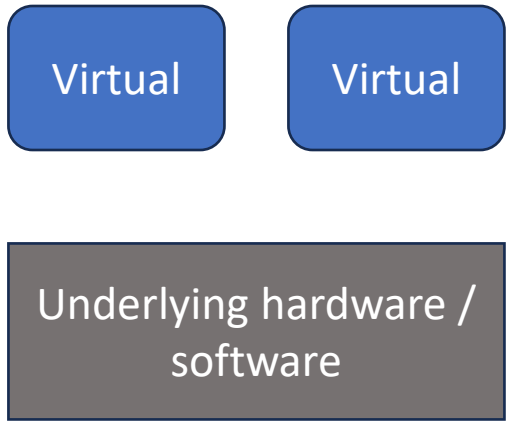
Problem / Overview

Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux

University of Colorado **Boulder**

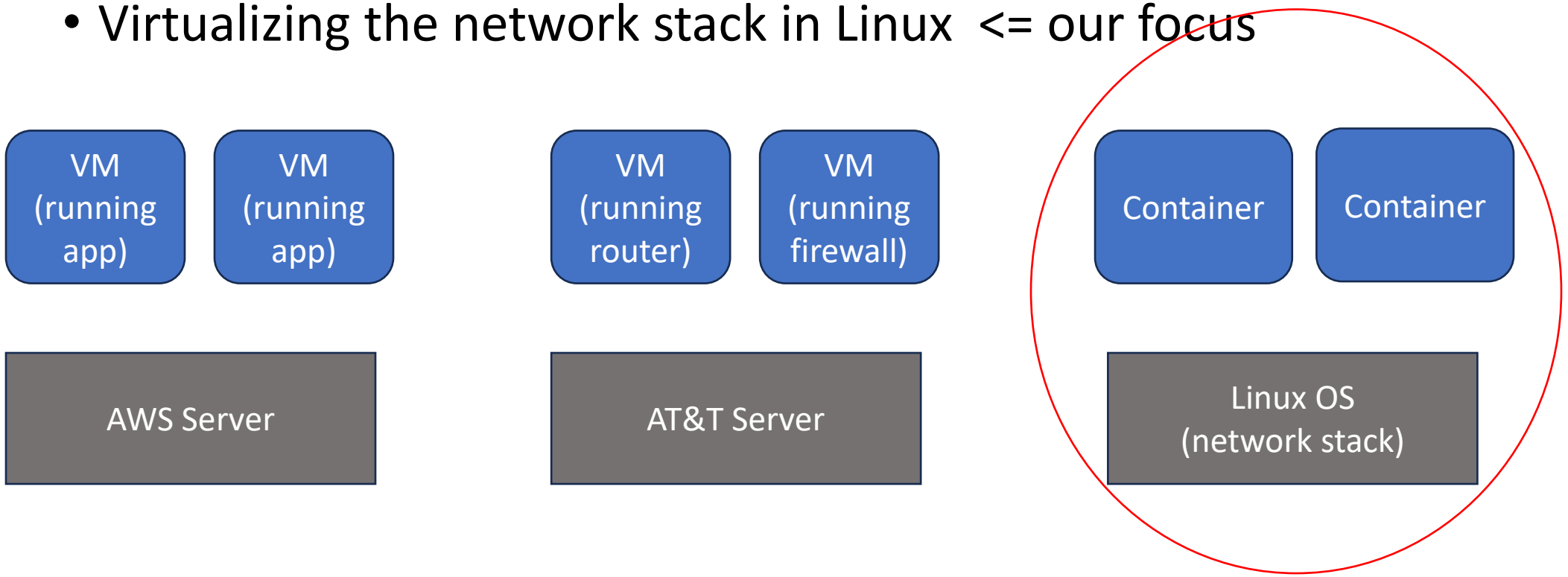
What is Virtualization?

- Technology for abstracting the underlying hardware / software
- Benefits:
  - More efficient use of resources
  - Simplified deployment by isolating configuration



Examples of Virtualization

- Cloud Computing
- Network Function Virtualization
- Virtualizing the network stack in Linux <= our focus



Outline

- Namespaces
- ip netns
- Networking Between Namespaces
- Docker Networking

University of Colorado **Boulder**

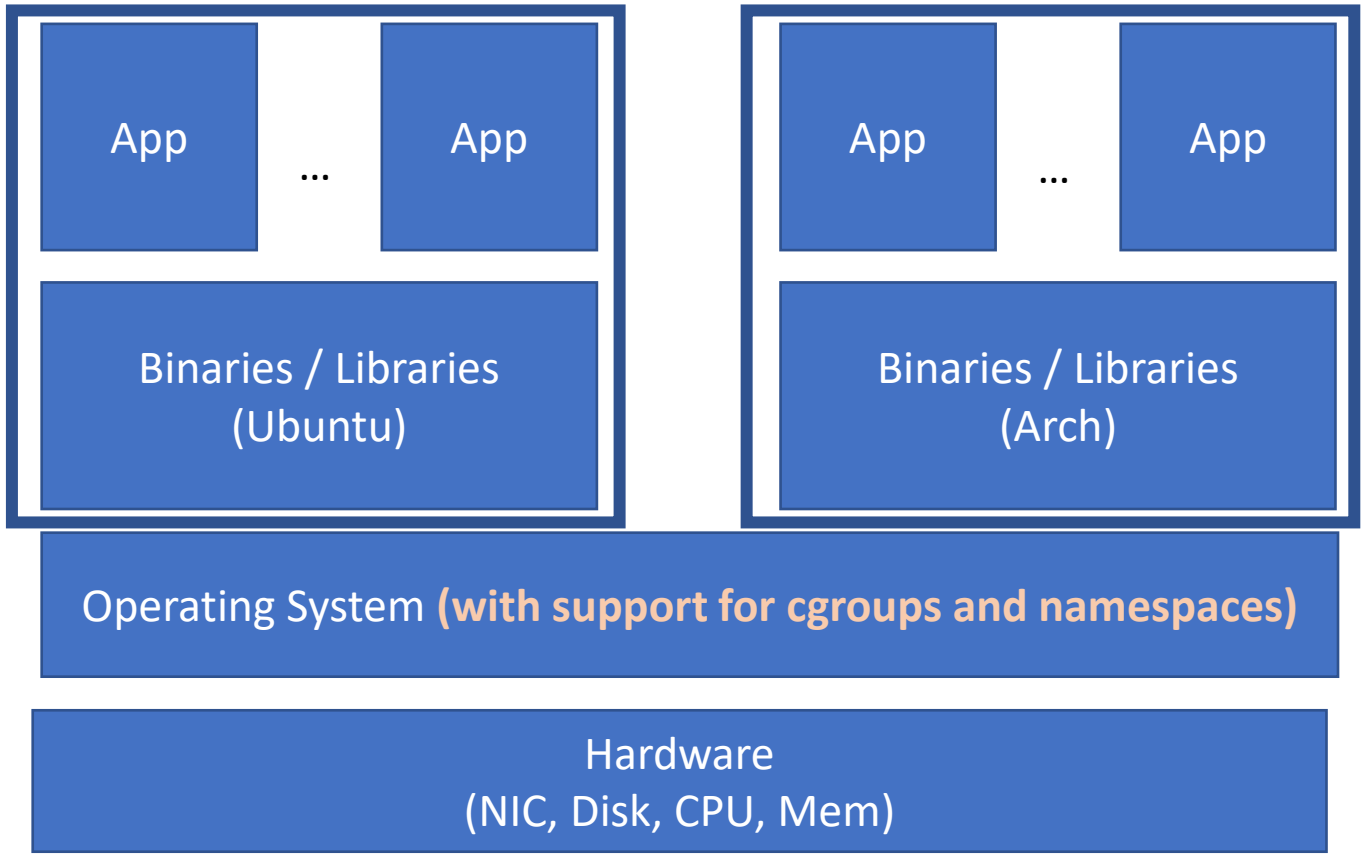
Network Namespaces

Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux

University of Colorado **Boulder**

Bare Metal -> Virtual Machines -> Containers

What if we don't need different OSes? Soln: Introduce isolation mechanisms into OS

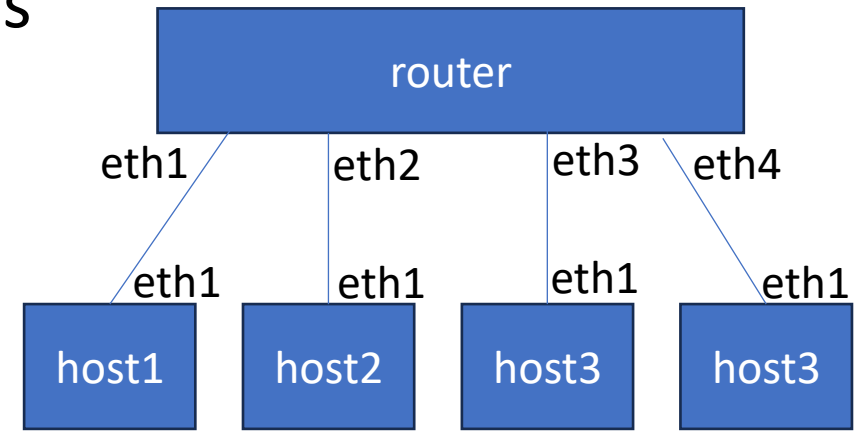


namespace - what resources and naming of those resources a process sees (file descriptors, IP addresses)

cgroup - (control group) groups processes and allocates resources (CPU, Memory) that the kernel enforces

Example Similar to Labs

- Containerlab Configuration file
- containerlab deploy – created docker containers
- docker exec commands inside of container



docker exec -it clab-demo-host1 ip link set dev eth1 address 22:33:22:44:55:44

docker exec -it clab-demo-router ip route add 10.0.2.0/24 dev eth2

namespaces

Creates isolation in the kernel that allows processes to have their own namespace for these resources.

