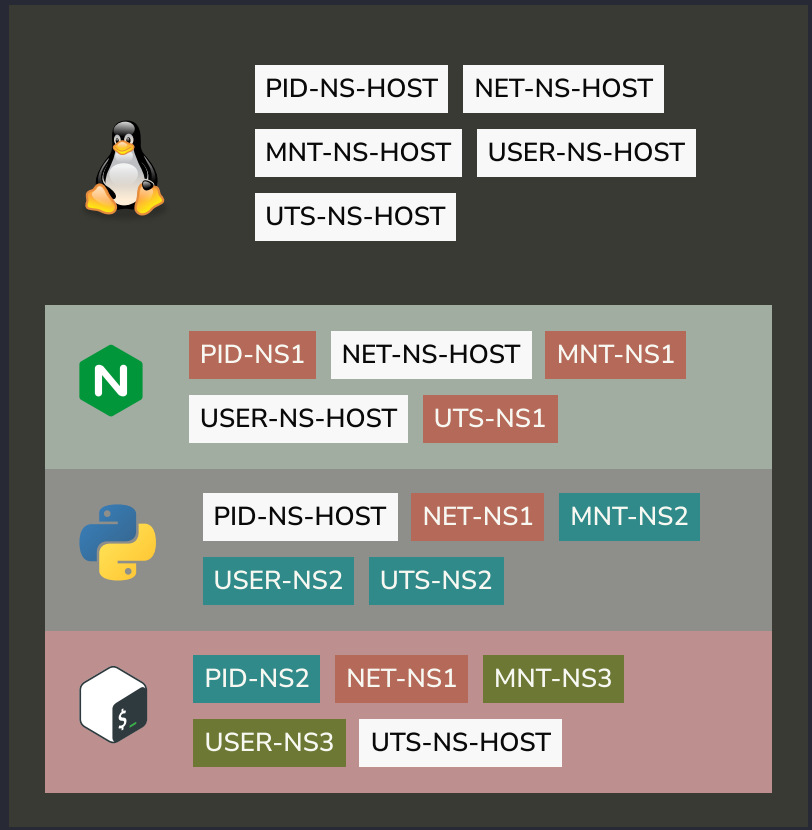
Namespaces

- Kernel feature that provides way to keep processe(s) separated in isolated groups, required for containers
- Allows one set of processes to virtually see different view of system resources compared to other set of processes.



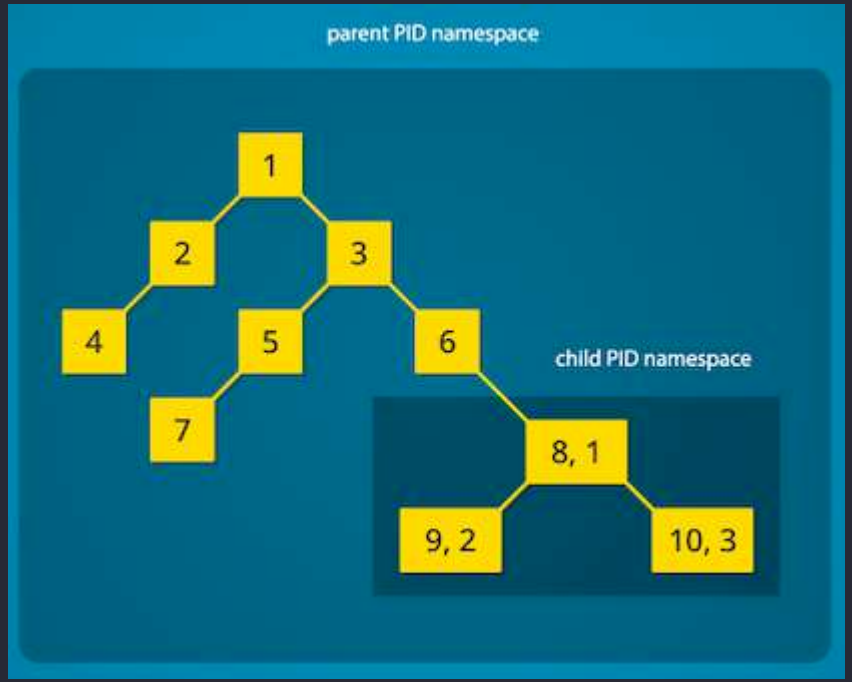
```
root@npnog10-vm:~# lsns -t pid -t net  -t uts -t net -t mnt
        NS TYPE NPROCS   PID USER                NETNSID NS
4026531836 pid     165     1 root
4026531838 uts     162     1 root
4026531840 mnt     157     1 root
4026531860 mnt       1    59 root
4026532312 net     165     1 root             unassigned
4026532640 mnt       1   488 root
4026532641 uts       1   488 root
4026532652 mnt       1   649 systemd-timesync
4026532653 uts       1   649 systemd-timesync
4026532654 mnt       1   689 systemd-network
4026532655 mnt       1   691 systemd-resolve
4026532711 mnt       1   723 root
4026532712 mnt       1   711 root
4026532713 mnt       1   759 root
4026532714 uts       1   723 root
root@npnog10-vm:~# ls -l /proc/1/ns/ | cut -d' ' -f9,10,11
cgroup → cgroup:[4026531835]
ipc → ipc:[4026531839]
mnt → mnt:[4026531840]
net → net:[4026532312]
pid → pid:[4026531836]
pid_for_children → pid:[4026531836]
time → time:[4026531834]
time_for_children → time:[4026531834]
user → user:[4026531837]
```

