differing topics → heading / subheading

everything else is "imp"

→ What is a container?

- A light weight VM? But actually no.
- It's more like chroot on steroids as:
  - (*) uses the host kernel. (it's just processes running on kernel)
  - (*) A "ps" would show all the process running inside the container? Yes!

- The kernel does not have containers, "searching" for containers in kernel show nothing. Instead they have namespaces & Cgroup!

→ CGroup.

- Cgroup allow for metering & limiting of resources used by the ~~resources~~ container.
- Each subsystem has its own hierarchical tree
- PS tree is an exact example of this.
- We can keep track of all resources used by each groups.

- A disk page is one that is got from the disk, swapping them is easy as no extra step
- But pages (heaps) generated during run time needs to swapped in & out of mem.
- To solve this cgroup keeps track of active & inactive pages.
- Each group can have 2 sorts of limit:
  - → hard limits
  - → soft limits

  → if one container goes out of memory, it kills process ONLY in that container.

- Soft limits are triggered when the ~~total~~ resources of the system are used up. In such a case, group that is breaching its soft limit the most gets affected.
- OOM - Out of Memory killer.

- Every time a page is given / taken a counter needs to be updated.
- Cgroups exist for Everything or None.

CPU Cgroups

- keeps track of user CPU time.
- Cannot set limits but they can set weights.

Why? because modern hardware can change their performance when needed.
# of ins is also wrong.

- cpuset is a way to pin processes to specific CPUs
  → helps avoid cpu jumping around CPUs & helps in NUMA architecture.
  → large context switches in memory cause this jumping around as well.

Blkio Cgroup

- keeps track of I/O for each group limits on I/O for each grp aswell.
- often writes are flushed to write cache & later

synchronously to the main mem, Blkio limits do not apply on writes to cache.

⇒ The actual write will not actually seem slow.

Net_cls & net_prio Cgroups

& prio & priorities

- Net cls will assign classes to packets
- works only on egress traffic. And can't really do control on speed of traffic.
- All it can do is apply a tag on the kind of traffic & use TCP/IP controls on them

Device Cgroups

- Gives control on various devices, lucky us that devices are basically files in linux.
- Some basic devices like tty, null, random. Access is given to all of them.
- /dev/random uses user inputs for random #'s but in a container env, this is not good enough

Some generics about Cgroups

- PID 1 is at top of every tree
- New process start under same group as parents

- Groups are stored in "Pseudo-fs"
- Groups are created simply by mkdir
- `echo $PID > /sys/fs' to actually move the process.
- Systemd, runit, cgmanages are all Cgroup mangers.

→ Name Spaces

- Cgroups limit what you can use, where name Space limit what you can view.
- Yet again there are many name Space, each process is one of the NS of each type.

types of

PID NS

- Process of one NS can only see processes in the same PID NS.
- A given process will have 2 PID, a global PID & a Local PID (inside NS). Hence every process in a container must have a common parent NS.
- The PID inside NS is not same as global. But the process that creates the NS always

starts assigning from 1. If two process with PID 1 in a NS is killed the entire NS is killed.

Network NS

- processes within a single NS get their own interfaces, routing tables, iptables, sockets
- When you run docker containers with same network you are making them share a net NS.
- You can then later connect these interfaces to host. kinda like having a software switch.

Mnt namespaces

- Each container can have their own root, every process in a given mnt NS share a common root.
- This is similar to what chroot does, each process is given a unique mnt name space like /tmp.

IPC NS

- a separate IPC for a bunch of processes? Not quite
- You get to have a bunch of semaphores, msg queues & IPC shared memory.

User NS

- Allows one user to have diff UID's GID's depending on user name space in processed.
- like: User 'docker' will have UID 0 in docker's user NS. but "docker" will not have any perm in root name space.

How does one manipulate name spaces?

- One creates new NS along with the process creation. (ofc happens with extra flags)
- All NS are also files in `/proc/<pid>/ns`

→ Copy on write storage. & storage on docker

- Instead of copying the whole fs of container, we can track the changes alone. this is CoW
- The container doesn't know about the fs ownership.
- Docker uses something called Union fs. to achieve this.
- MNT NS & Chroot can be used for this as well.

- Unionfs is a system that creates an illusion that multiple dirs from host seems like one in Name space without any modification.

- `mount -t unionfs -O dirs=/...:/... none /mnt/xyz`

  xyz will have all the files & dirs in . -O is a copy on write flag.

- each layer that constitutes an Unionfs is called a branch & can have permission.

- In docker:

  first layer: Base image → read only layer.
  2nd " = Container → r/w layer.

- Unionfs use the COW pattern.
  → All files that can be modified a copy is created.

- Unionfs is not good for long lived data.