Implementing a rootless container manager from scratch

Lessons learned writing my own container manager: **lilipod**

:~$ whoami

- Luca Di Maio - 89luca89
- SWE @Chainguard
- Open Source Enthusiast

github.com/89uca89

@LucaDiMaio11

fosstodon.org/@89luca89

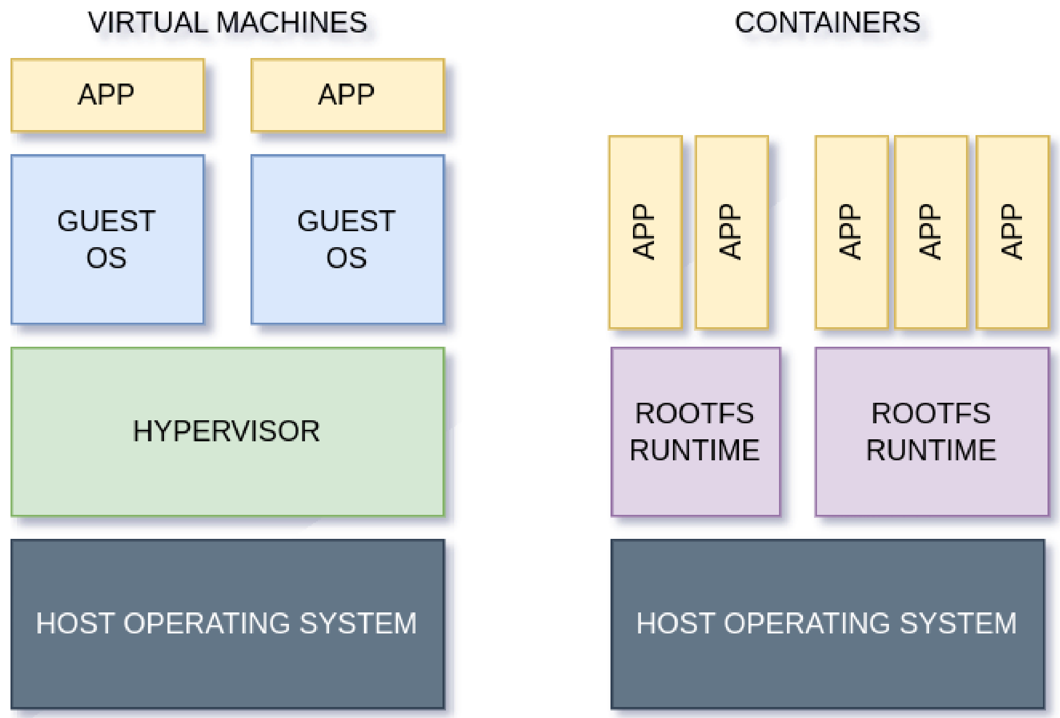
luca.dimaio1@gmail.com

lilipod

- Born as a support project for **Distrobox**
- Need was a self contained, lightweight and fast container manager
- Less features than Podman and Docker
  - Not in scope
  - Main target is lightness

- I wanted to know more about containers
- I wanted to improve my go

What are containers?



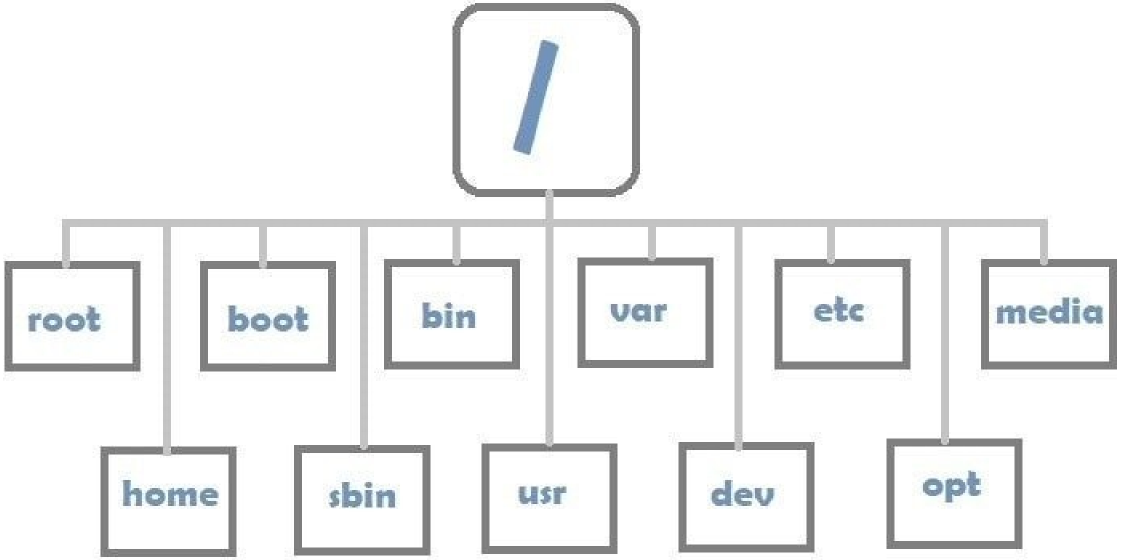
- Fast
- Light
- Disposable

What are containers?

- **Building blocks**
  - **Rootfs**
  - Namespaces
  - Capabilities
  - Cgroups
  - Seccomp filters
  - LSM (Selinux/Apparmor)

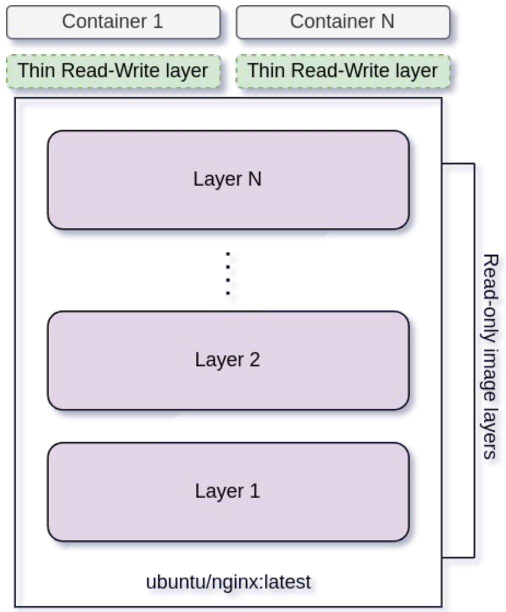
Rootfs

- Base file system for linux userland
- Distributed as **images** via OCI registries
  - dockerhub
  - quay.io
  - ghcr.io
  - gcr.io
  - public.ecr.aws