Linux namespaces

Kernel maintains data structures on a per-process basis
(file system, process IDs, etc.)

```
742
743  struct task_struct {
744  #ifdef CONFIG_THREAD_INFO_IN_TASK
745          /*
746           * For reasons of header soup (see current_thread_info()), this
747           * must be the first element of task_struct.
748           */
749          struct thread_info              thread_info;
750  #endif
751          unsigned int                    __state;
752
1099
1100         /* Namespaces: */
1101         struct nsproxy                  *nsproxy;
1102
```

```
15  /*
16   * A structure to contain pointers to all per-process
17   * namespaces - fs (mount), uts, network, sysvipc, etc.
18   *
19   * The pid namespace is an exception -- it's accessed using
20   * task_active_pid_ns.  The pid namespace here is the
21   * namespace that children will use.
22   *
23   * 'count' is the number of tasks holding a reference.
24   * The count for each namespace, then, will be the number
25   * of nsproxies pointing to it, not the number of tasks.
26   *
27   * The nsproxy is shared by tasks which share all namespaces.
28   * As soon as a single namespace is cloned or unshared, the
29   * nsproxy is copied.
30   */
31  struct nsproxy {
32          refcount_t count;
33          struct uts_namespace *uts_ns;
34          struct ipc_namespace *ipc_ns;
35          struct mnt_namespace *mnt_ns;
36          struct pid_namespace *pid_ns_for_children;
37          struct net           *net_ns;
38          struct time_namespace *time_ns;
39          struct time_namespace *time_ns_for_children;
40          struct cgroup_namespace *cgroup_ns;
41  };
42  extern struct nsproxy init_nsproxy;
43
```

https://elixir.bootlin.com/linux/v6.6.7/source/include/linux/sched.h#L743
https://elixir.bootlin.com/linux/v6.6.7/source/include/linux/nsproxy.h#L31

Data Structures in the Linux Kernel

```
struct net {
…
   struct netns_ipv4 ipv4;
…
    struct netns_nf nf;
```

The relevant one to us is the struct net, which contains all of the ipv4 forwarding data structures, the netfilter tables, etc.

```
struct netns_ipv4 {

#ifdef CONFIG_IP_MULTIPLE_TABLES
    struct fib_rules_ops *rules_ops;
    struct fib_table __rcu *fib_main;
    struct fib_table __rcu *fib_default;
    unsigned int fib_rules_require_fldissect;
    bool fib_has_custom_rules;
#endif
    bool fib_has_custom_local_routes;
    bool fib_offload_disabled;

    struct hlist_head *fib_table_hash;
    struct sock *fibnl;
```

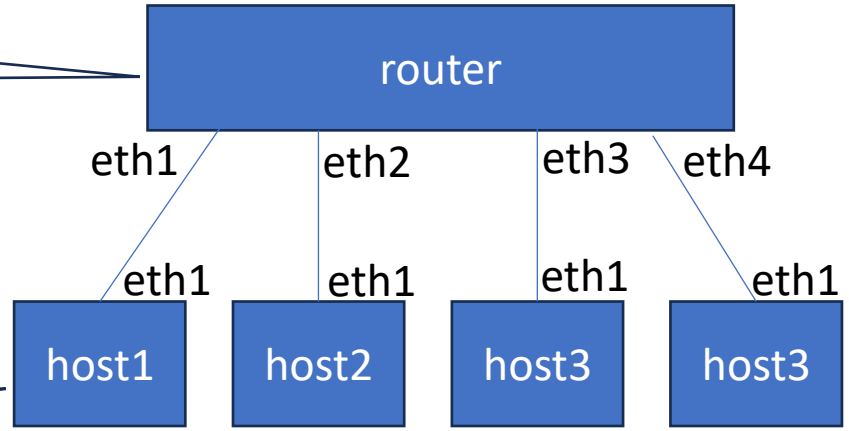
https://elixir.bootlin.com/linux/v6.6.7/source/include/net/net_namespace.h#L61

https://elixir.bootlin.com/linux/v6.6.7/source/include/net/netns/ipv4.h#L44

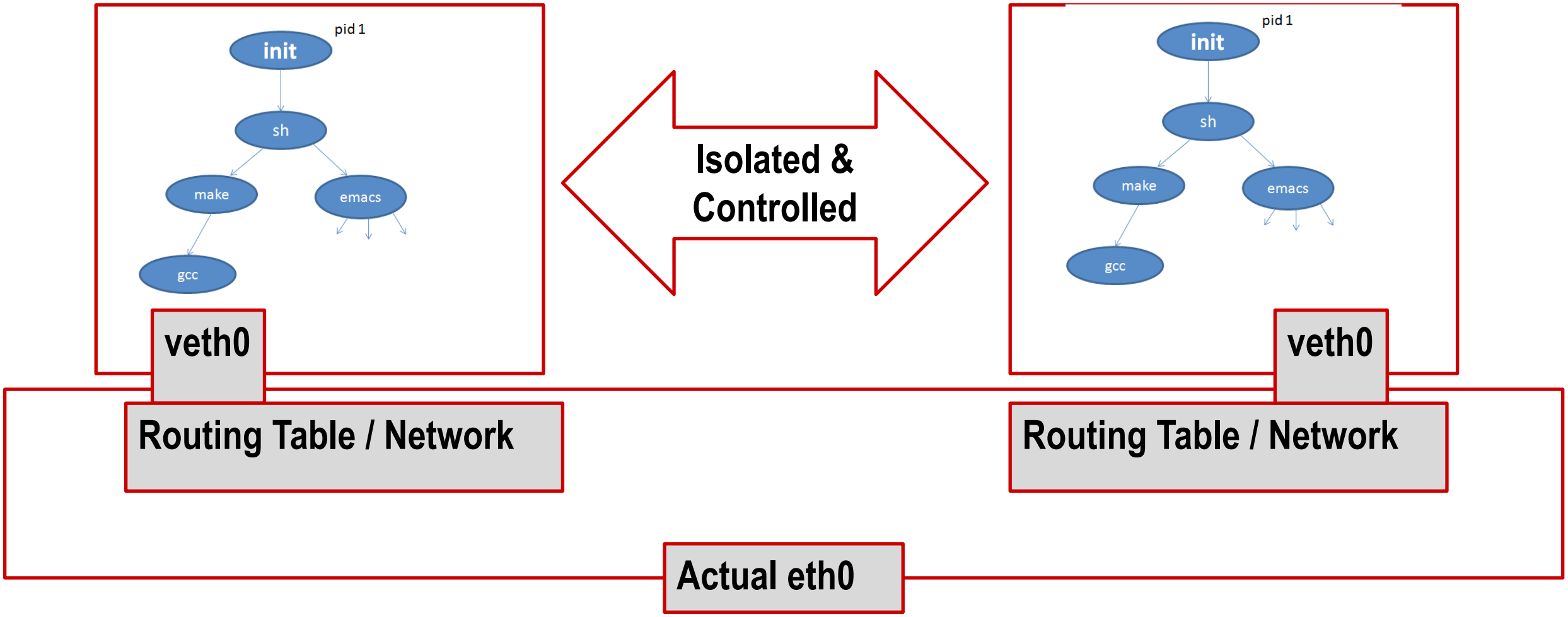
End Result:
Each namespace Gets its own set of tables

