ADVANCED CLOUD COMPUTING

CONTAINERS

DEEP DIVE

DEFINITIONS

CONTAINER IMAGE

A a file which is pulled down from a **Registry Server** and used locally as a **mount point** when **starting** Containers.

ATTENTION:often people say container images talking about reposiotries (bundle of multiple container Image Layers as well as metadata)

Formats

- Docker
- Appc
- LXD
- Open Container Initiative (OCI)

  most common

layman terms: container at rest

DEFINITIONS

CONTAINER ENGINE

A container engine is a piece of software that:

- accepts user requests (eg Command line options)
- pulls images,
- ~ runs the container
  (the container runtime runs the container)

Container engines:

- docker
- podman
- RKT
- CRI-O
- LXD.

cloud providers, often have their own container engines.

Is the software you will mostly use

DEFINITIONS

CONTAINER

A container is the runtime instantiation of a Container Image (Registry in exact terms).

A container is a **standard Linux process** typically created through a clone() system call instead of fork() or execvp(). Also, containers are often isolated further through the use of cgroups, SELinux or AppArmor.

DEFINITIONS

CONTAINER RUNTIME

lower level component used in a Container Engine

The OCI Runtime Standard reference implementation is runc.

- crun
- railcar
- katacontainers

runtime operations:

- use container mount point
- use container metadata
- start containerized processes (clone system call)
- Setting up cgroups
- Setting up SELinux Policy/App Armor rules

DEFINITIONS

SYSTEMS CALL

- clone()

  This system call provide precise control over what pieces of execution context are shared between the calling process and the child process. eg. the caller can control whether or not
  - the two processes share the virtual address space,
  - the table of file descriptors, and the table of signal handlers.
  - child process to be placed in separate namespaces.
- fork()

  creates a new process by duplicating the calling process. The new process is referred to as the child process.
- execvp()

  replaces the current process image with a new process image.

DEFINITIONS

REPOSITORY

repositories can be approximated with container images, but it’s important to realize that these repositories are actually made up of layers and include metadata (manifest.json)