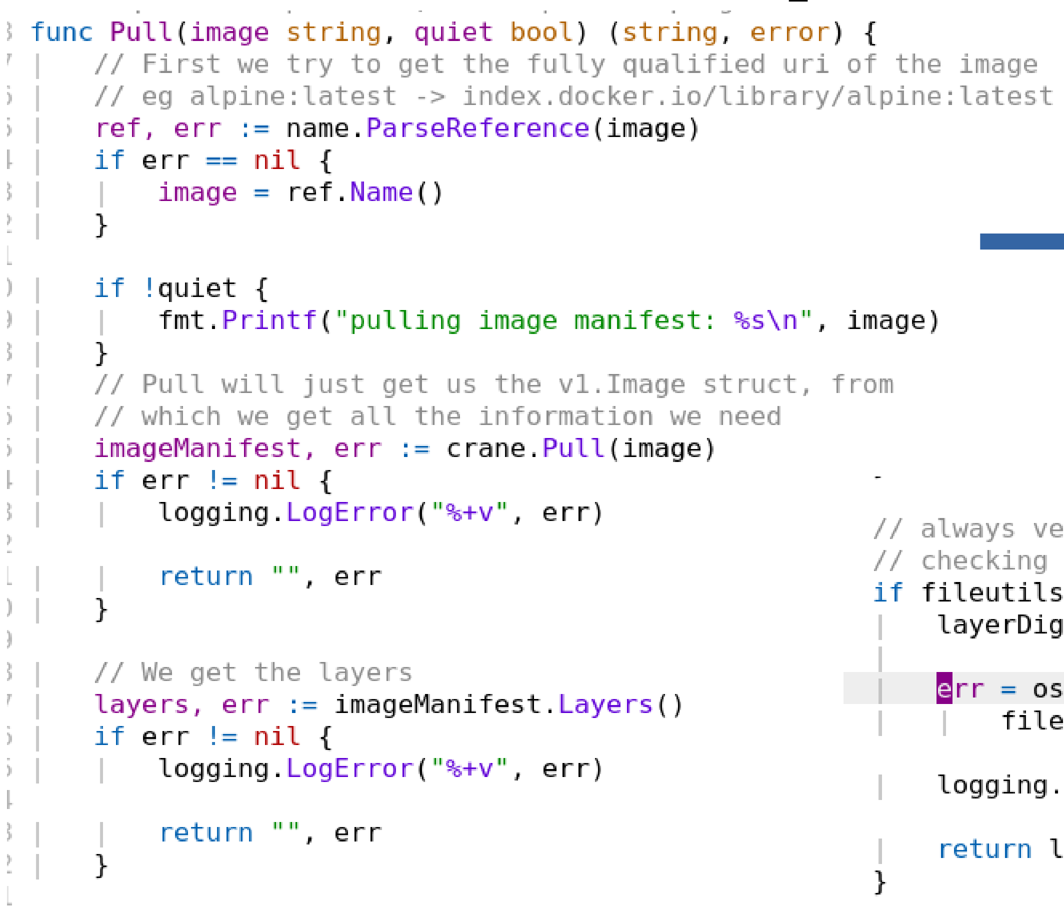
Rootfs

- json - Describes image composition
- Set of layers
- Tarballs
- Easy to dedup

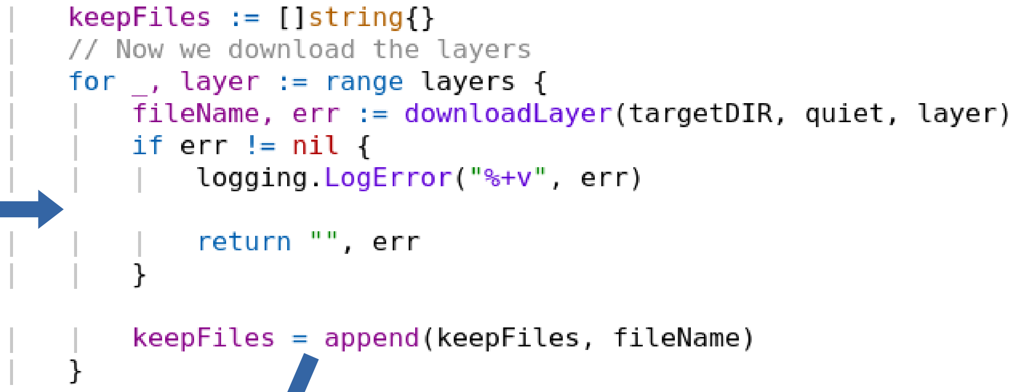


```
~$ docker manifest inspect nginx:latest
{
   "schemaVersion": 2,
   "mediaType": "application/vnd.oci.image.index.v1+json",
   "manifests": [
      {
         "mediaType": "application/vnd.oci.image.manifest.v1+json",
         "size": 2295,
         "digest": "sha256:3d696e8357051647b844d8c7cf4a0aa71e84379999a4f6af9b8ca1f7919ade42",
         "platform": {
            "architecture": "amd64",
            "os": "linux"
         }
      },
      {
         "mediaType": "application/vnd.oci.image.manifest.v1+json",
         "size": 841,
         "digest": "sha256:04ead2bc6e759e8e81d5ccfffba09138b98466f4c98918cbea8c802e4718b4b8",
         "platform": {
            "architecture": "unknown",
            "os": "unknown"
         }
      },
      {

        ...
        ...
},
      {
         "mediaType": "application/vnd.oci.image.manifest.v1+json",
         "size": 841,
         "digest": "sha256:9a688530c9457e4205bedf2a6d39e4fa5a3f3d56b522819f51261607ae2d0419",
         "platform": {
            "architecture": "unknown",
            "os": "unknown"
         }
      }
   ]
}
```

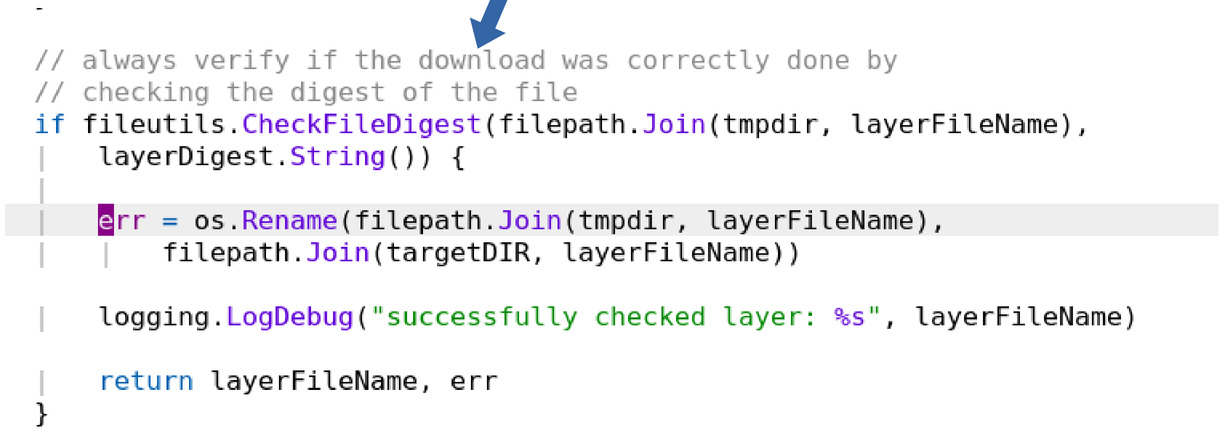
lilipod example



```go
func Pull(image string, quiet bool) (string, error) {
	// First we try to get the fully qualified uri of the image
	// eg alpine:latest -> index.docker.io/library/alpine:latest
	ref, err := name.ParseReference(image)
	if err == nil {
		image = ref.Name()
	}

	if !quiet {
		fmt.Printf("pulling image manifest: %s\n", image)
	}
	// Pull will just get us the v1.Image struct, from
	// which we get all the information we need
	imageManifest, err := crane.Pull(image)
	if err != nil {
		logging.LogError("%+v", err)

		return "", err
	}

	// We get the layers
	layers, err := imageManifest.Layers()
	if err != nil {
		logging.LogError("%+v", err)

		return "", err
	}
```



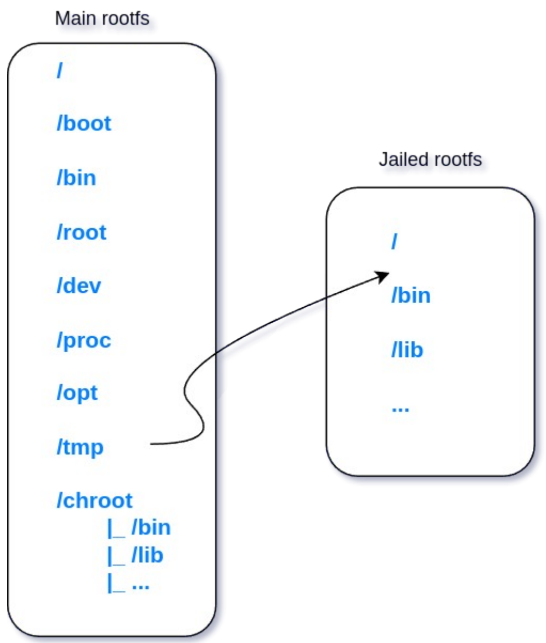
```go
	keepFiles := []string{}
	// Now we download the layers
	for _, layer := range layers {
		fileName, err := downloadLayer(targetDIR, quiet, layer)
		if err != nil {
			logging.LogError("%+v", err)

			return "", err
		}

		keepFiles = append(keepFiles, fileName)
	}
```



```go
// always verify if the download was correctly done by
// checking the digest of the file
if fileutils.CheckFileDigest(filepath.Join(tmpdir, layerFileName),
	layerDigest.String()) {

	err = os.Rename(filepath.Join(tmpdir, layerFileName),
		filepath.Join(targetDIR, layerFileName))

	logging.LogDebug("successfully checked layer: %s", layerFileName)

	return layerFileName, err
}
```

Using the rootfs

- **chroot**
- change rootfs for a process to a new directory
- good for:
  - restricting filesystem access
  - BYO distro for processes



Rootfs ready!

```
~$ chroot /tmp/rootfs
chroot: cannot change root directory to '/tmp/rootfs': Operation not permitted
```

Chroot

- for **chroot** we need:
  - to be the **root** user
  - to **mount** additional filesystems (sysfs, procfs, tmpfs…)

STOP!