prefix	Next
123.0.1.0/16	10.0.0.1 dev eth3
…	…

prefix	Next
123.0.1.0/16	11.1.1.1 dev eth1
…	…



Virtualization using Namespaces



University of Colorado **Boulder**

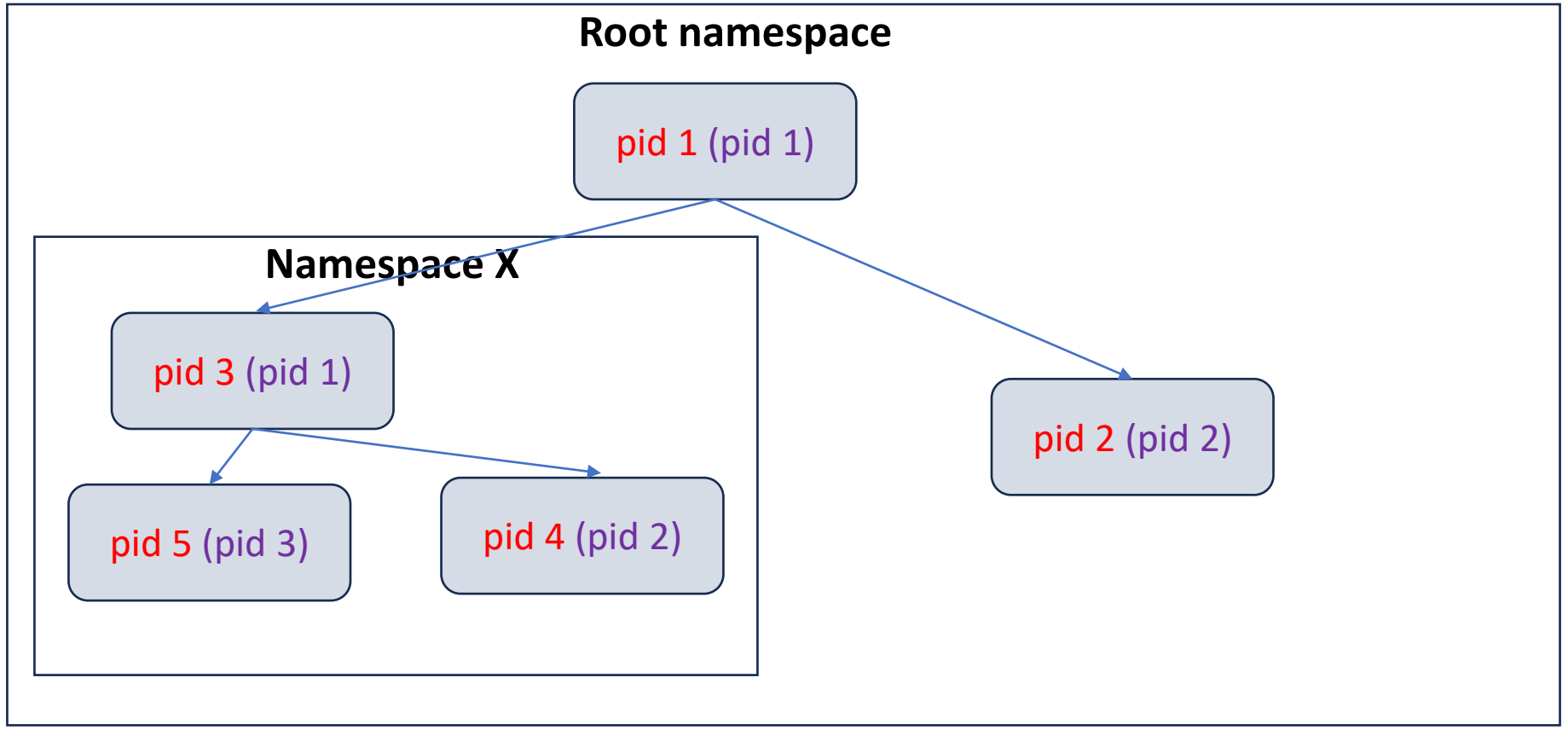
ip netns

Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux

University of Colorado **Boulder**

Process Relationship to Namespaces

- Processes Inherit from parent (starting with root/default namespace)



https://blog.devgenius.io/docking-a-docker-container-part-2-c1206e7c6677

ip netns

IP-NETNS(8) Linux IP-NETNS(8)

NAME top

ip-netns - process network namespace management

SYNOPSIS top

ip [*OPTIONS*] **netns** { *COMMAND* | **help** }

ip netns [**list**]

ip netns add *NETNSNAME*

ip netns attach *NETNSNAME PID*

ip [**-all**] **netns del** [*NETNSNAME*]

ip netns set *NETNSNAME NETNSID*

NETNSID := *auto* | *POSITIVE-INT*

...

https://man7.org/linux/man-pages/man8/ip-netns.8.html

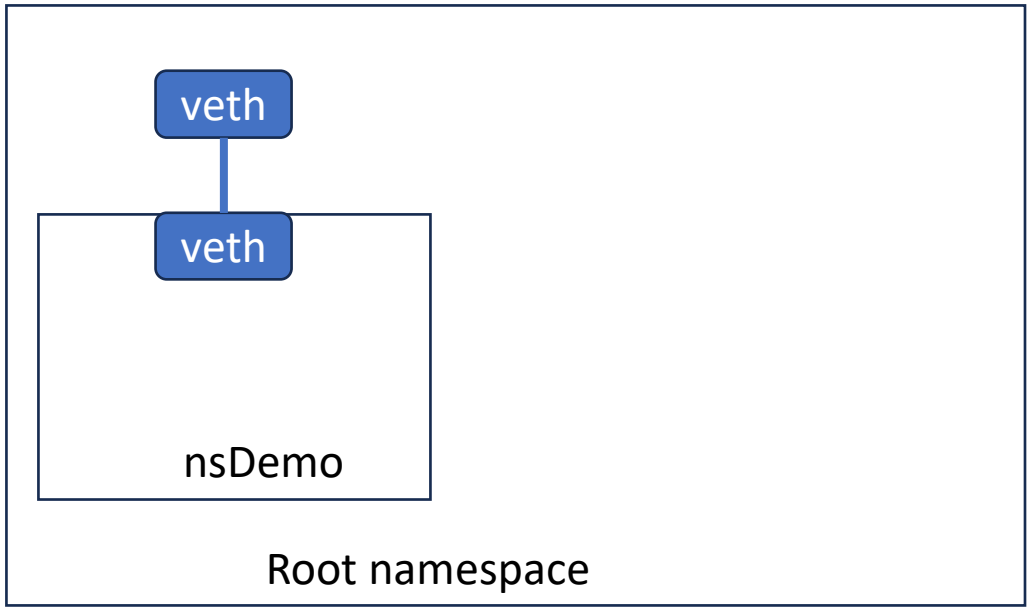
General Process

Setup:

- Create network namespace
- Create veth pair
- Attach veth devices to a namespace

Then you can:

- Execute commands inside namespace
- Use Linux networking
- …



Setup: Create Network Namespace

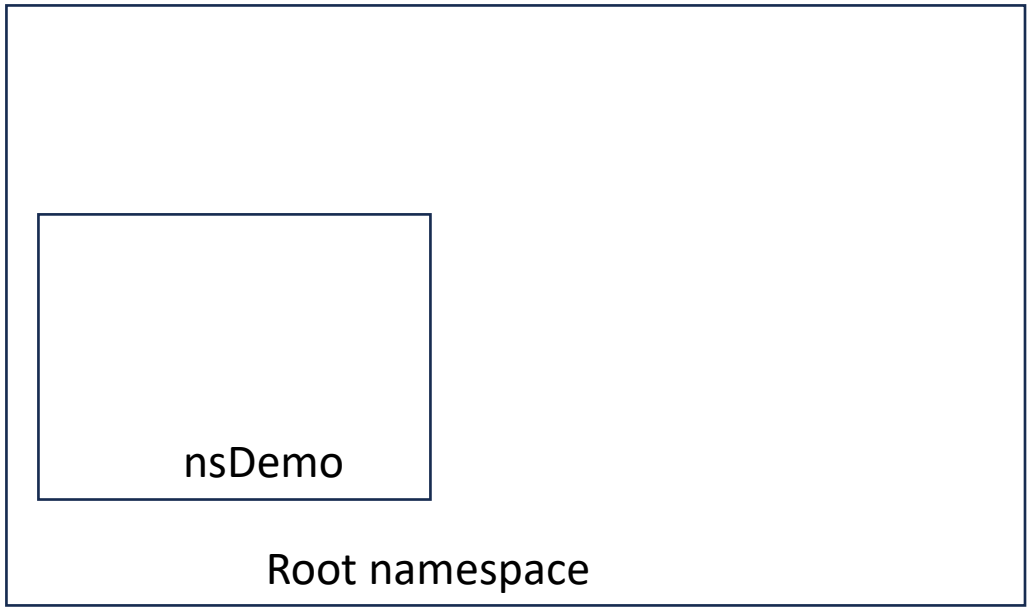
ip netns add NAME

- Create a new named network namespace

Example:

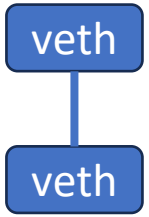
ip netns add nsDemo

ip netns list



veth devices

- veth - Virtual Ethernet device
- Always created in interconnected pairs
- Packets transmitted on one device in the pair are immediately received on the other device.



https://man7.org/linux/man-pages/man4/veth.4.html

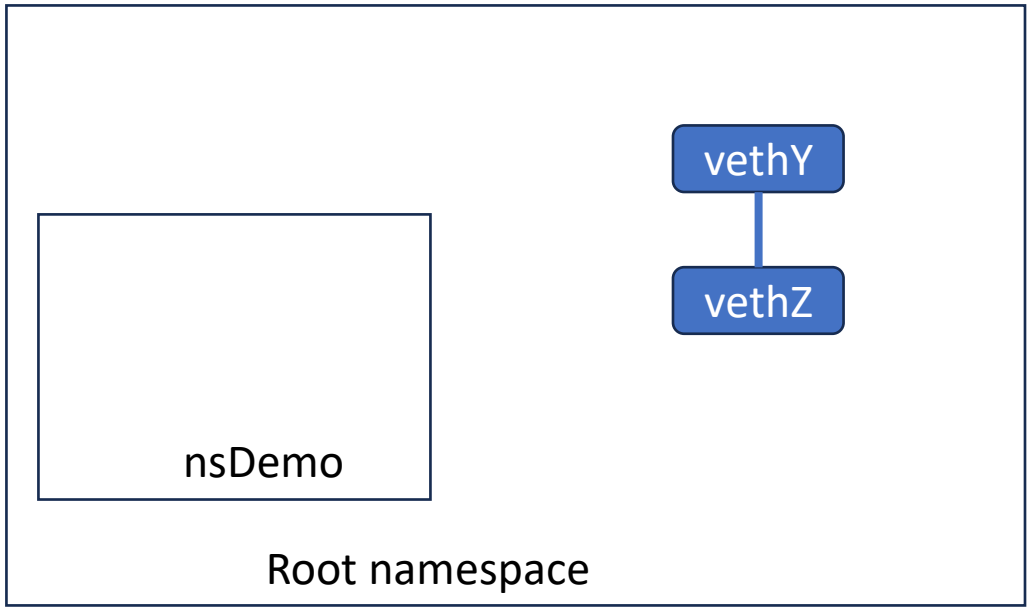
Setup: Create veth pair

ip link add <p1-name> type veth peer name <p2-name>

Example:
ip link add vethY type veth peer name vethZ

ip link # see both vethY and vethZ

ip link set dev vethY up
ip link set dev vethZ up



Finding the Peer

ethtool(8) can be used to find the peer of a **veth** network interface, using commands something like:

> ethtool -S vethY # Discover interface index of peer

NIC statistics:

peer_ifindex: 5

….

