Topic-2 LAB (Network namespace)

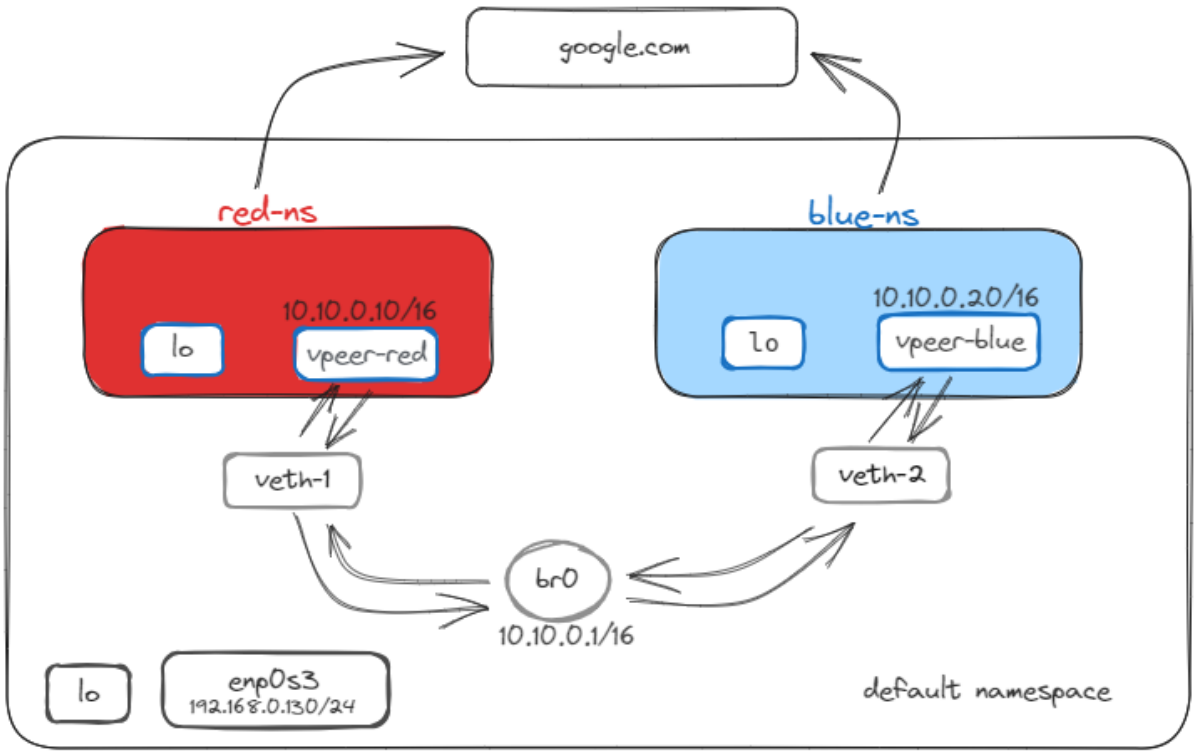
- Linux network namespaces are a Linux kernel feature allowing us to isolate network environments through virtualization.
- A network namespace is logically another copy of the network stack, with its own routes,firewall rules, and network devices.
- By convention a named network namespace is an object at /var/run/netns/NAME that can be opened.
- Network namespaces provide isolation of the system resources associated with networking: network devices, IPv4 and IPv6 protocol stacks, IP routing tables, firewall rules, the /proc/net directory (which is a symbolic link to /proc/PID/net), the /sys/class/net directory, various files under /proc/sys/net, port numbers (sockets), and so on. In addition, network namespaces isolate the UNIX domain abstract socket namespace.
- A virtual network (veth) device pair provides a pipe-like abstraction that can be used to create tunnels between network namespaces, and can be used to create a bridge to a physical network device in another namespace. When a namespace is freed, the veth devices that it contains are destroyed.

Note:

- Network namespaces are **not persistent across system restarts**. You will need to create a script that is run at startup and arrange to have it run.
- How you cause it to run depends on your needs, you may be able to cal it from **/etc/rc.local** or you may be able to hook it into some other script.

LAB:

- Create two network namespace and ping each other. Also check connectivity to the internet from each namespace.