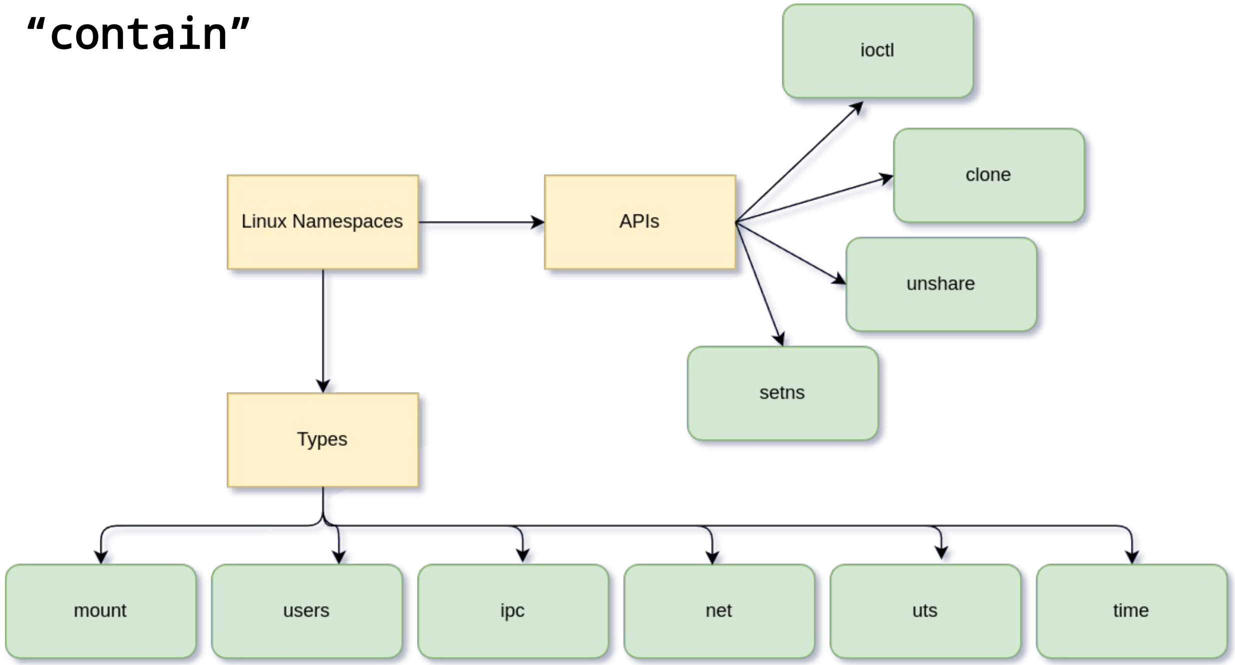
ROOTLESS TIME!

What are containers?

- **Building blocks**
  - Rootfs
  - **Namespaces**
  - Capabilities
  - Cgroups
  - Seccomp filters
  - LSM (Selinux/Apparmor)

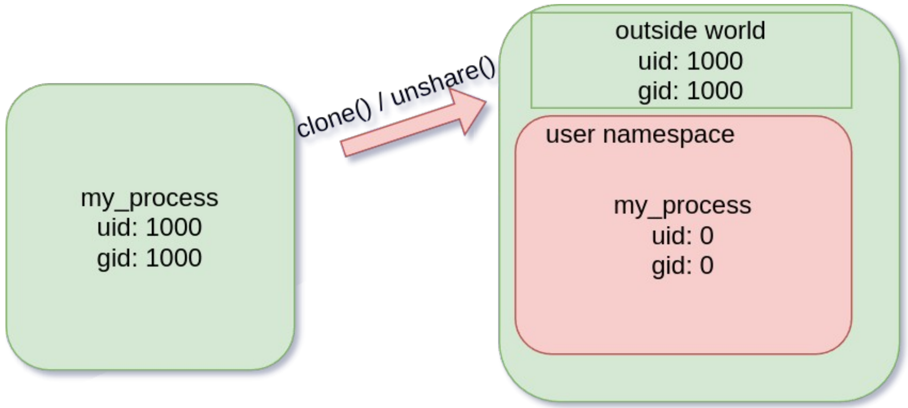
Namespaces

- Kernel provides process isolation means using **namespaces**
- Technology to provide isolated views of Linux resources
- **Basically: how containers “contain”**
  - **Mount** namespace
  - **User** namespace
  - **UTS** namespace
  - **PID** namespace
  - **IPC** namespace
  - **Network** namespace
  - **Time** namespace