> ip link | grep '^5:' # Look up interface

5: vethZ@vethY: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc …

https://man7.org/linux/man-pages/man4/veth.4.html

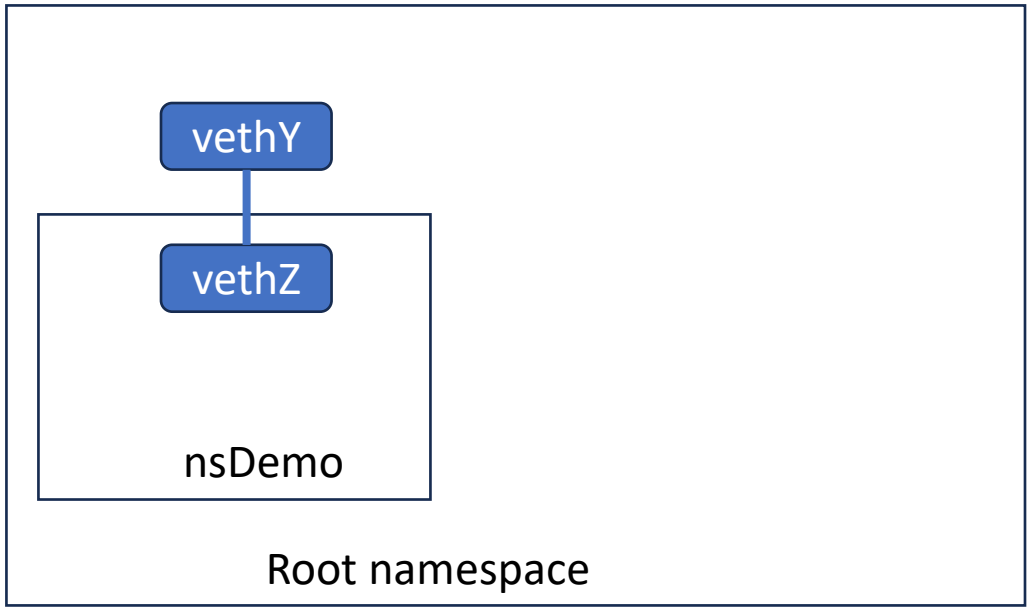
Setup: Attach to Namespace

ip link set <p2-name> netns <p2-ns>

Example:

ip link set vethZ netns nsDemo

ip link # note vethZ no longer shown



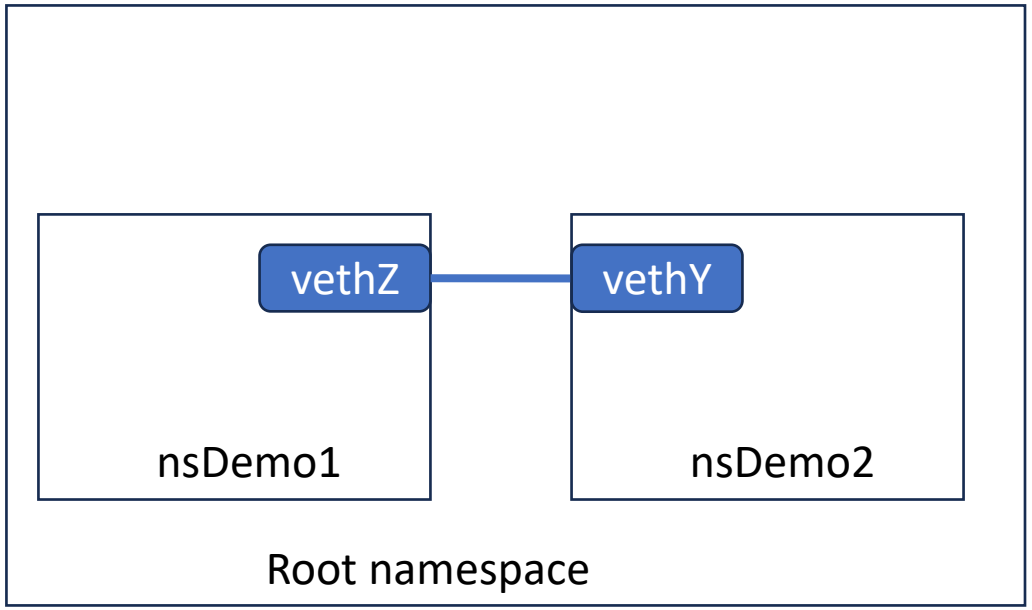
Can connect two namespaces

- Simply attach the other end of a veth pair to another namespace

Example:
ip netns add nsDemo1
ip netns add nsDemo2
ip link add vethY type veth peer name vethZ

ip link set vethZ netns nsDemo1

ip link set vethY netns nsDemo2



After: Execute Commands in netns

- ip netns exec <netns> <command>

Example – set an address on each veth device, and ping between them:
ip netns exec nsDemo ip link
ip netns exec nsDemo ip addr add 10.10.10.10/24 dev vethZ
ip netns exec nsDemo ip link set vethZ up

ip addr add 10.10.10.11/24 dev vethY
ip netns exec nsDemo ping 10.10.10.11
ip netns exec nsDemo tcpdump -i vethZ

Other misc.

- Moving veth device out of a namespace

  ip netns exec nsDemo ip link set vethZ netns 1 # 1 is default ns
- Note a –n option in ip commands (as alternative to ip netns exec):

  ip -n <netns> <iproute2 command>

  ip -n nsDemo route add 11.11.0.0/16 dev vethZ

  ip -n nsDemo route

After: Linux Networking

- If you do “ip link” (in root namespace) you’ll see the veth devices that are in the root namespace.
- veth devices are just Ethernet devices.
- So, can do Linux networking stuff with them.

- We’ll explore more in the next lesson

University of Colorado **Boulder**

Networking Between Namespaces

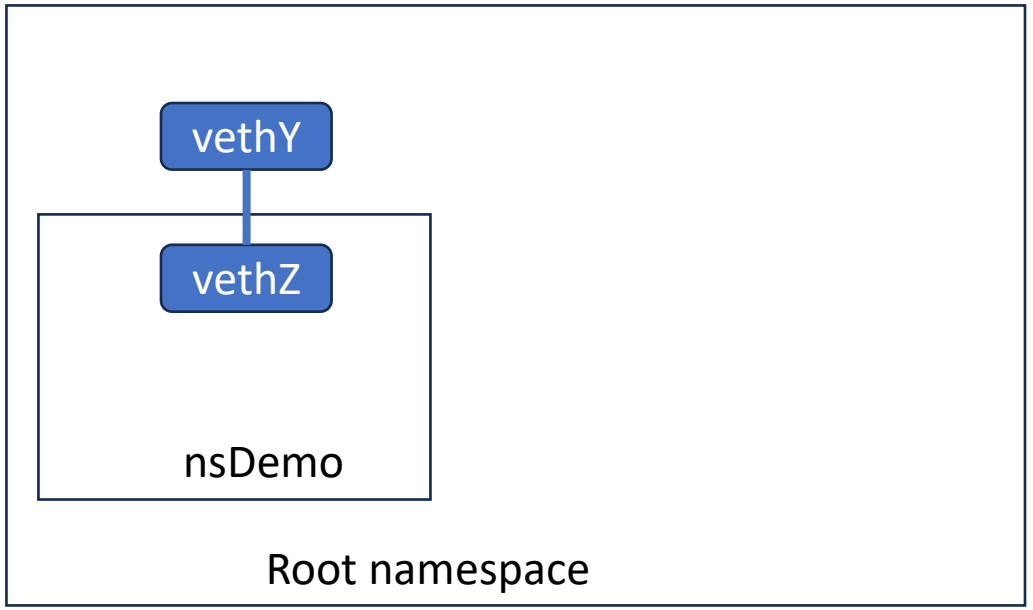
Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux

University of Colorado **Boulder**

Recap – Network Namespaces

- Create network namespaces
  ip netns add nsDemo
- Create veth pair
  ip link add vethY type veth peer name vethZ
- Attach veth to network namespace
  ip link set vethZ netns nsDemo

