Deep dive into network namespace and container networking

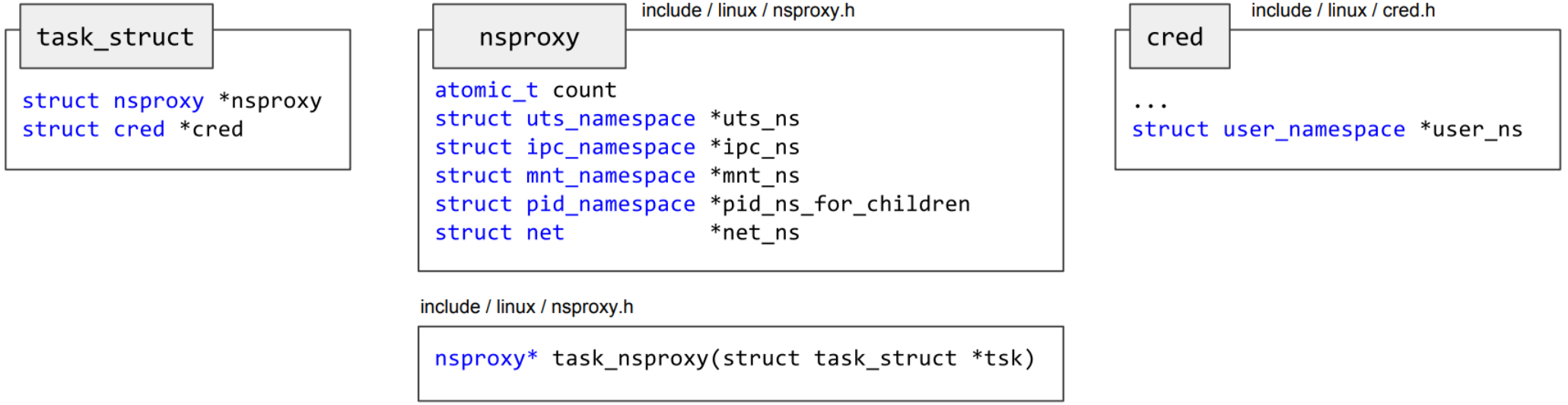
CS695 – Topics in Virtualization and Cloud Computing

Debojeet Das

Understanding namespace bottom up

Namespaces ~ ways to isolate resources among userspace applications.

How are they implemented in linux kernel?

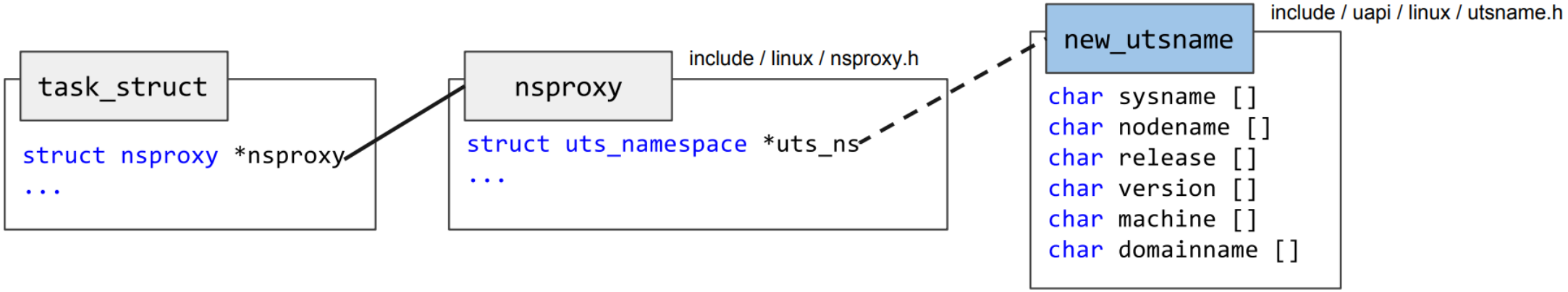



```
# ls -al /proc/1/ns
# ls -al /proc/<any-valid-pid>/ns
```

Understanding namespace bottom up

Namespaces ~ ways to isolate resources among userspace applications.