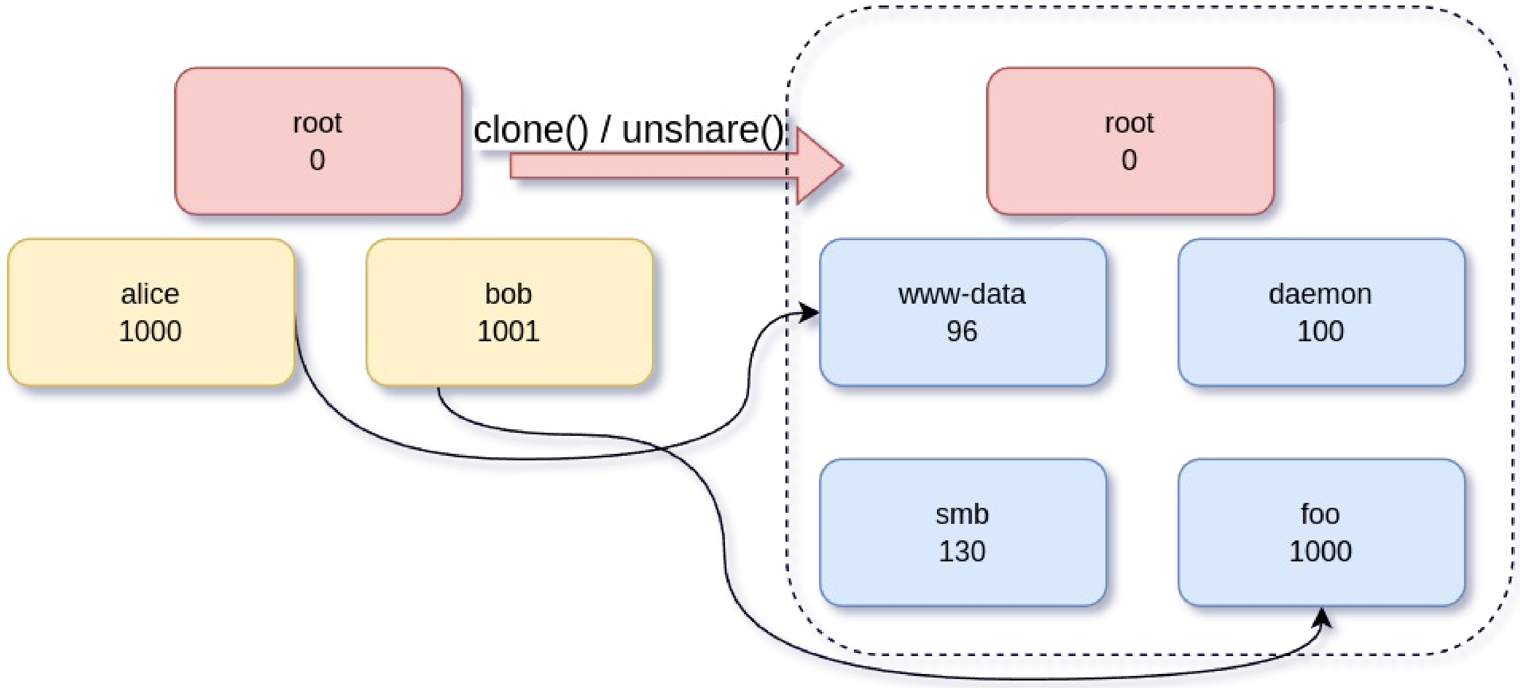


Namespaces

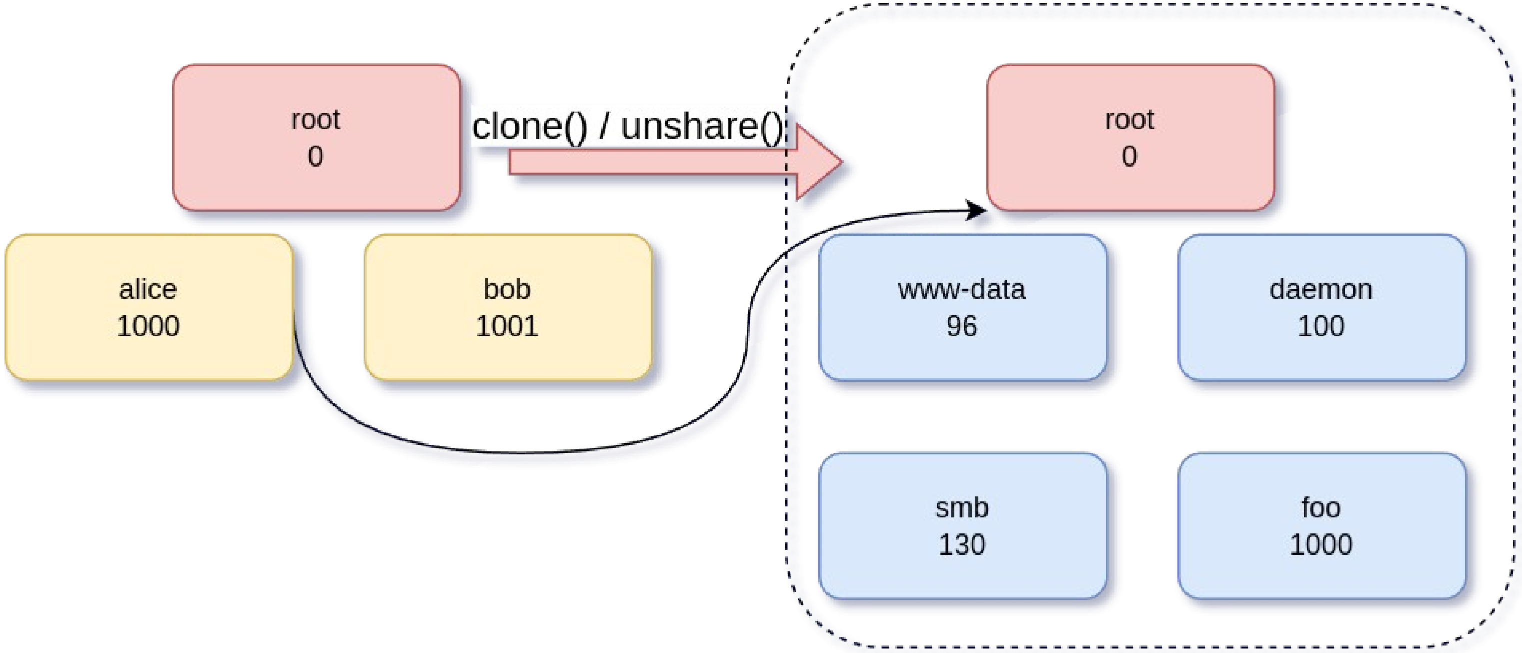
- call syscall **unshare,** **fork** the process
- child will live in its **own namespace**
  - rw local copy of mount tree
  - rw local copy of user tree



User Namespace



User Namespace



Namespaces going rootless

```
~$ unshare --mount --user --map-root-user chroot /tmp/rootfs /bin/sh -l
framework13:/# cat /etc/os-release
ID=wolfi
NAME="Wolfi"
PRETTY_NAME="Wolfi"
VERSION_ID="20230201"
HOME_URL="https://wolfi.dev"
BUG_REPORT_URL="https://github.com/wolfi-dev/os/issues"
```

lilipod example

```go
cloneFlags := syscall.CLONE_NEWUTS | syscall.CLONE_NEWNS

if config.Userns == constants.KeepID &&
    os.Getenv("ROOTFUL") != constants.TrueString {
    cloneFlags |= syscall.CLONE_NEWUSER
}

if config.Ipc == constants.Private {
    cloneFlags |= syscall.CLONE_NEWIPC
}

if config.Network == constants.Private {
    cloneFlags |= syscall.CLONE_NEWNET
}

if config.Pid == constants.Private {
    cloneFlags |= syscall.CLONE_NEWPID
}

if config.Cgroup == constants.Private {
    cloneFlags |= syscall.CLONE_NEWCGROUP
}

if config.Time == constants.Private {
    cloneFlags |= syscall.CLONE_NEWTIME
}
```

```go
// Set Namespaces with generated value,
// this value will keep-id with the host.
cmd.SysProcAttr = &syscall.SysProcAttr{
    Credential:                 &syscall.Credential{Uid: 0, Gid: 0},
    Cloneflags:                 uintptr(cloneFlags),
    GidMappingsEnableSetgroups: true,

    Setsid:     true,
    Foreground: false,
    Pdeathsig:  syscall.SIGTERM,
}
```

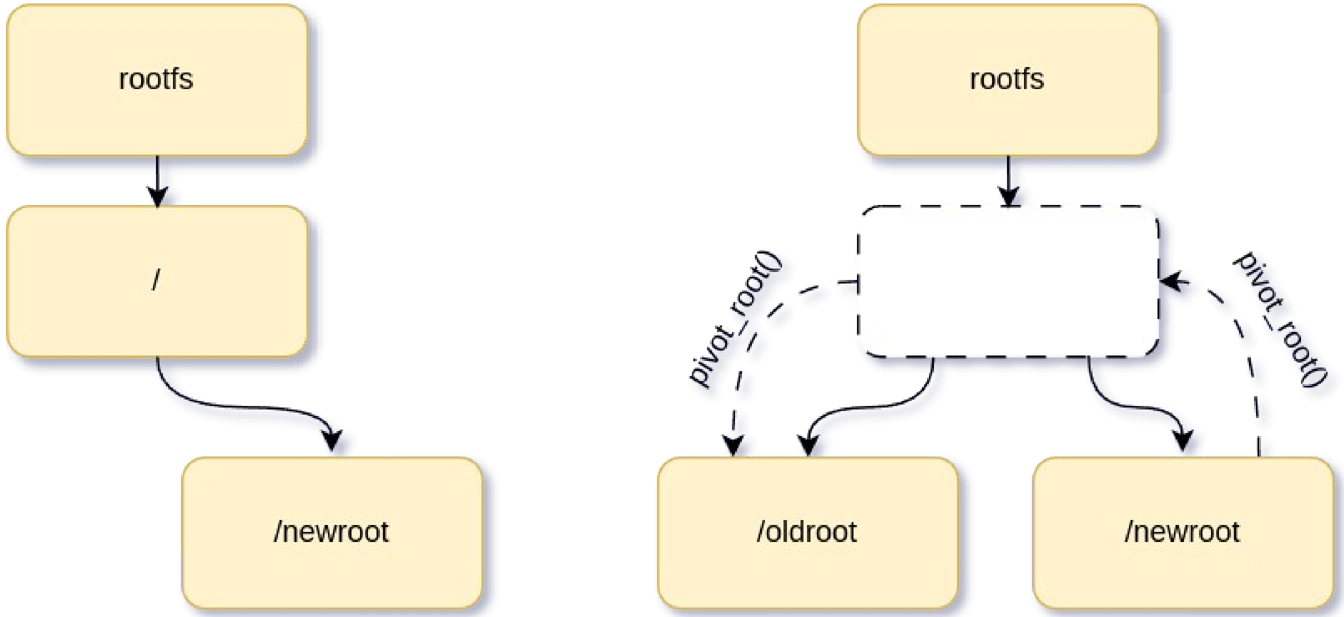
```go
logging.LogDebug("pivotroot: pivot from %s to %s", path, pivotDir)

err = syscall.PivotRoot(path, pivotDir)
if err != nil {
    logging.LogDebug("error: %+v", err)

    return fmt.Errorf("pivotroot: %w", err)
}

// path to pivot dir now changed, update
pivotDir = filepath.Join("/", filepath.Base(pivotDir))
```

Chroot caveats

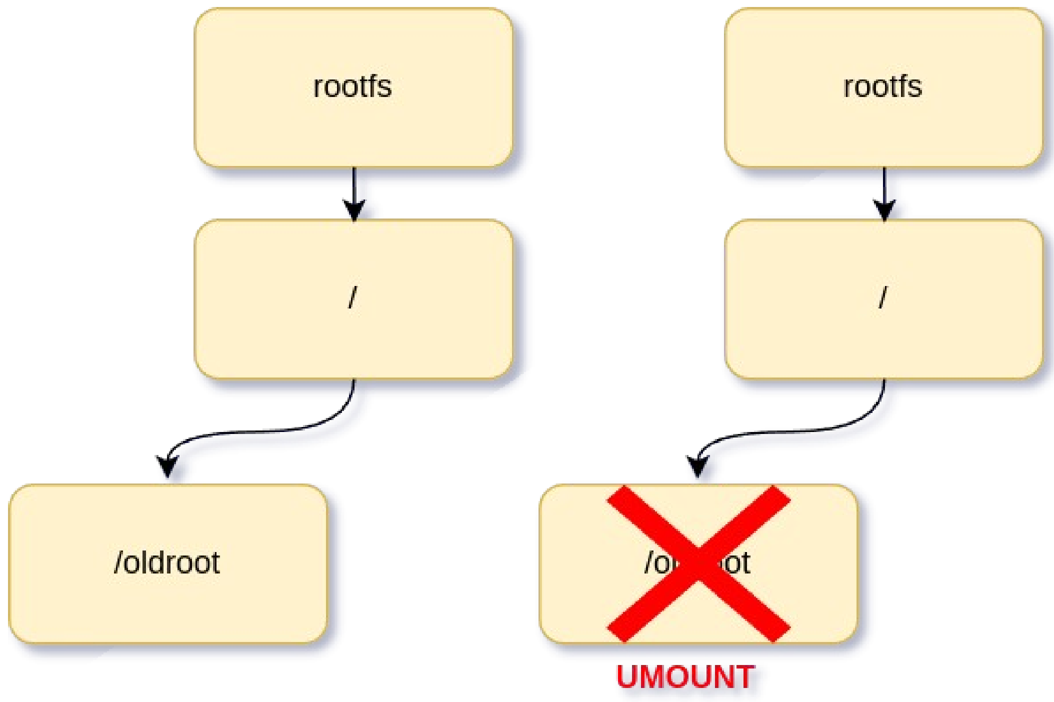
- chroot can be **escaped easily**:
  - for (int i = 0; i < 1024; ++i) {chdir(".."); chroot(“.”); }