DEFINITIONS

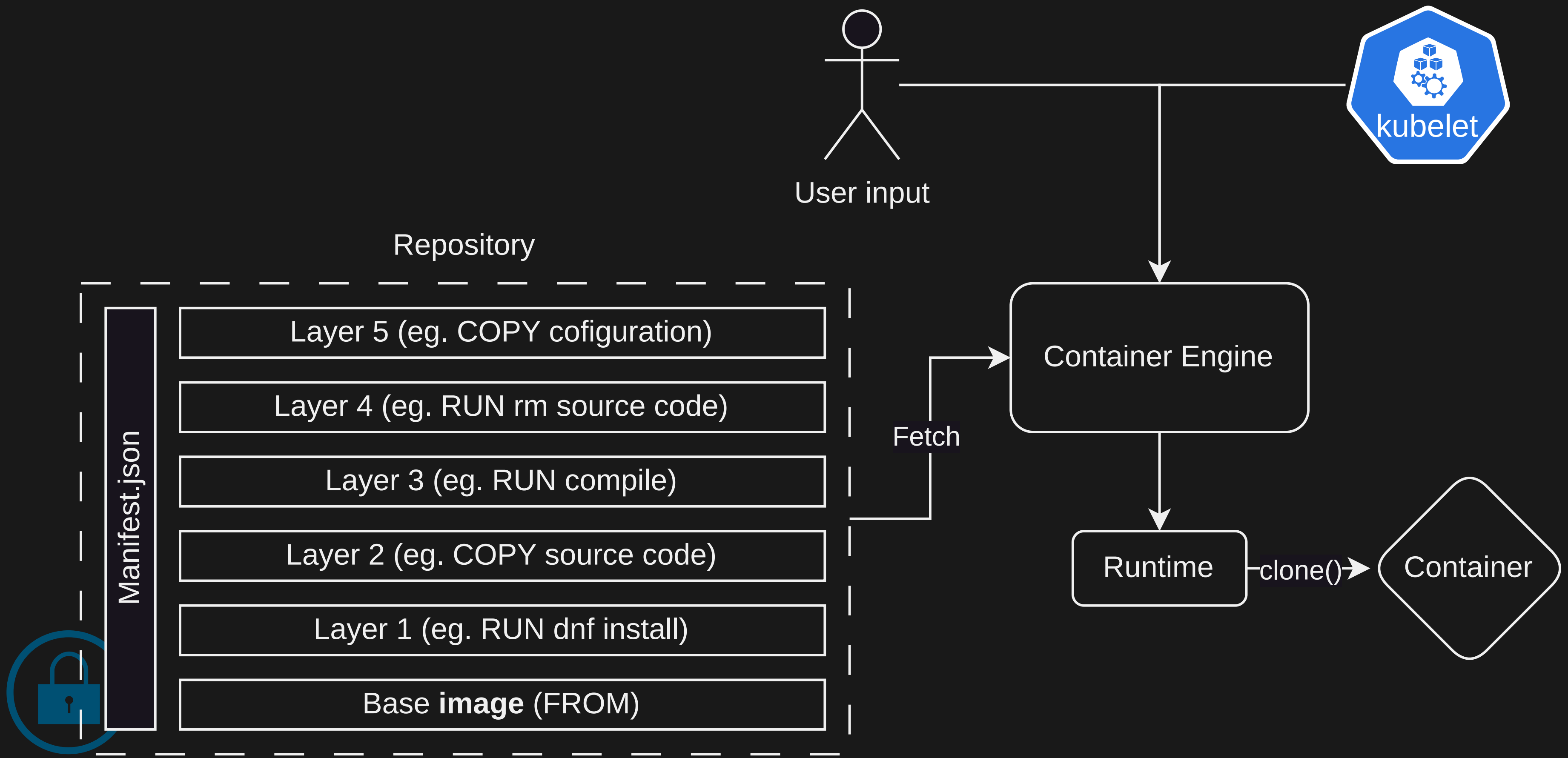
REPOSITORY

repositories can be approximated with container images, but it’s important to realize that these repositories are actually made up of layers and include metadata (manifest.json)

what does it means: `docker pull rhel7`?

- `docker pull registry.access.redhat.com/rhel7:latest`
- REGISTRY/..optinal groups../REPOSITORY[:TAG]