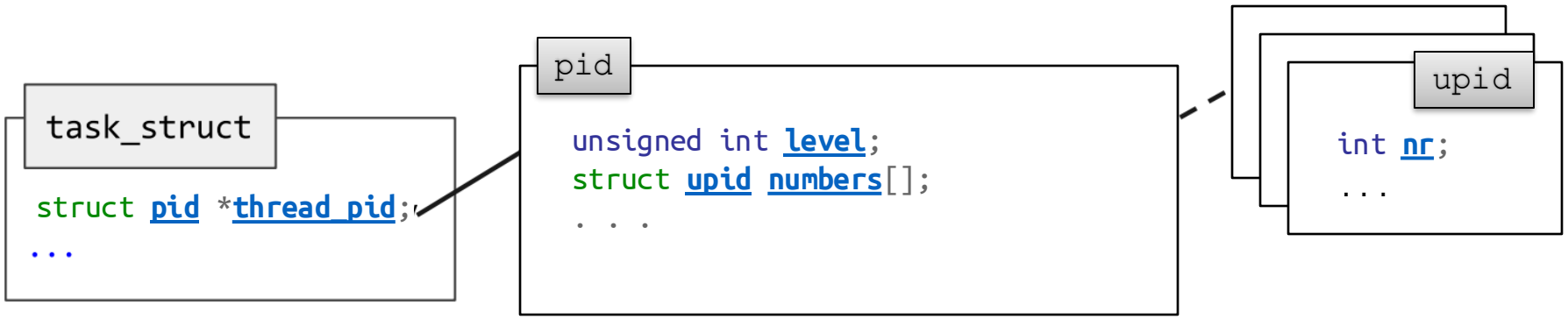
Let’s take example of uts namespace.



Understanding namespace bottom up

Namespaces ~ ways to isolate resources among userspace applications.

Working of pid namespaces.



Understanding namespace bottom up

Namespaces ~ ways to isolate resources among userspace applications.

Let’s talk about network namespace now.

What kind of resources will be present in network namespace?

- Loopback device.
- List of physical and virtual devices.
- Network configurations (ipv4, ipv6, etc.)
- Ip table rules, etc.

Check include/net/net_namespace.h

An ip tool detour

ip tools allows you to create network namespaces easily.

```
# ip netns ls
# cat /var/run/netns
# ip netns add test
# cat /var/run/netns
```

What exactly happened when we added a new network namespace?

```
# strace ip netns add newtest
```

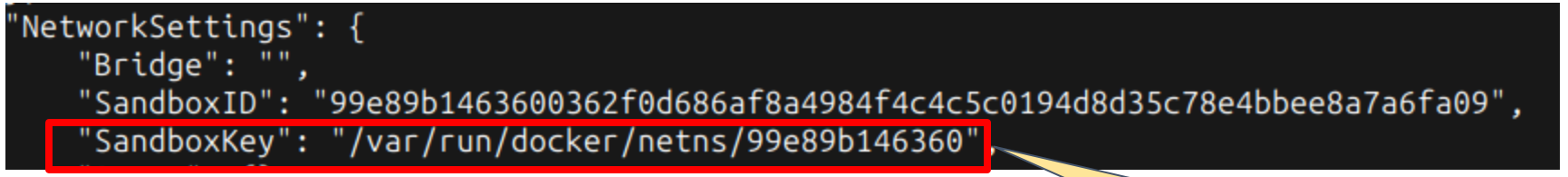
An ip tool detour

ip tools allows you to create network namespaces easily.

```
$ ip a
$ sudo ip netns exec test /bin/bash
# ip a
# exit
$ sudo ip netns del test
```

Understanding Docker networking

```
$ docker run -p 80:8000 --name cont-test hellocs695:latest
$ docker ps
$ docker inspect cont-test
```



network namespace inode
(can be linked to /var/run/netns for netns usage)

```
$ sudo mkdir /var/run/netns
$ sudo ln -s <sandbox-key> /var/run/netns/test
$ ip netns ls
```

Understanding Docker networking

```
$ docker exec -it cont-test /bin/bash
# apt install iproute2 (if not available)
$ ip a
```

The shell is in docker container's network namespaces but nothing else is isolated!

You can do whatever you want to mess up containers network setup.
Let's try to add a new interface

```
$ sudo ip link add dev inside-test type veth peer name outside-test
netns test
```

Docker Internals - Networking

Docker's networking options:

1. **Bridge:** The default network driver. (Docker0 / Custom)
2. **Host:** Use host network namespace.
3. **None:** Completely isolate a container from the host and other containers.

Other options exists such as **Overlay, Ipvlan, and Macvlan**.