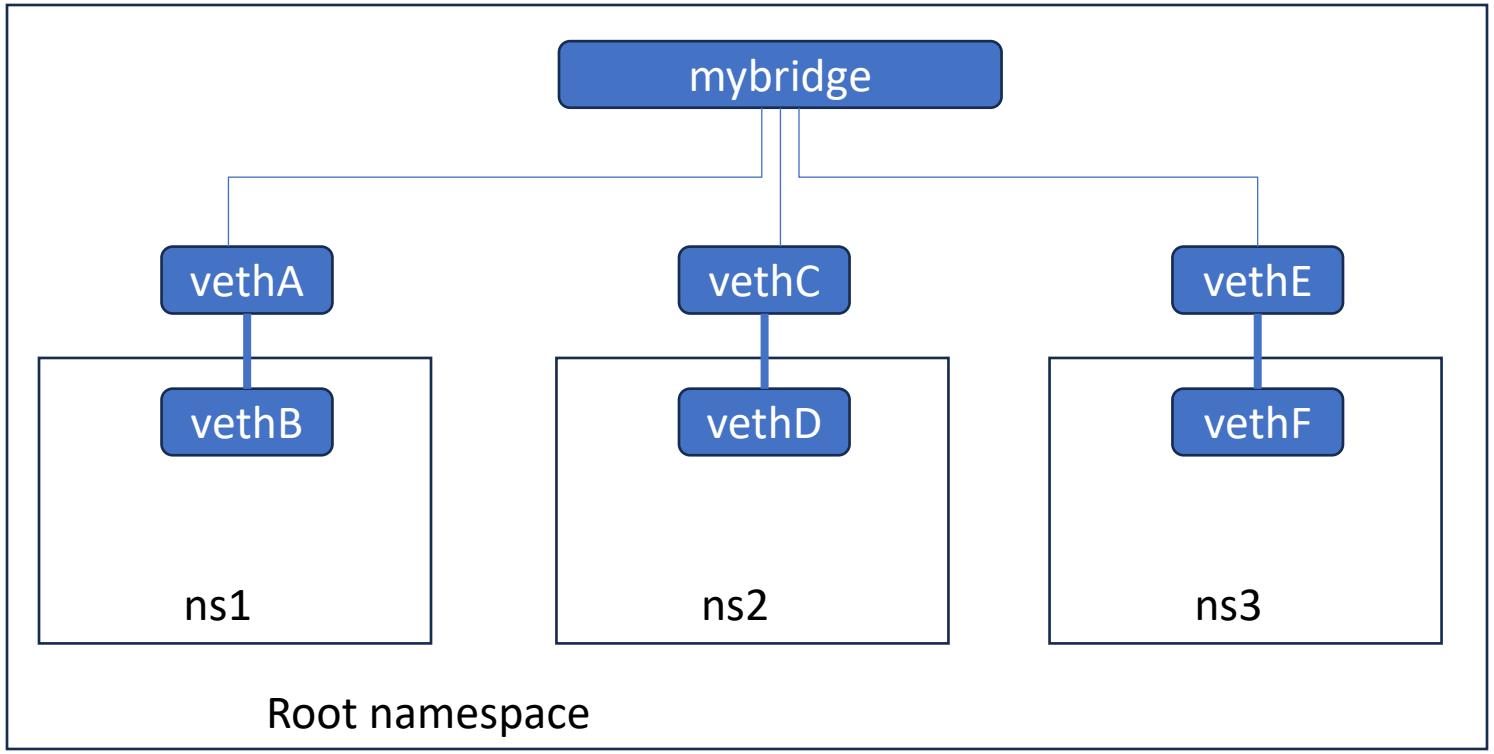
- Note: you'll also want to set the device state up, set an address and routes, etc.



Lab Setup

- Show vagrant file
- vagrant up
  - Brings up both VMs
- vagrant up node1
  - Brings up only node1
- Github link: https://github.com/eric-keller/npp-linux-04-virtual

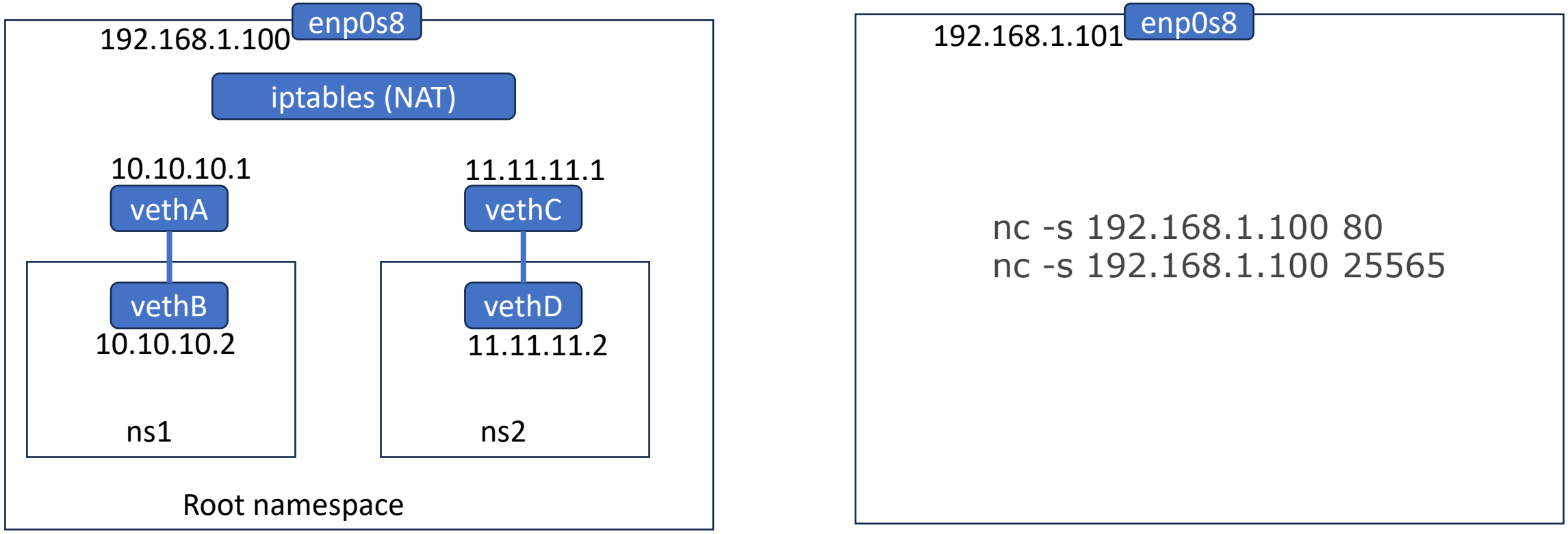
Case 1: Connecting Together w/ Bridge



Switch to console

- Link to github
  https://github.com/eric-keller/npp-linux-04-virtual

Case 2: Connecting To External World (e.g., port forwarding)