DEFINITIONS

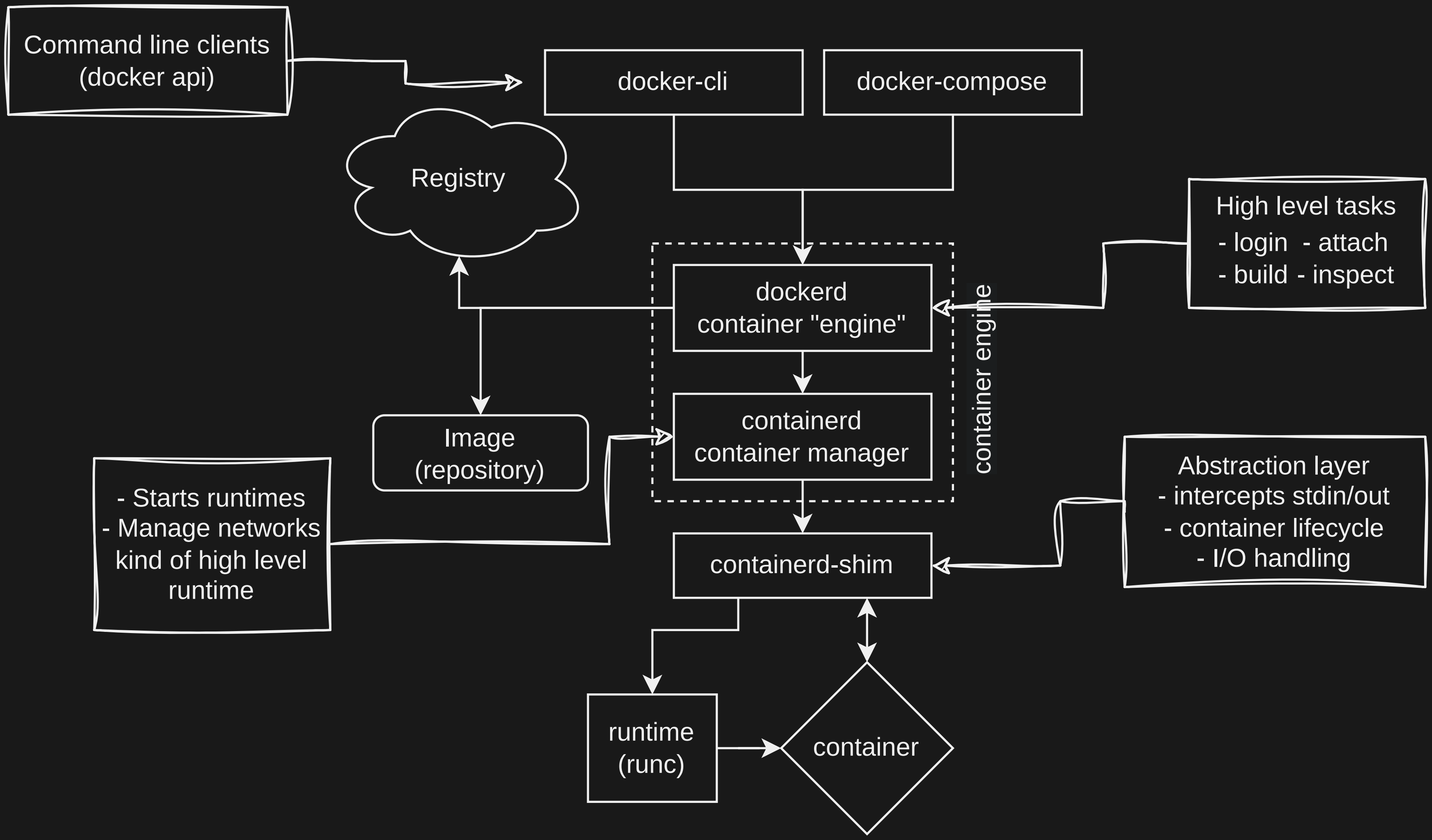


User input
kubelet
Repository
Manifest.json
Layer 5 (eg. COPY cofiguration)
Layer 4 (eg. RUN rm source code)
Layer 3 (eg. RUN compile)
Layer 2 (eg. COPY source code)
Layer 1 (eg. RUN dnf install)
Base **image** (FROM)
Fetch
Container Engine
Runtime
clone()
Container