Docker Internals - Networking

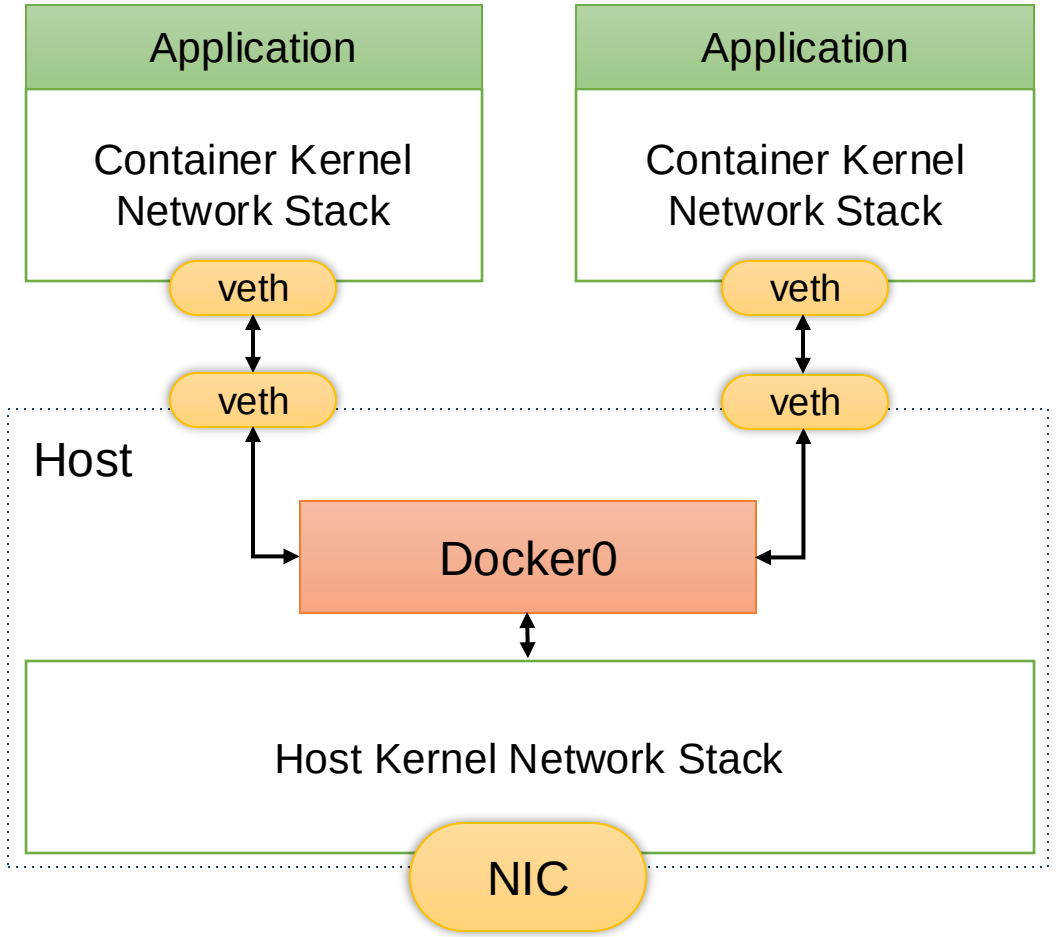
Dcker's networking options:

Bridge: Docker0

```
$ docker run –it ubuntu sh
```

In host:

```
$ sudo iptables -L
```



Docker Internals - Networking

Dcker's networking options:

Bridge: User-defined

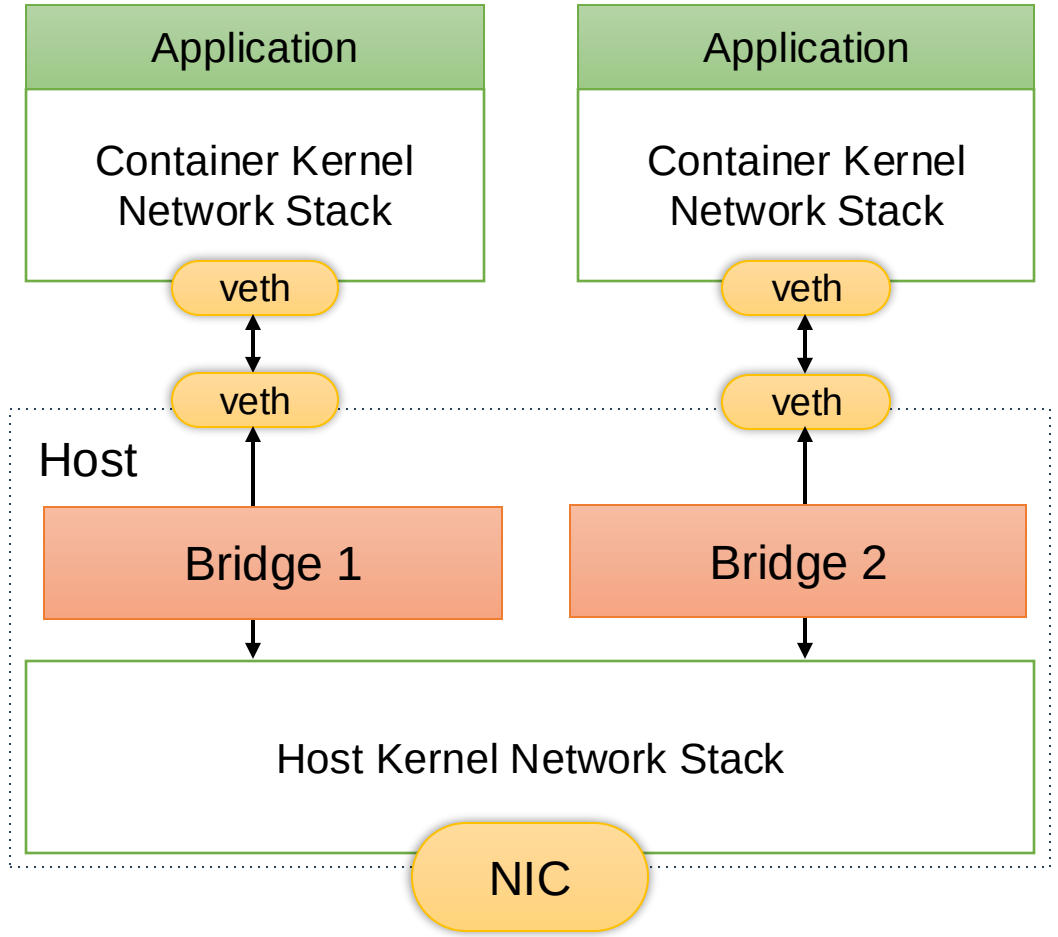
```
$ docker network create my-net

$ docker network connect my-net <c-name>

$ sudo iptables -L

$ docker run -it --name
my-ubuntu \
--network my-net \
--publish 8080:80 \
ubuntu bin/bash

$ sudo iptables -L
```



Understanding Linux capabilities

Permissions in Linux isn’t divided into Privileged and unprivileged. But into several capabilities.

Unprivileged capabilities:

Capability Key	Capability Description
AUDIT_WRITE	Write records to kernel auditing log.
CHOWN	Make arbitrary changes to file UIDs and GIDs (see chown(2)).
DAC_OVERRIDE	Bypass file read, write, and execute permission checks.
FOWNER	Bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
FSETID	Don't clear set-user-ID and set-group-ID permission bits when a file is modified.
KILL	Bypass permission checks for sending signals.
MKNOD	Create special files using mknod(2).
NET_BIND_SERVICE	Bind a socket to internet domain privileged ports (port numbers less than 1024).
NET_RAW	Use RAW and PACKET sockets.
SETFCAP	Set file capabilities.
SETGID	Make arbitrary manipulations of process GIDs and supplementary GID list.
SETPCAP	Modify process capabilities.
SETUID	Make arbitrary manipulations of process UIDs.
SYS_CHROOT	Use chroot(2), change root directory.

Understanding Linux capabilities

Permissions in Linux isn’t divided into Privileged and unprivileged. But into several capabilities.

Privileged capabilities:

There are many more

Check capabilities of your process:
$ /proc/<pid>/status

Capability Key	Capability Description
AUDIT_CONTROL	Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.
AUDIT_READ	Allow reading the audit log via multicast netlink socket.
BLOCK_SUSPEND	Allow preventing system suspends.
BPF	Allow creating BPF maps, loading BPF Type Format (BTF) data, retrieve JITed code of BPF programs, and more.
CHECKPOINT_RESTORE	Allow checkpoint/restore related operations. Introduced in kernel 5.9.
DAC_READ_SEARCH	Bypass file read permission checks and directory read and execute permission checks.
IPC_LOCK	Lock memory (mlock(2), mlockall(2), mmap(2), shmctl(2)).
IPC_OWNER	Bypass permission checks for operations on System V IPC objects.
LEASE	Establish leases on arbitrary files (see fcntl(2)).
LINUX_IMMUTABLE	Set the FS_APPEND_FL and FS_IMMUTABLE_FL i-node flags.
MAC_ADMIN	Allow MAC configuration or state changes. Implemented for the Smack LSM.
MAC_OVERRIDE	Override Mandatory Access Control (MAC). Implemented for the Smack Linux Security Module (LSM).
NET_ADMIN	Perform various network-related operations.
NET_BROADCAST	Make socket broadcasts, and listen to multicasts.

Docker capabilities

Option	Description
`--cap-add`	Add Linux capabilities
`--cap-drop`	Drop Linux capabilities
`--privileged`	Give extended privileges to this container
`--device=[]`	Allows you to run devices inside the container without the `--privileged` flag.

[Image credit: https://twitter.com/sidpalas/status/1638558418830233600]

Thank You!!