- **pivot_root** is a different approach
  - switches the directory as the **root of the mount tree**
  - can leverage mount namespace to **unmount the old rootfs**
  - original rootfs is completely **not accessible anymore**

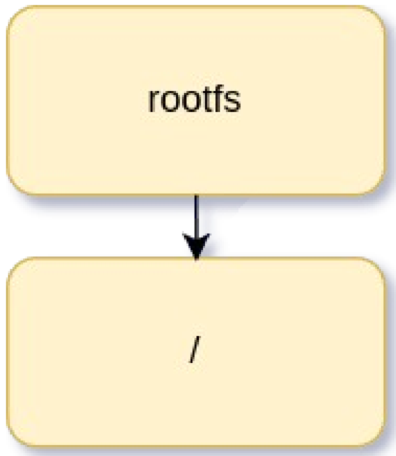
Pivot Root


rootfs
/
/newroot
rootfs
pivot_root()
pivot_root()
/oldroot
/newroot

Pivot Root



Pivot Root



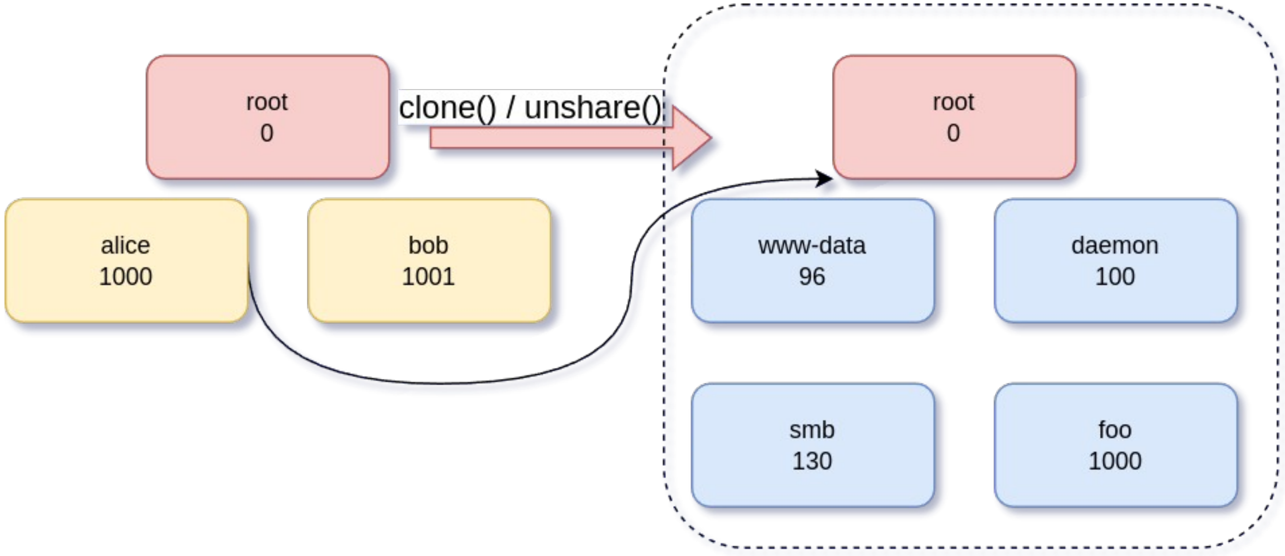
User Namespace caveats

```
unshare --mount --pid --fork --user --map-root-user chroot rootfs /bin/bash -l
root@framework13:/# mount -t proc proc /proc/
root@framework13:/# cat /proc/self/uid_map
         0       1000          1
root@framework13:/# cat /proc/1/setgroups
deny
root@framework13:/# apt update
E: setgroups 65534 failed - setgroups (1: Operation not permitted)
E: setegid 65534 failed - setegid (22: Invalid argument)
E: seteuid 42 failed - seteuid (22: Invalid argument)
E: setgroups 0 failed - setgroups (1: Operation not permitted)
rm: cannot remove '/var/cache/apt/archives/partial/*.deb': Permission denied
Reading package lists... Done
W: chown to _apt:root of directory /var/lib/apt/lists/partial failed - SetupAPTPartialDirectory (22: Invalid argument)
W: chown to _apt:root of directory /var/lib/apt/lists/auxfiles failed - SetupAPTPartialDirectory (22: Invalid argument)
E: setgroups 65534 failed - setgroups (1: Operation not permitted)
E: setegid 65534 failed - setegid (22: Invalid argument)
E: seteuid 42 failed - seteuid (22: Invalid argument)
E: setgroups 0 failed - setgroups (1: Operation not permitted)
E: Method gave invalid 400 URI Failure message: Failed to setgroups - setgroups (1: Operation not permitted)
E: Method gave invalid 400 URI Failure message: Failed to setgroups - setgroups (1: Operation not permitted)
E: Method http has died unexpectedly!
E: Sub-process http returned an error code (112)
```

User Namespace caveats

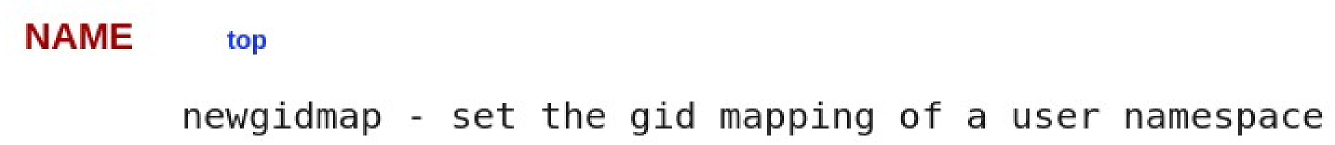
- Single user mapping

```
root@framework13:/# cat /proc/self/uid_map
         0       1000          1
root@framework13:/# cat /proc/1/setgroups
deny
```