Process(es) and Namespaces



PID namespace

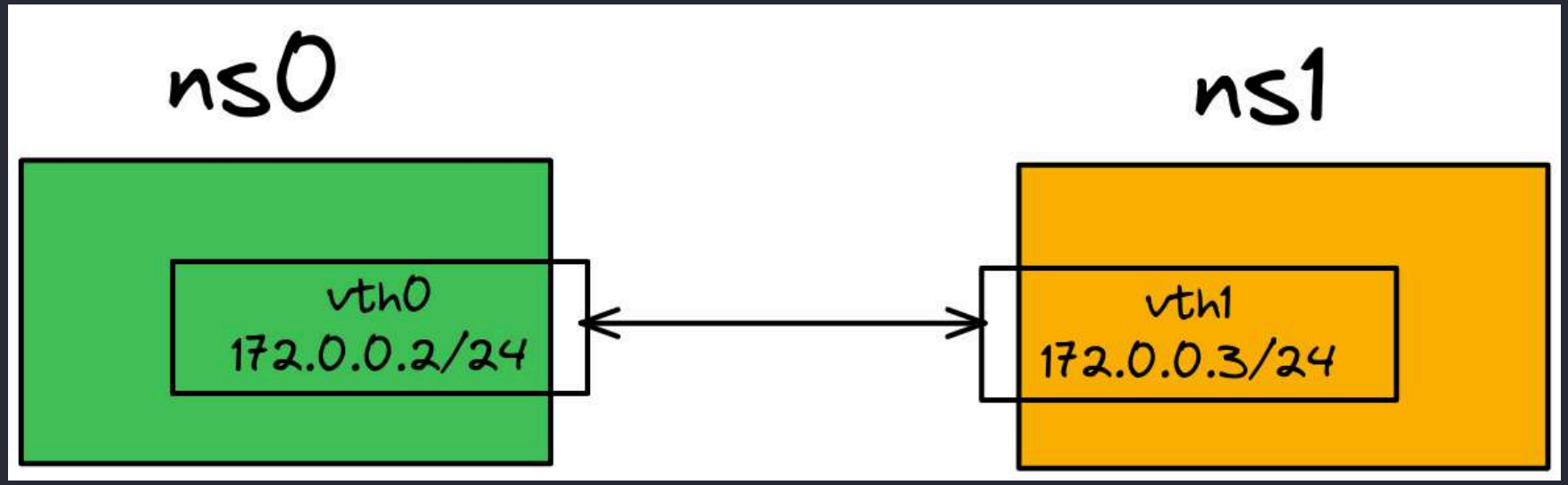
Isolated view of Process IDs and own separate view of process tree