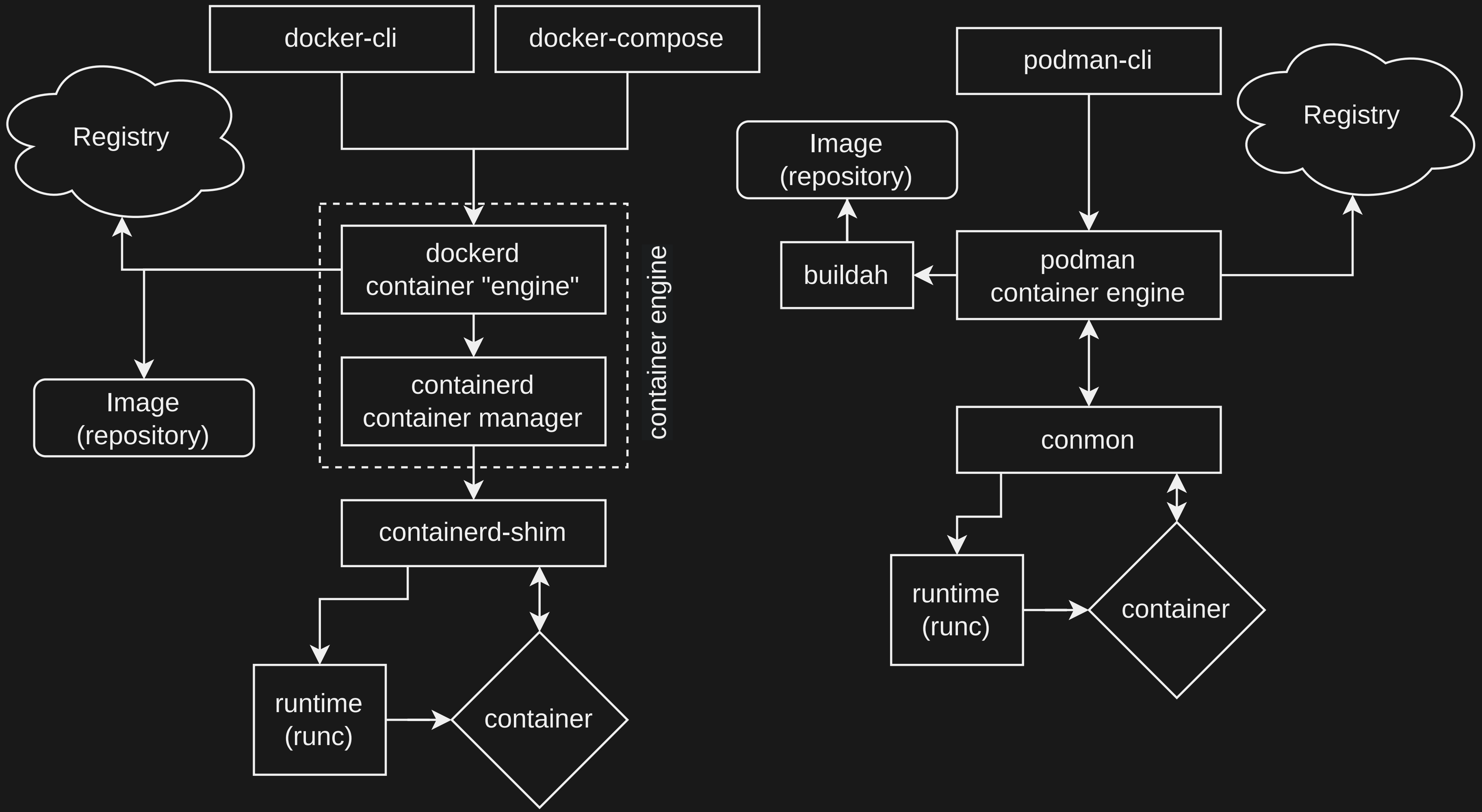
ALL TOGHETHER

IMPLEMENTATION

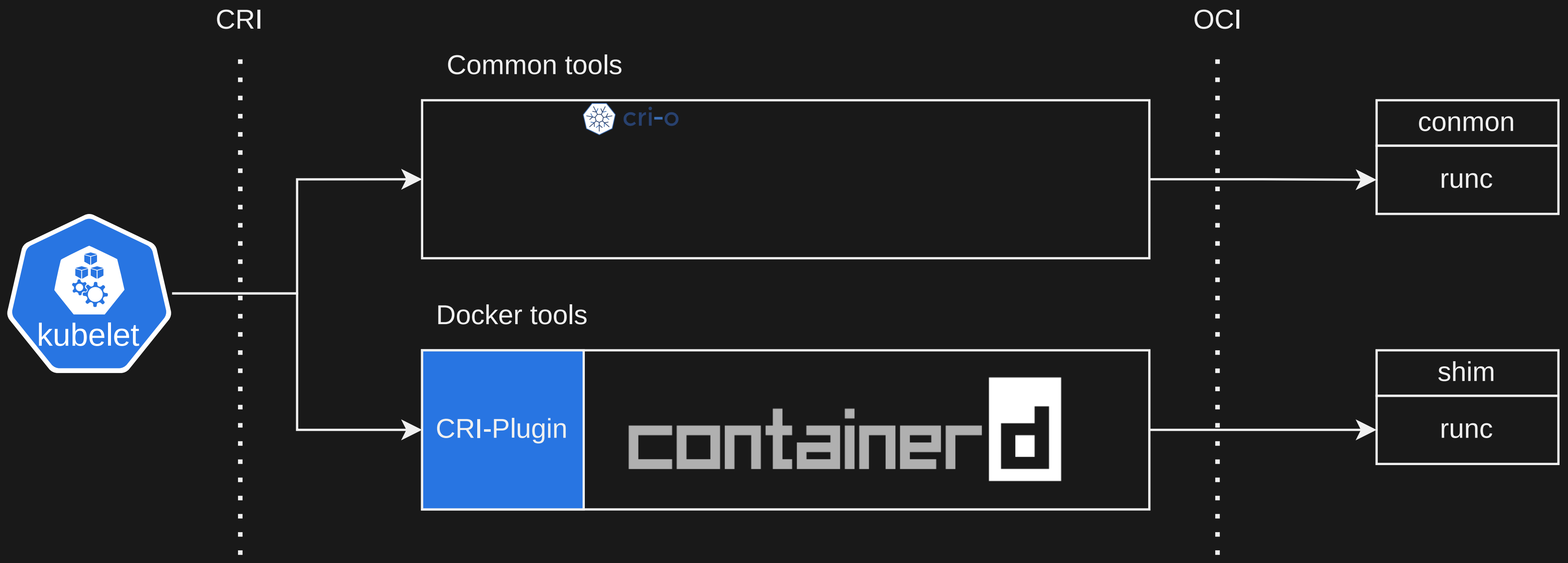
DOCKER STACK



DOCKER VS PODMAN



KUBERNETES


CRI
OCI
Common tools
cri-o
conmon
runc
kubelet
Docker tools
CRI-Plugin
containerd
shim
runc

LINUX NAMESPACES

- Feature of the linux kernel
- partition kernel resources

a set of processes that shares a namespace set has access to the same resources

COMMON NAMESPACES

- Process ID (PID)
- net
- uts
- user
- mnt
- IPC
- cgroups

Isolate the PID number space.

Processes in different PID ns can have the same PID.

From an host point of view the process has two PIDs, one inside and one outside the process ns.

- PID 1 inside each ns
- PID ns can be migrated between hosts
- a processe can only view processes insides the ns and sub ns.

COMMON NAMESPACES

- PID
- network
- uts
- user
- mnt
- IPC
- cgroups

Each net ns has its own resources

- network devices
- IP addresses
- IP routing tables
- nftables
- `/proc/net`

COMMON NAMESPACES

- PID
- net
- Unix Timesharing System (uts)
- user
- mnt
- IPC
- cgroups

old name, now is only used for hostname segregation

COMMON NAMESPACES

- PID
- net
- uts
- user
- mnt
- IPC
- cgroups

Most important. Isolate the user and group ID number spaces. A process user and group can be differnt inside and outside the ns.

Interest case: having a user ID 0 inside the namespace but beeing unpriviledge outside

COMMON NAMESPACES

- PID
- net
- uts
- user
- mount
- IPC
- cgroups

Isolate mountpoints to hide files, and show different filesystem hierarchy. Can be interpreted a bit as `chroot`.

COMMON NAMESPACES

- PID
- net
- uts
- user
- mnt
- interprocess communication (IPC)
- cgroups

IPCs handle the communication between process using shared memory areas and POSIX message queues and semaphores.