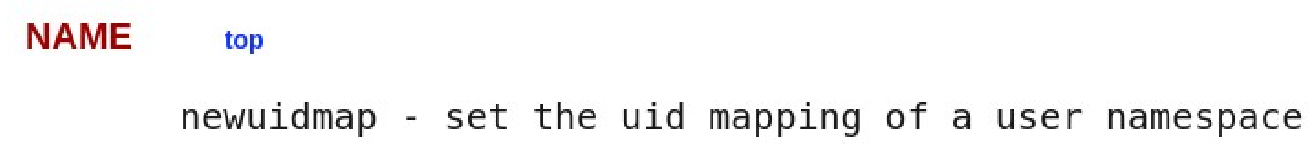
User Namespace caveats

- Does not allow extra groups and users
- We need **subuid** and **subgid** maps



NAME top

```
newgidmap - set the gid mapping of a user namespace
```



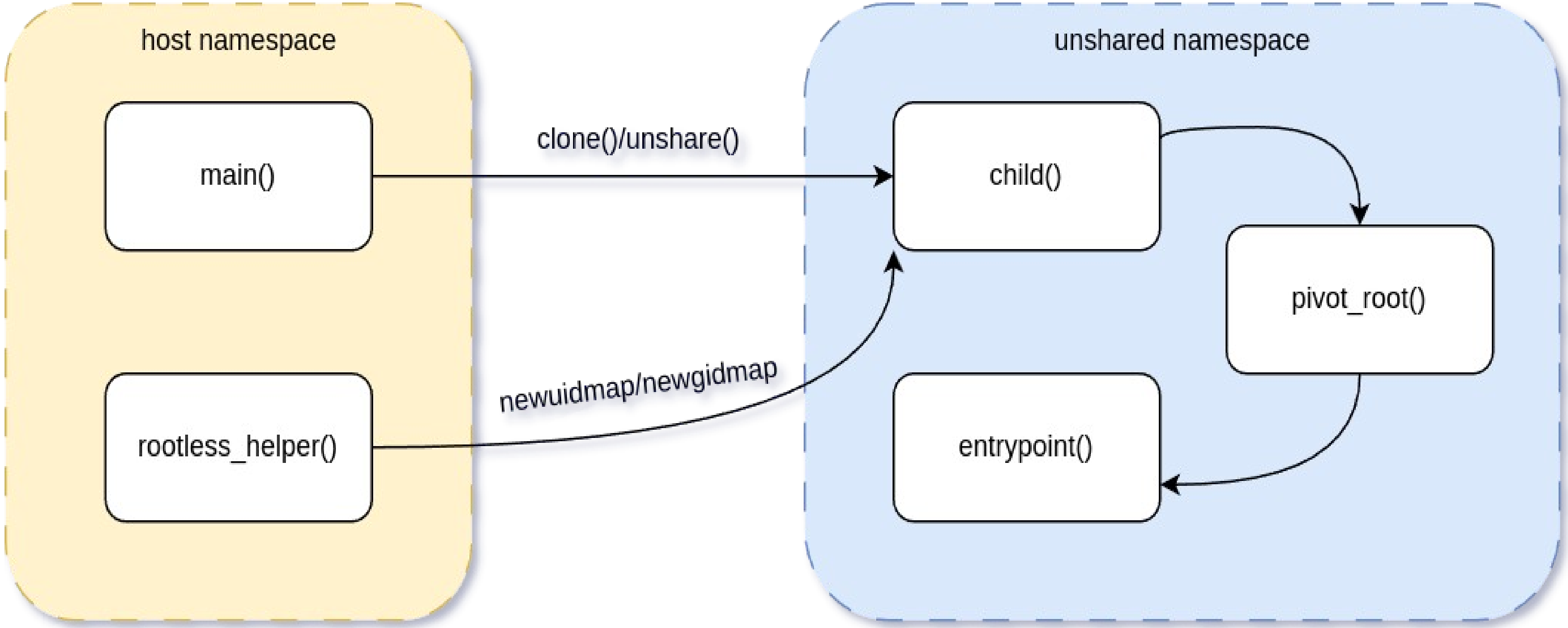
NAME top

```
newuidmap - set the uid mapping of a user namespace
```

User Namespace helper

- Only **root** can map multiple IDs and let **setgroups**
- newuidmap/newgidmap are **setuid binaries**
    - Launch **newuidmap/newgidmap** on unshared process
    - Process will have range of uid/gid to use

User Namespace helper



lilipod example

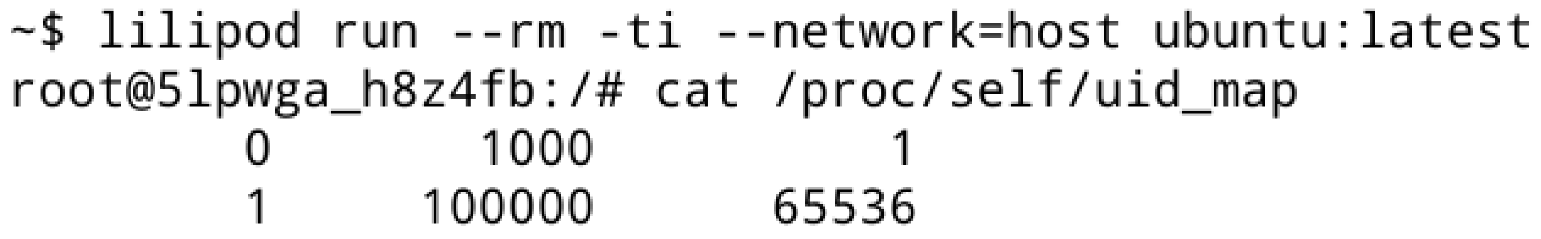
```go
uidMap := []string{
    strconv.Itoa(pid),
    "0",
    uMaps[0],
    "1",
    "1",
    uMaps[1],
    uMaps[2],
}
gidMap := []string{
    strconv.Itoa(pid),
    "0",
    gMaps[0],
    "1",
    "1",
    gMaps[1],
    gMaps[2],
}
```

```go
cmd := exec.Command("newuidmap", uidMap...)

logging.LogDebug("setting uidmap: executing %v", cmd.Args)

out, err := cmd.CombinedOutput()
if err != nil {
    log.Fatal(string(out))

    return err
}
```

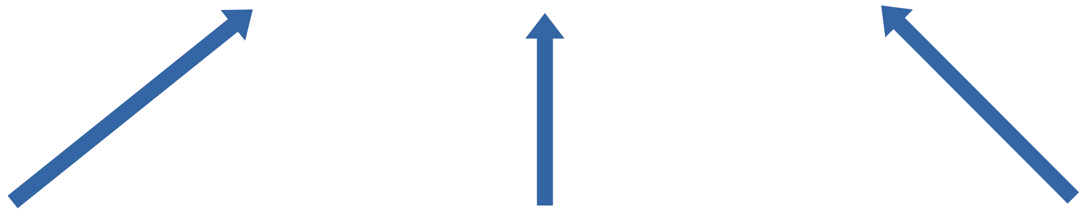
```
exec: newuidmap <child_pid> 0 1000 1 1 100000 65536
```

```
~$ lilipod run --rm -ti --network=host ubuntu:latest
root@5lpwga_h8z4fb:/# cat /proc/self/uid_map
         0       1000          1
         1     100000      65536
```

User Namespace helper



```
~$ lilipod run --rm -ti --network=host ubuntu:latest
root@5lpwga_h8z4fb:/# cat /proc/self/uid_map
         0       1000          1
         1     100000      65536
```