node1

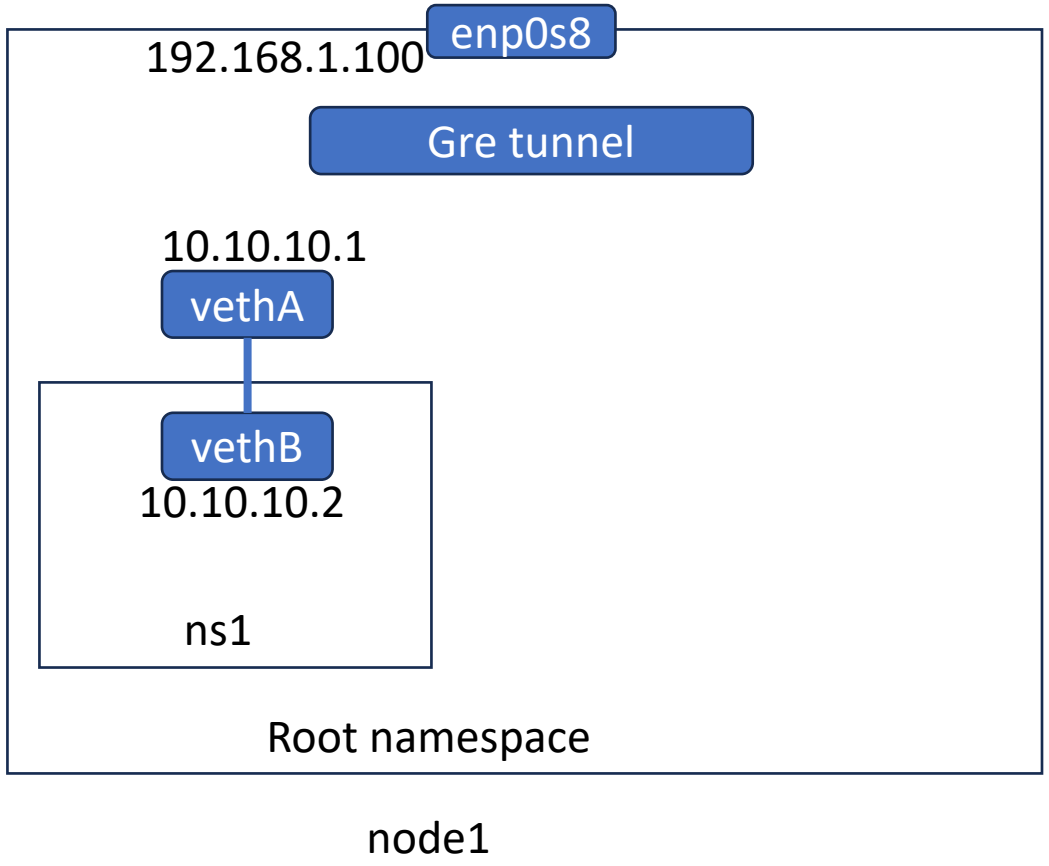
node2

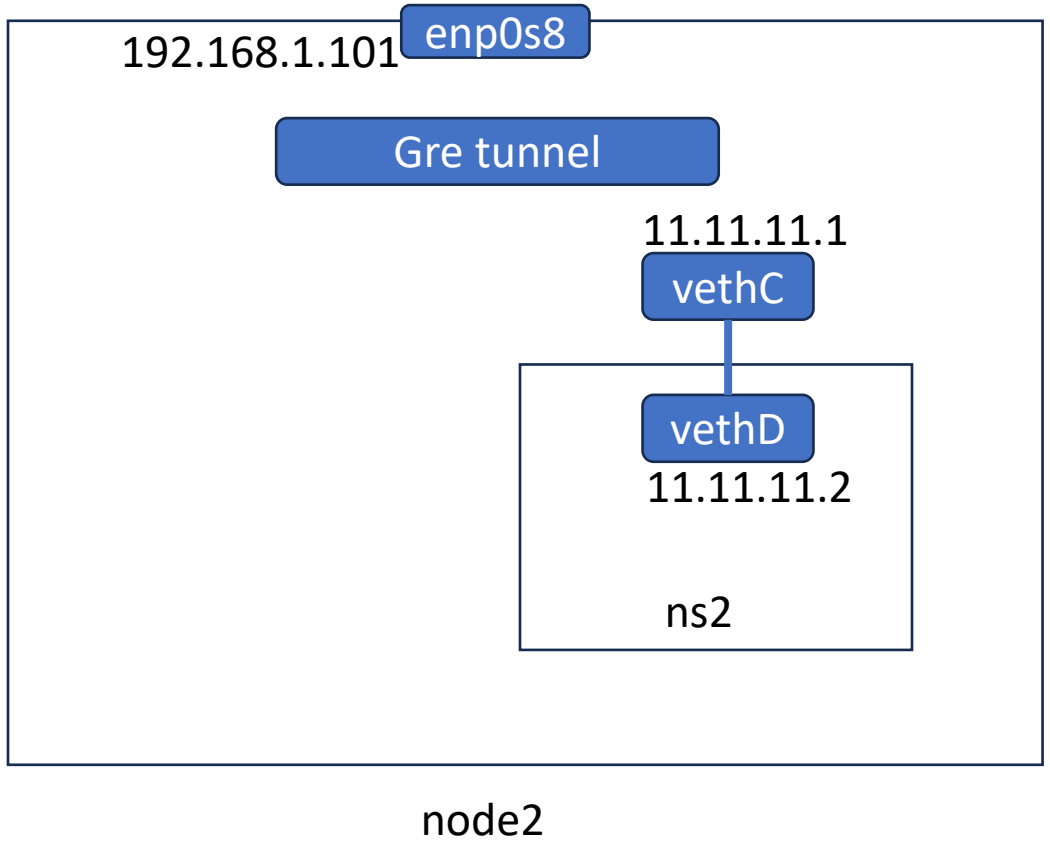
In ns1 – run service on port 80
In ns2 – run service on port 25565

Switch to console

- Link to github
  https://github.com/eric-keller/npp-linux-04-virtual

Case 3: Extending network between machines (try on your own)





Try on your own

Lab Overview

- You are provided with a create and delete script that have been obfuscated. (github link)
- Only need one node:
  - vagrant up node1
  - clear-fw.sh
  - ob-create-lab1-mod4.sh
  - Answer questions in Coursera
  - ob-del-lab1-mod4.sh
- Useful commands:
  - ip netns list, ip netns exec, ip route, ethtool, ip link, ping, tshark or tcpdump

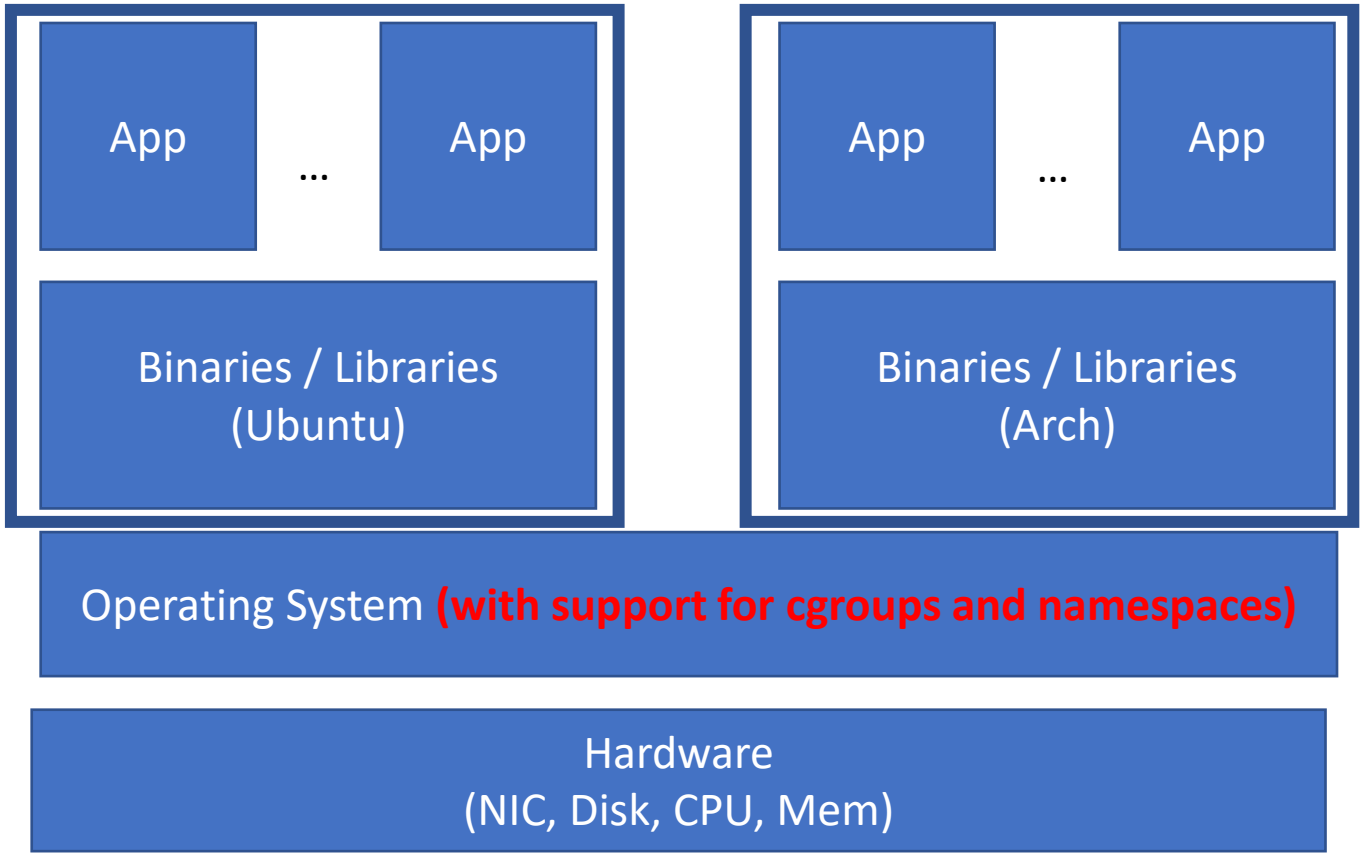
University of Colorado **Boulder**

Docker Networking

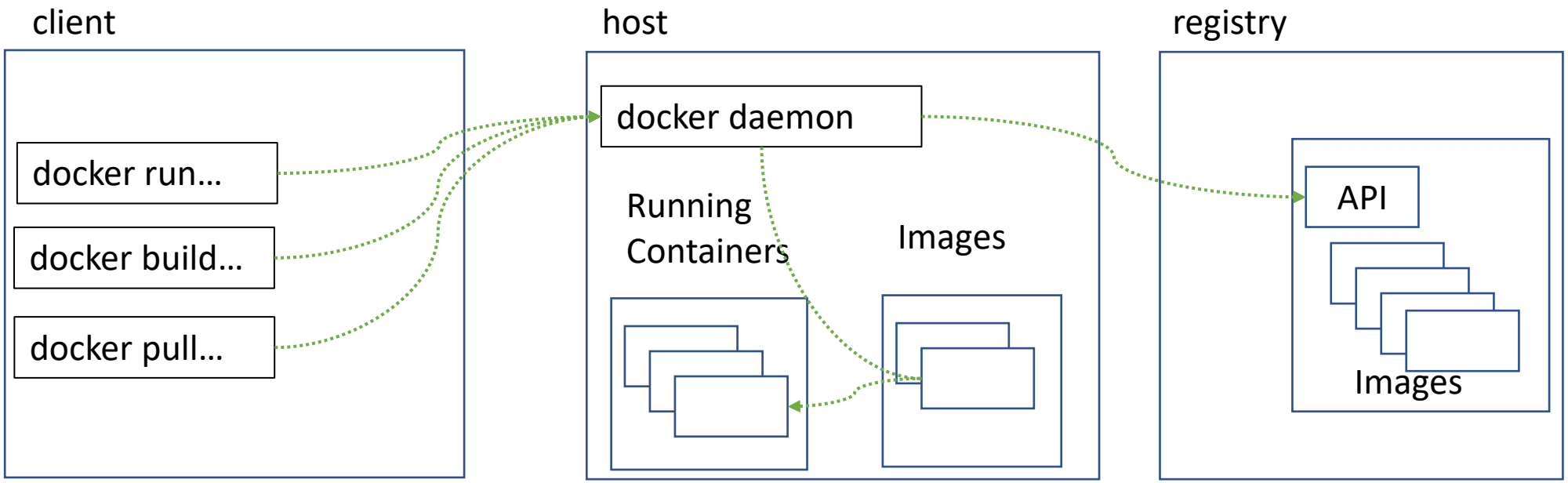
Course: Networking Principles in Practice – Linux Networking
Module: Virtual Networking in Linux

University of Colorado **Boulder**

Recap: Container



Recap: Docker




```
sudo docker run -ti --rm ubuntu:22.04 /bin/bash
```

Recap: Docker Containers

- A temporary filesystem
  - layered over an image
  - fully writable (copy on write)
  - disappears when End of Life
- A Network Stack
- A Process Group - one main process, with possible subprocesses (which exits when main process exits)

What Networking is Set Up?