COMMON NAMESPACES

- PID
- net
- uts
- user
- mnt
- IPC
- cgroups

repoints the `/sys/fs/cgroup` folder

- allow for ns migration between hosts
- Avoid leaking sensitive information about the host's resource.

CAPABILITIES

categories of processes:

- privileged processes (effective user ID is 0), bypass all kernel permission checks
- unprivileged processes (effective UID is nonzero), subject to full permission checking

privileges associated with superuser are now devided into distinct units, known as capabilities.

Capabilities are a per-thread attribute.

WHAT IF YOU WANT TO EXECUTE PRIVILEGED OPERATIONS?

- legacy approach: `setuid` bit, this allow the user to run the program as the program owner.

```
➜  ~ ls -la /usr/bin/passwd
-rwsr-xr-x. 1 root root 91624 Oct 15 02:00 /usr/bin/passwd
```

- novel approach: leverage capabilities

CAPABILITY SETS

- Permitted

  limiting superset for the effective capabilities that the thread may assume.

- Effective

  capabilities used by the kernel to perform permission checks for the threa

  - inheritable
  - bounding
  - ambient

CAPABILITIES

38 capabilities are responsible for controlling syscall

- CAP_SYS_BOOT allow rebooting
- CAP_IPC_LOCK allow memory allocation with huge pages via mmap
- CAP_WAKE_ALARM Trigger something that will wake up the system
- CAP_NET_ADMIN perform network operations

WHY SHOULD I CARE?

capability set are tied to a **user namespace**

this means that each namespace will have its own set of capabilites, however a child namespace can never be granted more capabilities than the creating process. you will have to fix them sometimes on kubernetes

IN PRACTICE

```
➜  ~ getpcaps $$
1021810: cap_wake_alarm=i
➜  ~ getpcaps 1
1: =ep
➜  ~ unshare -UinpmrC -f /bin/bash
bash-5.2→ whoami
root
bash-5.2→ getpcaps $$
1: =ep
bash-5.2→ exit
```

NAMESPACES

USER NS

User ns isolate security-related identifiers and attributes

- user IDs/group IDs,
- keyrings
- capabilities.

A process’s user and group IDs can be different inside and outside a user ns.