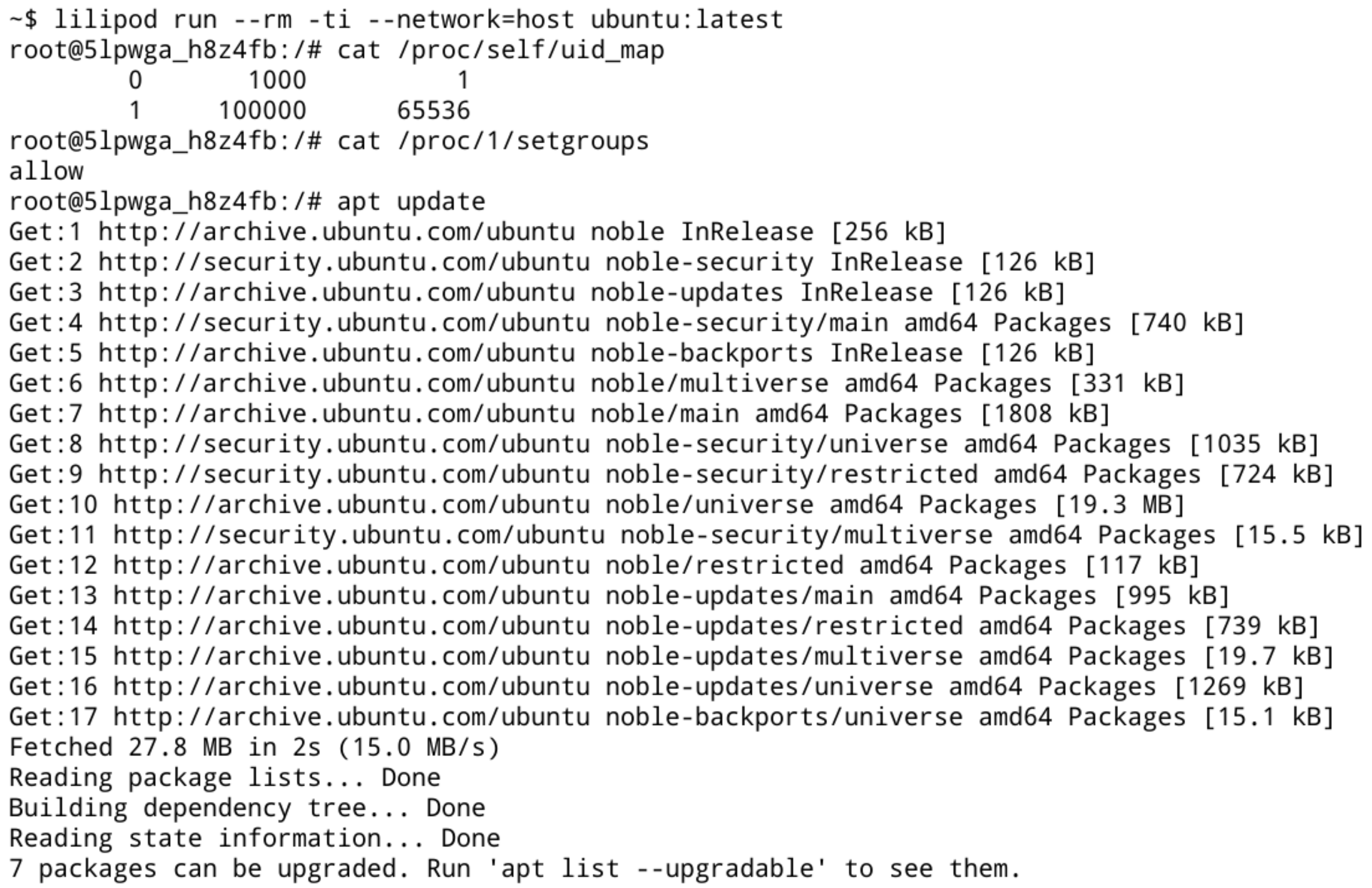


Start of the namespace's IDs range

Start of the main IDs range mapped to

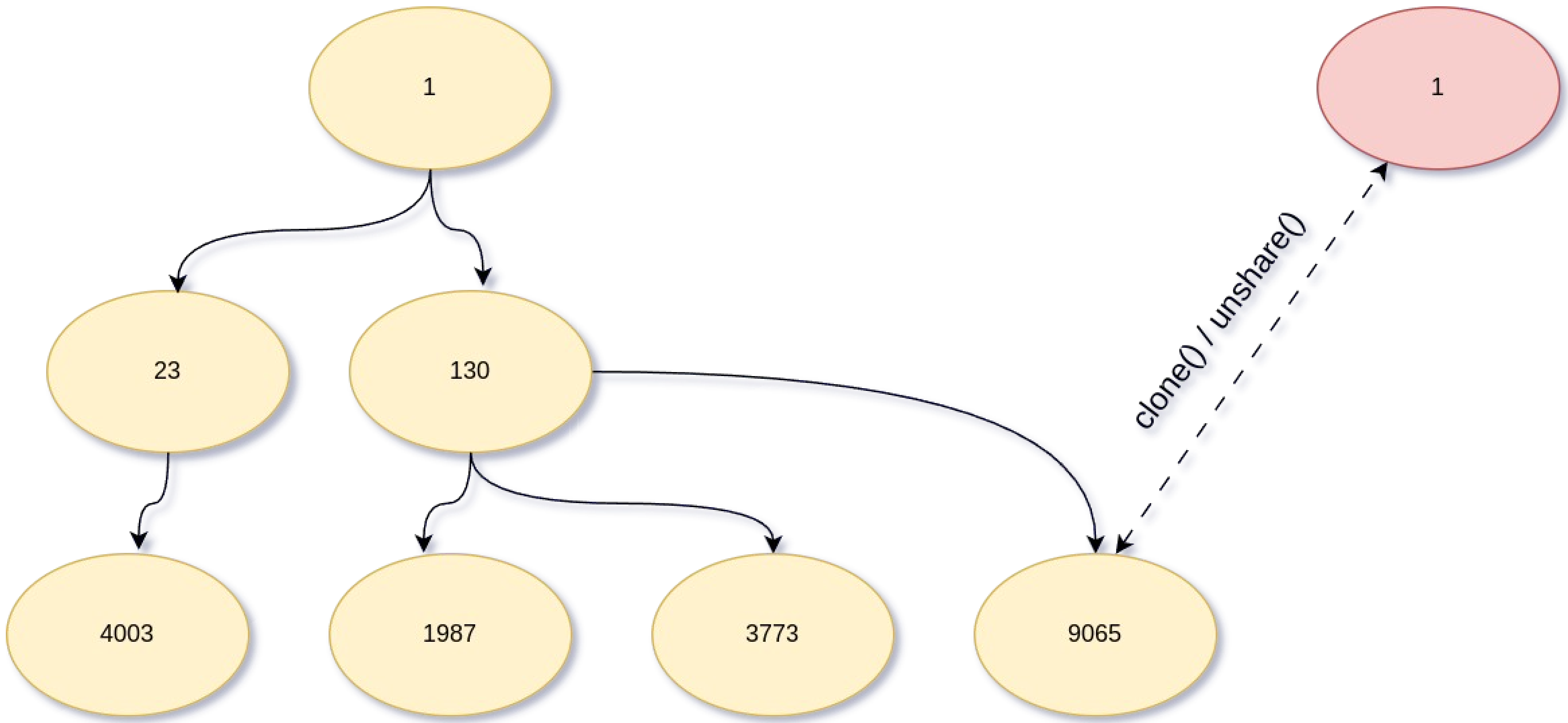
Size of the range to map

User Namespace helper

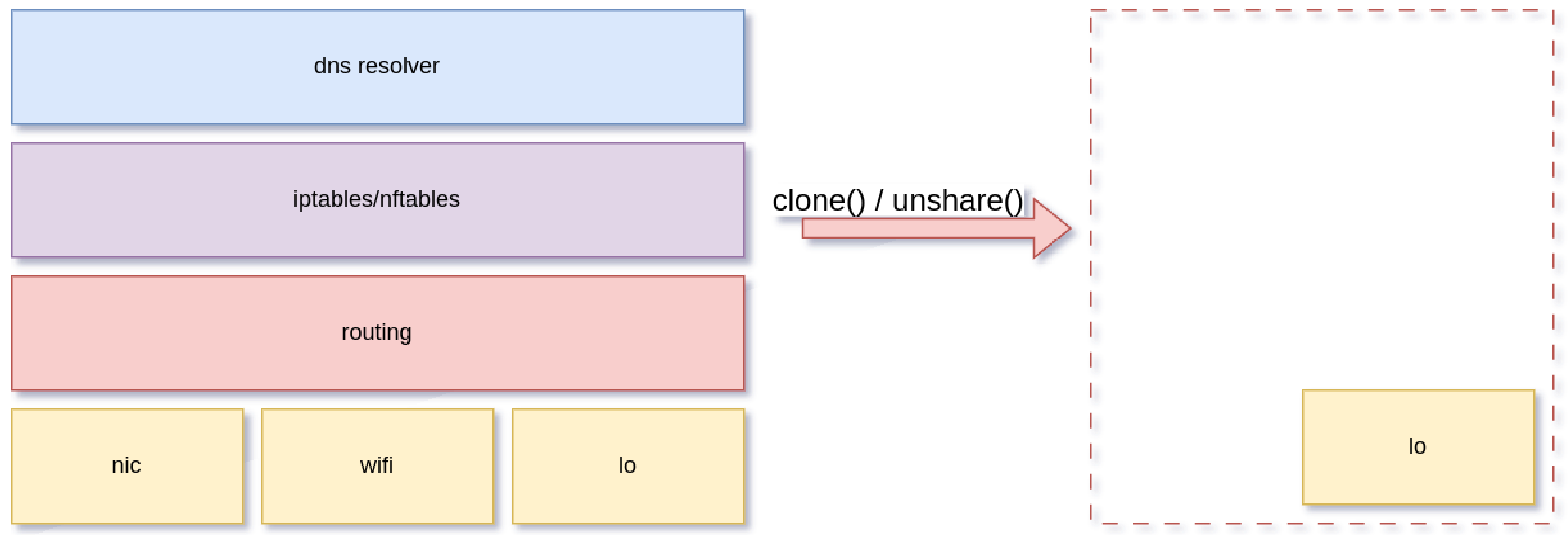


```
~$ lilipod run --rm -ti --network=host ubuntu:latest
root@5lpwga_h8z4fb:/# cat /proc/self/uid_map
        0       1000          1
        1     100000      65536
root@5lpwga_h8z4fb:/# cat /proc/1/setgroups
allow
root@5lpwga_h8z4fb:/# apt update
Get:1 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:2 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:4 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [740 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble/multiverse amd64 Packages [331 kB]
Get:7 http://archive.ubuntu.com/ubuntu noble/main amd64 Packages [1808 kB]
Get:8 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [1035 kB]
Get:9 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [724 kB]
Get:10 http://archive.ubuntu.com/ubuntu noble/universe amd64 Packages [19.3 MB]
Get:11 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [15.5 kB]
Get:12 http://archive.ubuntu.com/ubuntu noble/restricted amd64 Packages [117 kB]
Get:13 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [995 kB]
Get:14 http://archive.ubuntu.com/ubuntu noble-updates/restricted amd64 Packages [739 kB]
Get:15 http://archive.ubuntu.com/ubuntu noble-updates/multiverse amd64 Packages [19.7 kB]
Get:16 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1269 kB]
Get:17 http://archive.ubuntu.com/ubuntu noble-backports/universe amd64 Packages [15.1 kB]
Fetched 27.8 MB in 2s (15.0 MB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
7 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

PID Namespace



Network Namespace

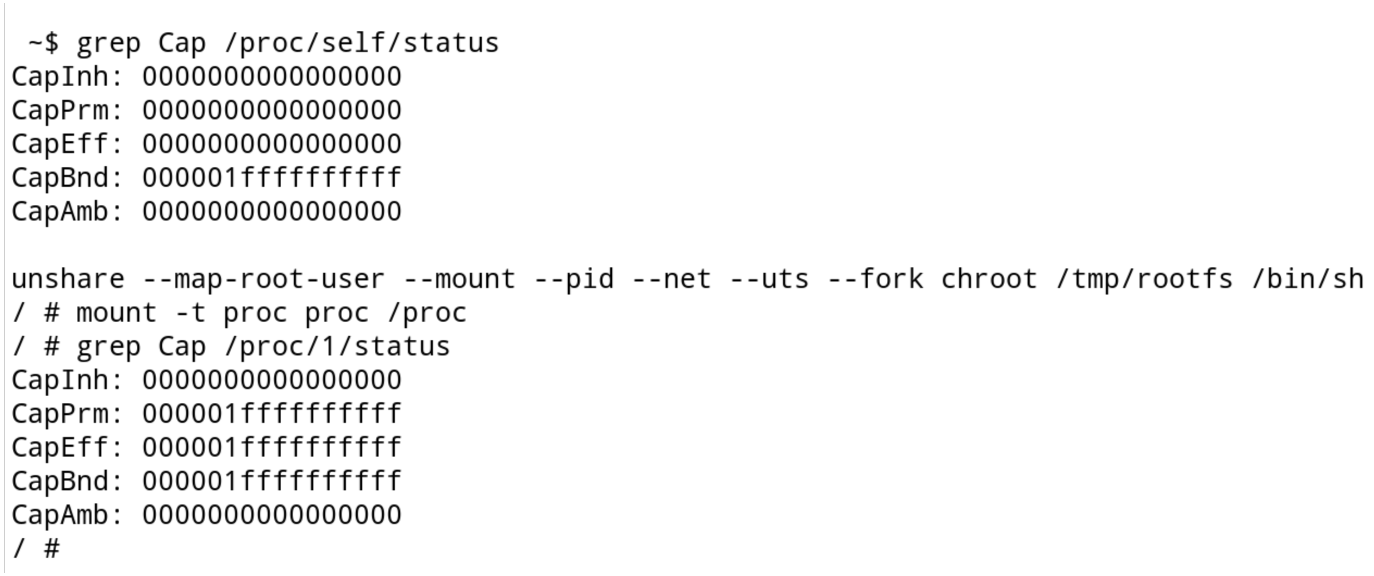


What are containers?

- **Building blocks**
  - Rootfs
  - Namespaces
  - **Capabilities**
  - Cgroups
  - Seccomp filters
  - LSM (Selinux/Apparmor)

Capabilities

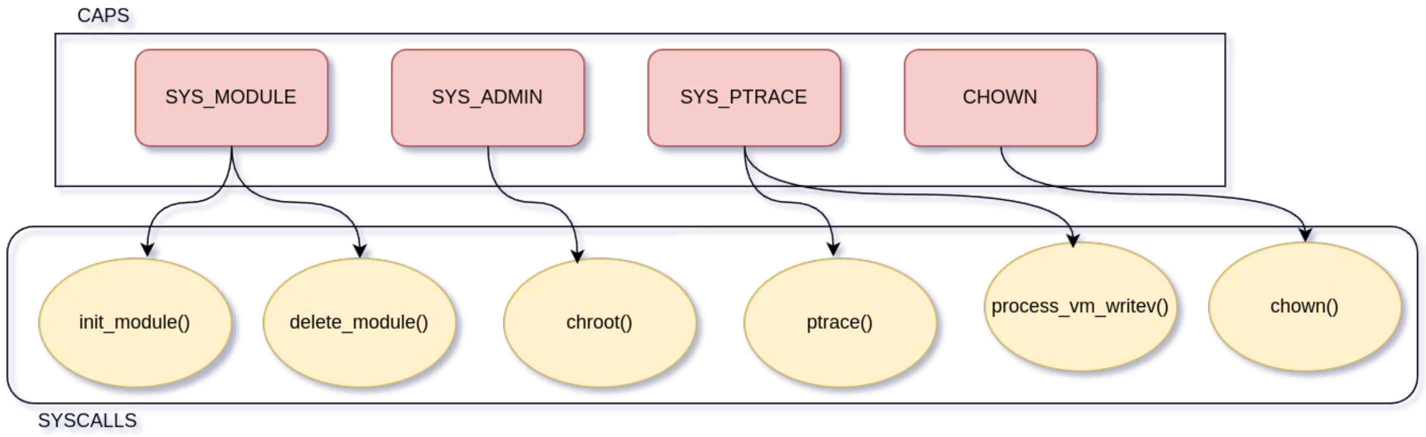
- Starting with kernel 2.2, Linux separates privileges into distinct units which can be independently enabled and disabled
- **Namespaces are not enough**
- Container is now defaulting to **all capabilities allowed**



```
 ~$ grep Cap /proc/self/status
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 000001ffffffffff
CapAmb: 0000000000000000

unshare --map-root-user --mount --pid --net --uts --fork chroot /tmp/rootfs /bin/sh
/ # mount -t proc proc /proc
/ # grep Cap /proc/1/status
CapInh: 0000000000000000
CapPrm: 000001ffffffffff