NET namespace

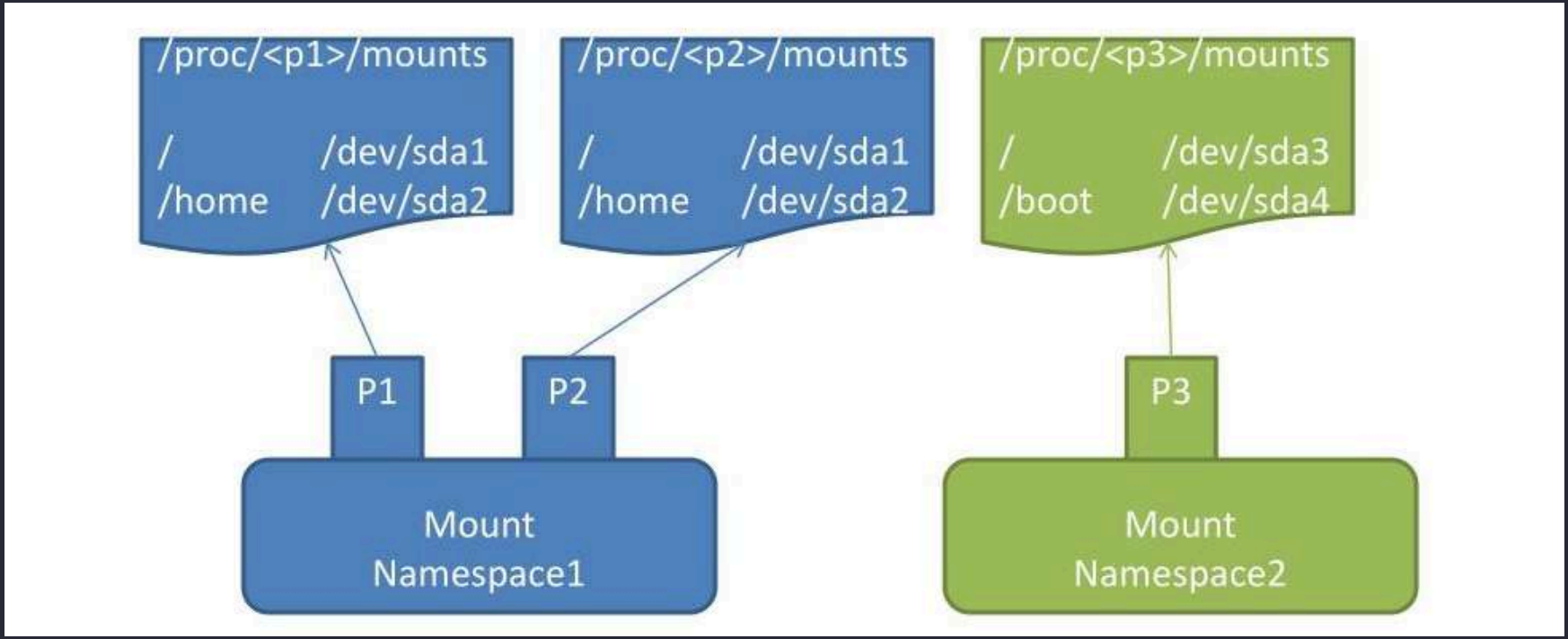
Isolated view of Network Inferfaces

Network Inferfaces in different namespace can be interconnected by using Virtual Ethernet (veth) pairs



MNT namespace

Isolated view of Mount Points and its separate view of filesystem layout



User namespace

Mapped view of User IDs and Group IDs as different IDs in the namespace

EXAMPLE
root (UID=0) user inside a namespace can be mapped to some unprivileged user (UID=10001) of host

UTS namespace

Different hostname and domain can be assigned per namespace without affecting rest of the system.

Play: lsns, unshare and nsenter

CLI tools that uses namespace syscalls to work with namespaces

```
root@npnog10-vm:~# lsns --type pid
        NS TYPE NPROCS   PID USER COMMAND
4026531836 pid     168     1 root /sbin/init
```

```
root@npnog10-vm:~# unshare --pid --net --uts --mount-proc -fork bash
root@npnog10-vm:~# pstree -ap
bash,1
  └─pstree,11 -ap
root@npnog10-vm:~# hostname container1
```

```
root@npnog10-vm:~# lsns --type pid
        NS TYPE NPROCS   PID USER COMMAND
4026531836 pid     168     1 root /sbin/init
4026532668 pid       1 55045 root bash
```

```
root@npnog10-vm:~# hostname
npnog10-vm
root@npnog10-vm:~# nsenter --target 55097 --uts hostname
container1
```

Docker and namespaces

Let’s find out the host process ID for PID 1 in container

```
root@npnog10-vm# docker inspect mynginx --format '{{ .State.Pid }}'
1171
```

Let’s find out the namespaces that process belongs to

```
root@npnog10-vm# lsns -p 1171
        NS TYPE   NPROCS   PID USER COMMAND
4026531834 time       38     1 root /sbin/init
4026534257 mnt         9  1171 root nginx: master process nginx -g daemon off;
4026534615 user       38     1 root /sbin/init
4026536531 uts         9  1171 root nginx: master process nginx -g daemon off;
4026536532 ipc         9  1171 root nginx: master process nginx -g daemon off;
4026536533 pid         9  1171 root nginx: master process nginx -g daemon off;
4026536534 net         9  1171 root nginx: master process nginx -g daemon off;
4026536589 cgroup      9  1171 root nginx: master process nginx -g daemon off;
root@npnog10-vm#
```

CGroups (Control Groups)

Kernel feature that allows

- Accounting
- Limitation

on Resource usage for collection of process(es)

Resources

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

CGroup VFS

User space utilities and engines communicate to kernel over virtual file system typically mounted on /sys/fs/cgroup

```
root@npnog10-vm:~# mount -t cgroup2
none on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime)
root@npnog10-vm:~#
```

Docker and CGroups

Let’s run a container in docker with memory limit

```
root@npnog10-vm# docker run -d --name limit-container --memory 20M nginx:alpine
47955c3723adbe7034fbcbb6dabc056767677be9076faac1f5d5ca8fa40d4648
```

Let’s find out the cgroup vfs path for the container process

```
root@npnog10-vm# docker inspect limit-container --format '{{ .State.Pid }}'
1609962
root@npnog10-vm# cat /proc/1609962/cgroup
0::/system.slice/docker-47955c3723adbe7034fbcbb6dabc056767677be9076faac1f5d5ca8fa40d4648.scope
```

Let’s see how the memory limit can be seen in the cgroup VFS

```
root@npnog10-vm# cat \
/sys/fs/cgroup/system.slice/docker-47955c3723adbe7034fbcbb6dabc056767677be9076faac1f5d5ca8fa40d4648.scope/memory.max
20971520
root@npnog10-vm#
```

THANK YOU