FFECT OF CAPABILITIES

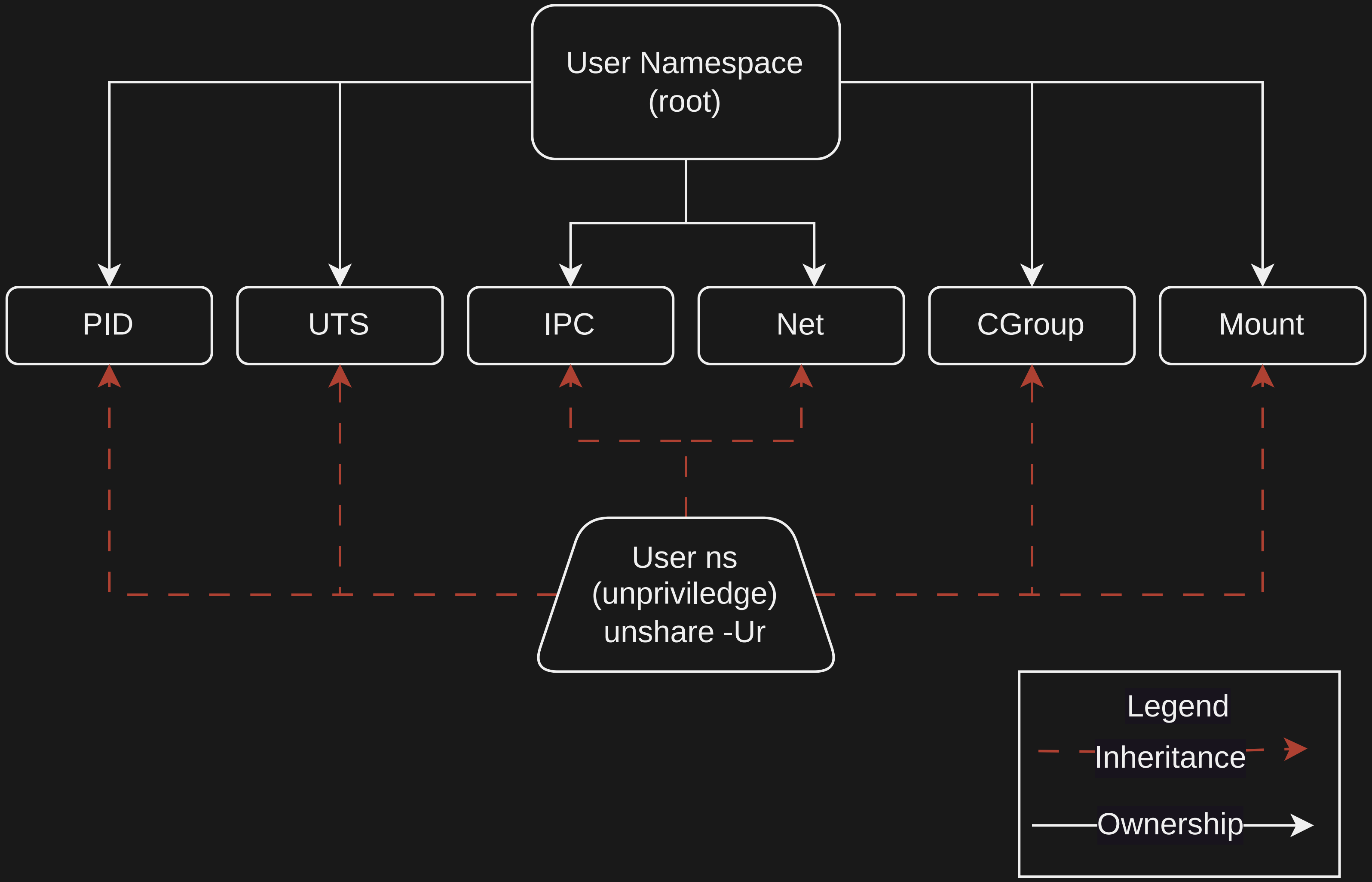
Having a capability inside a user ns permits a process to perform operations only on **resources governed by that ns**.

INTERACTION OF USER NS AND OTHER TYPES OF NS

- unprivileged processes can create user ns
- when nonuser ns is created, it is owned by the user ns in which the creating process is a member.

  Privileged operations on resources governed by the nonuser ns require that the process has capabilities in the user ns that owns the nonuser ns.

HANDSON



User Namespace (root)
PID
UTS
IPC
Net
CGroup
Mount
User ns (unpriviledge) unshare -Ur
Legend
Inheritance
Ownership

HANDSON

Stream online

MOUNT NS

Mount ns provide isolation of the list of mounts seen by the processes in each ns instance. Thus, the processes in each of the mount ns instances will see distinct single-directory hierarchies.

MOUNT NS

Unexpected behaviour: **a new mount ns is not empty**

actions taken on a poorly configured mount ns will impact the host.

MOUTPOINT PROPAGATION

Mountpoints propagates between mount ns because of the **shared subtree** feature.