CapEff: 000001ffffffffff
CapBnd: 000001ffffffffff
CapAmb: 0000000000000000
/ #
```

Capabilities

- Too many can lead to container escape
- Eg:
  - load evil kernel module (CAP_SYS_MODULE)
  - mount and chroot via procfs (CAP_SYS_ADMIN + unshared PID ns)
- **Drop unneeded caps**



lilipod example

```go
var keepCaps = []string{
	"chown",
	"dac_override",
	"fsetid",
	"fowner",
	"mknod",
	"net_raw",
	"setgid",
	"setuid",
	"setfcap",
	"setpcap",
	"net_bind_service",
	"sys_chroot",
	"kill",
	"audit_write",
}
```

```go
knownCapsList := capability.ListKnown()
for _, capSpec := range keepCaps {
	// nocap
	capToSet := capability.Cap(-1)
	for _, c := range knownCapsList {
		if strings.EqualFold(c.String(), capSpec) {
			capToSet = c
			break
		}
	}
	caps.Set(capability.BOUNDING, capToSet)
	caps.Set(capability.EFFECTIVE, capToSet)
	caps.Set(capability.PERMITTED, capToSet)
}
```

```go
setCapabilities(keepCaps...)
return syscall.Exec(commandPath,
	conf.Entrypoint, conf.Env)
```

```
lilipod run --rm -ti cgr.dev/chainguard/wolfi-base:latest
4is8g9_n0708h:/# grep Cap /proc/1/status
CapInh:     0000000000000000
CapPrm:     00000000a80425fb
CapEff:     00000000a80425fb
CapBnd:     00000000a80425fb
CapAmb:     0000000000000000
4is8g9_n0708h:/#
```

Almost there

```
~$ lilipod pull cgr.dev/chainguard/wolfi-base:latest
pulling image manifest: cgr.dev/chainguard/wolfi-base:latest
pulling layer [...].tar.gz
Copying blob sha256:[...] 100% |██████████████████████████████| (42 MB/s)
saving layer sha256:[...] done
saving manifest for cgr.dev/chainguard/wolfi-base:latest
saving config for cgr.dev/chainguard/wolfi-base:latest
saving metadata for cgr.dev/chainguard/wolfi-base:latest
done
b3ba229143b1f6062d17c0da67192fe9

~$ lilipod run --rm -ti cgr.dev/chainguard/wolfi-base:latest
px9hht_s35yqe:/# id
uid=0(root) gid=0(root)
px9hht_s35yqe:/# ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 /sbin/pty /bin/sh -l
   15 root      0:00 /bin/sh -l
   18 root      0:00 ps aux
```

```
px9hht_s35yqe:/# cat /proc/self/uid_map
         0       1000          1
         1     100000      65536
px9hht_s35yqe:/# cat /proc/1/setgroups
allow
px9hht_s35yqe:/# grep Cap /proc/1/status
CapInh:    0000000000000000
CapPrm:    00000000a80425fb
CapEff:    00000000a80425fb
CapBnd:    00000000a80425fb
CapAmb:    0000000000000000
px9hht_s35yqe:/# ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Still not enough

- **Building blocks**
  - Rootfs
  - Namespaces
  - Capabilities
  - **Cgroups**
  - **Seccomp filters**
  - **LSM (Selinux/Apparmor)**

https://github.com/
89luca89/lilipod

Thanks !

Any
Questions?