- sudo ip netns
- sudo ip link
- sudo docker run -d -rm --name nginx1 nginx
- sudo ip link
- sudo ip netns

- Nothing is listed?

Docker / ip netns?

“ *ip netns ls* command looks up network namespaces file in the */var/run/netns* directory.

However, the **Docker daemon doesn’t create a reference of the network namespace file in the */var/run/netns* directory after the creation.** Therefore, *ip netns ls* cannot resolve the network namespace file.”

https://www.baeldung.com/linux/docker-network-namespace-invisible

Soln – see run-container.sh (and del-container.sh)

Now, let’s re-run ip link

- See the veth in the root namespace
  - sudo ip link
- Get the veth this is paired to (gives ID)
  - ethtool –S <veth>
- Look inside of container
  - sudo ip netns exec <containername> ip link
- There it is

Where does the veth traffic go (in root)?

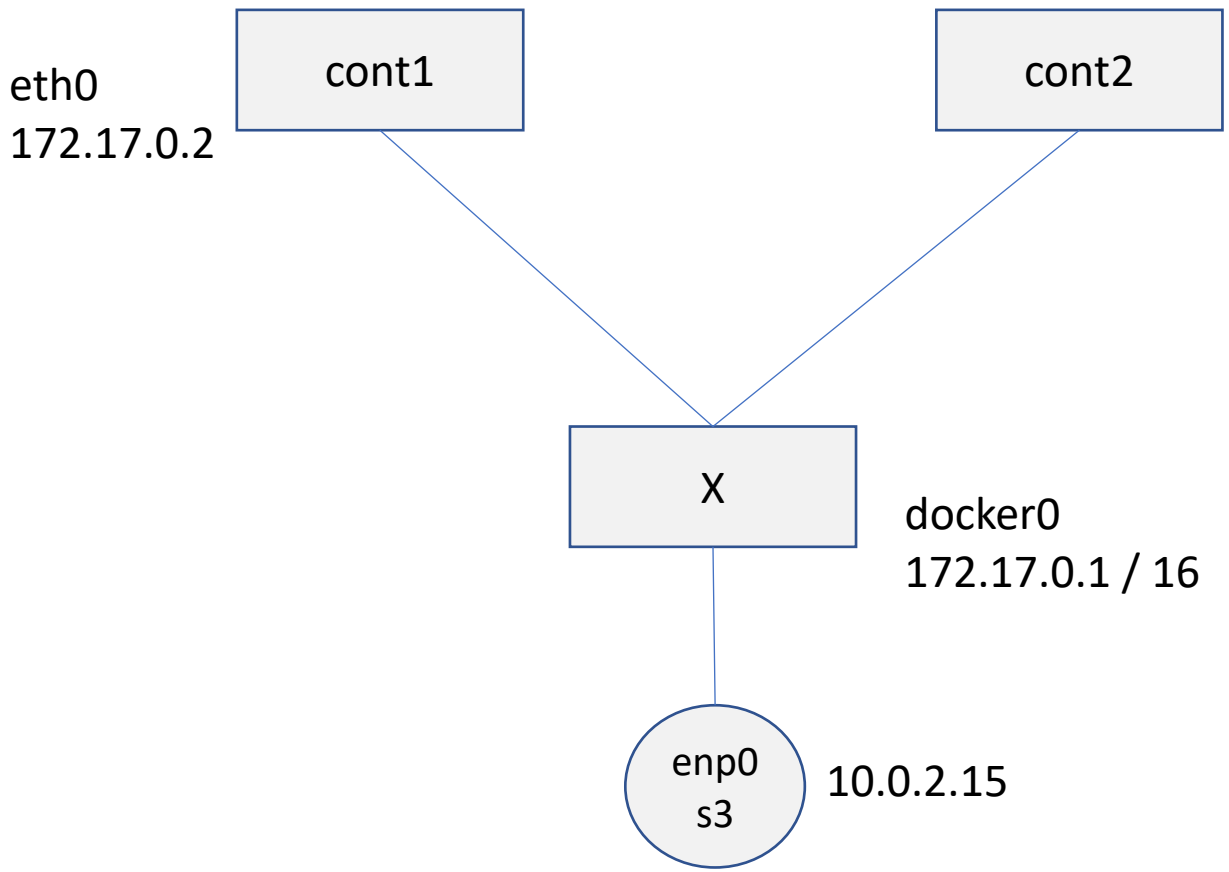
ip link

Sample output:

4: **docker0**: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default link/ether 02:42:ee:da:17:05 brd ff:ff:ff:ff:ff:ff

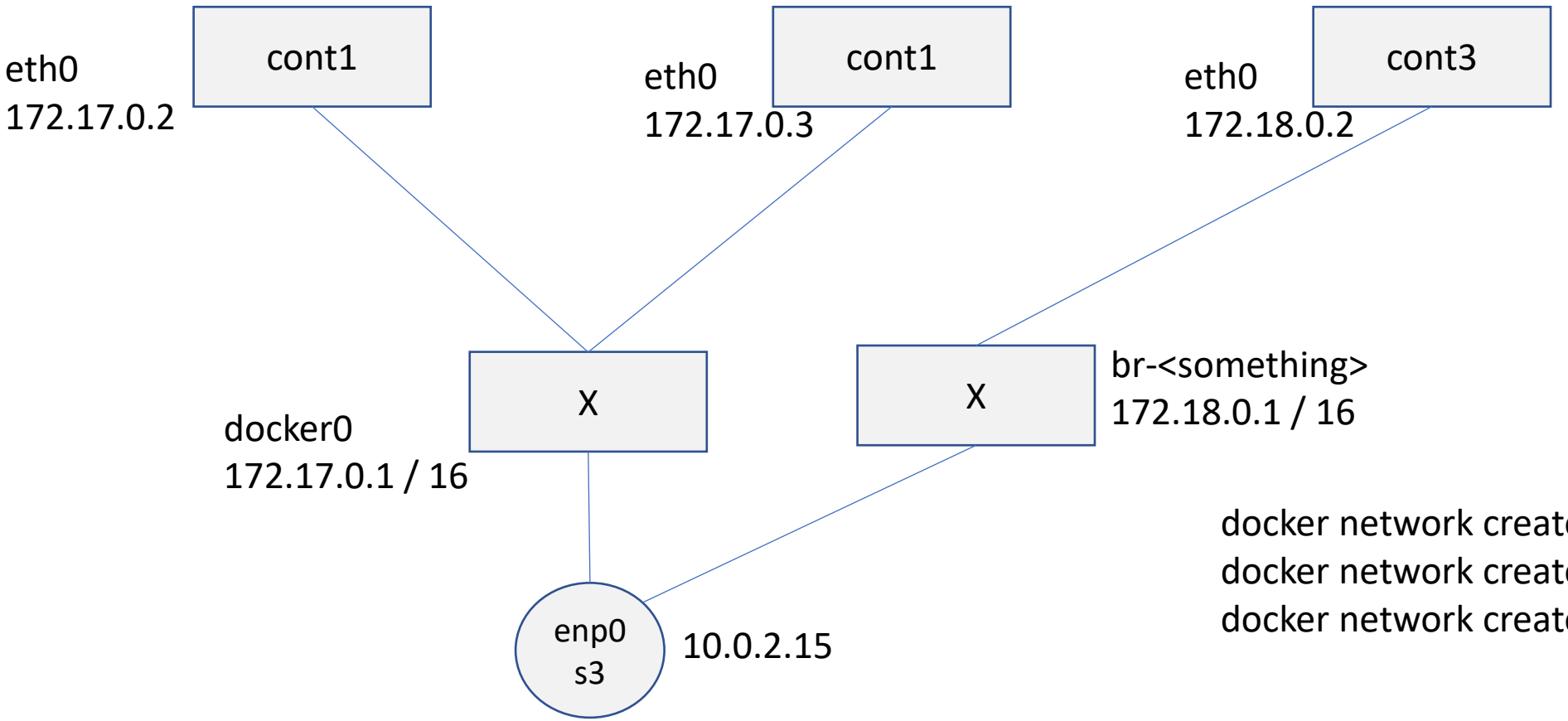
84: veth34e2745@if83: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue **master docker0** state UP mode DEFAULT group default link/ether 92:53:22:f2:24:6f brd ff:ff:ff:ff:ff:ff link-netnsid 2

Docker Networking: docker0 bridge



docker0 is a default network bridge that all containers attach to (unless otherwise specified)