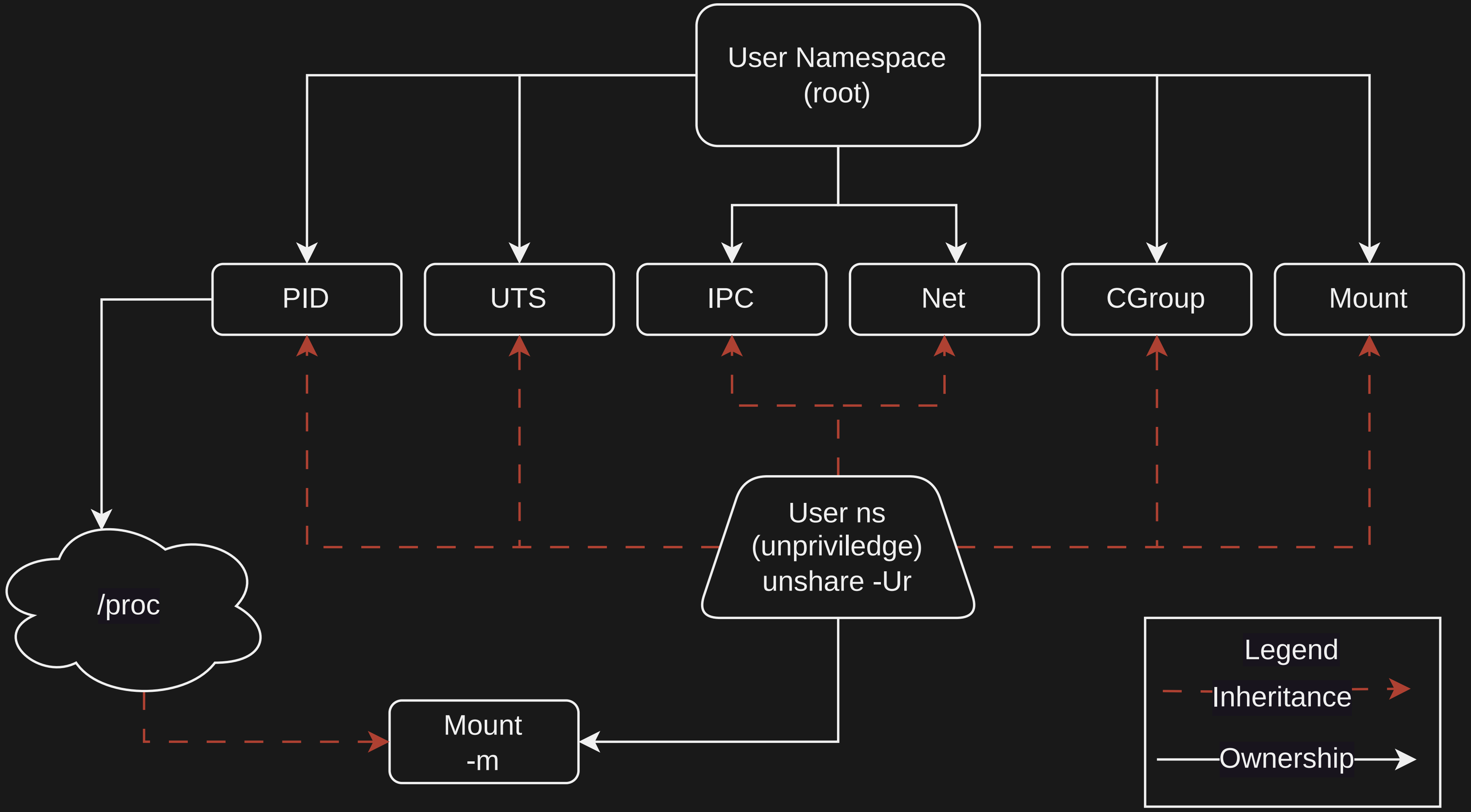
modify mount point propagation:

- shared: A mount that belongs to a peer group. Any changes will propagate to all members.
- slave: One-way propagation. master propagate events to a slave, but not viceversa.
- shared and slave: Indicates that the mount point has a master, but it also has its own peer group. The master will not be notified of changes to a mount point.
- private: Does not receive or forward any propagation events.
- unbindable: cannot be bind mounted.

HANDSON

Stream online

HANDSON



User Namespace (root)
PID
UTS
IPC
Net
CGroup
Mount
User ns (unpriviledge) unshare -Ur
/proc
Mount -m
Legend
Inheritance
Ownership