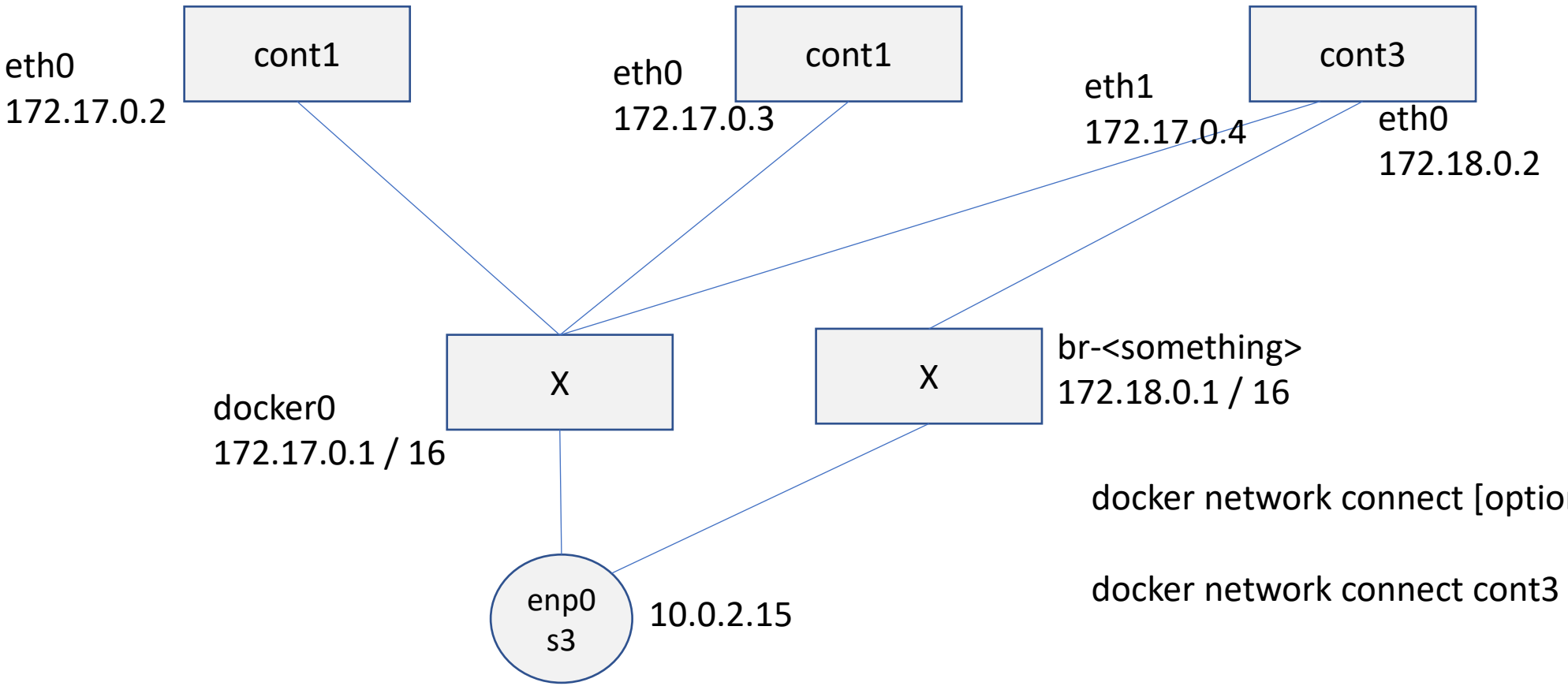
Custom Networks in Docker



docker network create --help
docker network create mynet1 (default to 172.18…)
docker network create --subnet=11.11.11.0/24 mynet

docker run --net=mynet1 --name cont3 -dit myimg
docker exec cont3 ifconfig

Attaching to Multiple Networks

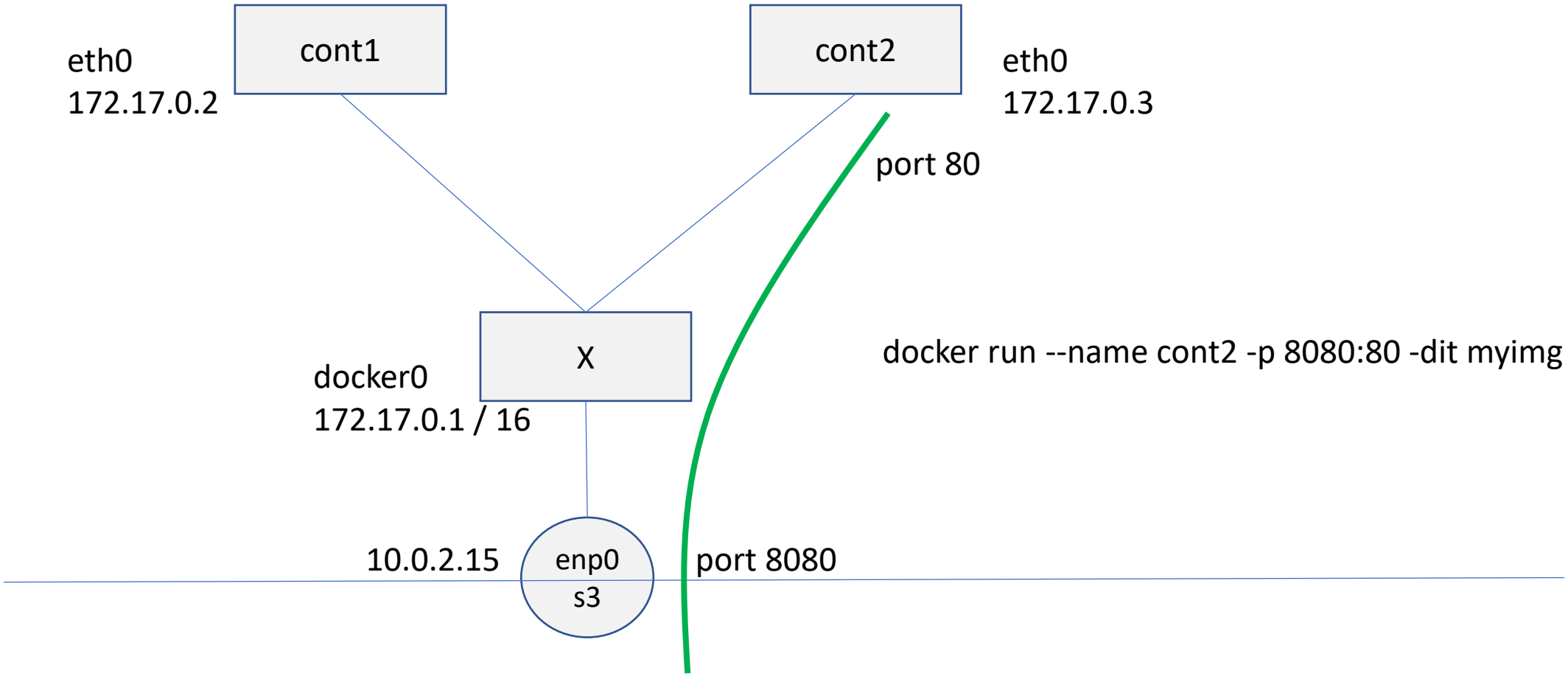


docker network connect [options] <network> <container>

docker network connect cont3 docker0

Port Forwarding

docker run -p <host_port>:<container_port> <image>



Docker compose wrapper

What is it:

- YAML wrapper for docker commands (build, pull, run…)
- Also supports multiple containers

To install: https://docs.docker.com/compose/install/

To use: https://docs.docker.com/compose/reference/

docker-compose --help (build, run, … - a lot of the same commands)

Example

docker-compose.yaml

```
services:
  proxy:
    build: ./proxy
    networks:
      - frontend
  app:
    build: ./app
    networks:
      - frontend
      - backend
  db:
    image: postgres
    networks:
      - backend
```

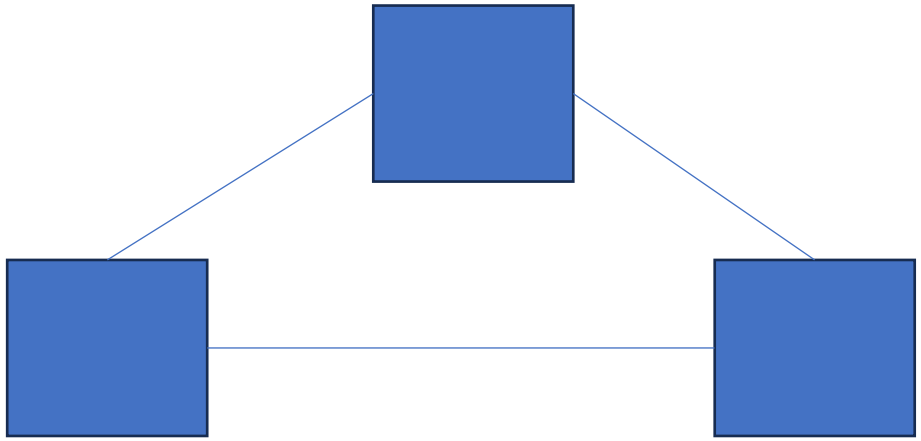
docker compose up

```
networks:
  frontend:
    # Use a custom driver
    driver: custom-driver-1
  backend:
    # Use a custom driver which takes special options
    driver: custom-driver-2
    driver_opts:
      foo: "1"
      bar: "2"
```

https://docs.docker.com/compose/networking/

Practice Exercise

- Create a containerlab topology configuration for the following topology
- Inspect the network that was created
- Attempt to recreate it manually



University of Colorado **Boulder**