Step: 1 Check current Status

1.1 Display all existing namespaces (default ns)

```
$ lsns
```

```
        NS TYPE   NPROCS   PID USER    COMMAND
4026531834 time         4   893 subarno /lib/systemd/systemd --user
4026531835 cgroup       4   893 subarno /lib/systemd/systemd --user
4026531836 pid          4   893 subarno /lib/systemd/systemd --user
4026531837 user         4   893 subarno /lib/systemd/systemd --user
4026531838 uts          4   893 subarno /lib/systemd/systemd --user
4026531839 ipc          4   893 subarno /lib/systemd/systemd --user
4026531840 net          4   893 subarno /lib/systemd/systemd --user
4026531841 mnt          4   893 subarno /lib/systemd/systemd --user
```

```
$ip a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP gr
oup default qlen 1000
    link/ether 08:00:27:0e:d4:68 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.130/24 brd 192.168.0.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe0e:d468/64 scope link
       valid_lft forever preferred_lft forever
```

1.2 Display network namespaces

```
$ ip netns
```

***nothing

Step:2 Creating a Network Namespace(red-ns and blue-ns)

```
$ sudo ip netns add red-ns
$ sudo ip netns add blue-ns
$ ip netns
blue-ns
red-ns
```

Step:3 Connecting the cables (create veth link)

3.1 Create a connection with host network and namespace

We will connect these two namespaces with a virtual ethernet cable (vETH cable) and ethernet devices are created in pairs.

```
$sudo ip link add veth-1 type veth peer name vpeer-red
$sudo ip link add veth-2 type veth peer name vpeer-blue
```

```
$ sudo ip a | tail -8
```

```
3: vpeer-red@veth-1: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 5e:19:3b:ec:d7:4c brd ff:ff:ff:ff:ff:ff
4: veth-1@vpeer-red: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether f2:f6:67:72:91:0f brd ff:ff:ff:ff:ff:ff
5: vpeer-blue@veth-2: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 8a:5e:08:88:85:2b brd ff:ff:ff:ff:ff:ff
6: veth-2@vpeer-blue: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 76:cd:ec:51:56:3a brd ff:ff:ff:ff:ff:ff
```

3.2 Bring up the newly created cable

vETH will work as a network cable where one end is attached to the host network, and the other end to the newly created network namespace.

```
$ sudo ip link set veth-1 up
$ sudo ip link set veth-2 up
$ sudo ip  a | tail -8
```

```
3: vpeer-red@veth-1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 5e:19:3b:ec:d7:4c brd ff:ff:ff:ff:ff:ff
4: veth-1@vpeer-red: <NO-CARRIER,BROADCAST,MULTICAST,UP,M-DOWN> mtu 1500 qdisc noqueue state LOWERLAYERDOWN gro
up default qlen 1000
    link/ether f2:f6:67:72:91:0f brd ff:ff:ff:ff:ff:ff
5: vpeer-blue@veth-2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 8a:5e:08:88:85:2b brd ff:ff:ff:ff:ff:ff
6: veth-2@vpeer-blue: <NO-CARRIER,BROADCAST,MULTICAST,UP,M-DOWN> mtu 1500 qdisc noqueue state LOWERLAYERDOWN gr
oup default qlen 1000
    link/ether 76:cd:ec:51:56:3a brd ff:ff:ff:ff:ff:ff
```

3.3 Add peers to ns

```
$ sudo ip link set vpeer-red netns red-ns
$ sudo ip link set vpeer-blue netns blue-ns
```

```
$ sudo ip  a | tail -4
```

```
4: veth-1@if3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state LOWERLAYERDOWN group
default qlen 1000
    link/ether f2:f6:67:72:91:0f brd ff:ff:ff:ff:ff:ff link-netns red-ns
6: veth-2@if5: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state LOWERLAYERDOWN group
default qlen 1000
    link/ether 76:cd:ec:51:56:3a brd ff:ff:ff:ff:ff:ff link-netns blue-ns
```

Note: 3-4 merged and 5-6 merged

Step: 4 Creating loopback interface and setup peer to namespace

4.1 Create loopback interface

```
$ ip netns
```

```
blue-ns (id: 1)
red-ns (id: 0)
```

```
$sudo ip netns exec red-ns ip link set lo up
$sudo ip netns exec blue-ns ip link set lo up
```

```
$ sudo ip  a | tail -4
```

```
4: veth-1@if3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state LOWERLAYERDOWN group
default qlen 1000
    link/ether f2:f6:67:72:91:0f brd ff:ff:ff:ff:ff:ff link-netns red-ns
6: veth-2@if5: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state LOWERLAYERDOWN group
default qlen 1000
    link/ether 76:cd:ec:51:56:3a brd ff:ff:ff:ff:ff:ff link-netns blue-ns
```

4.2 Setup peer ns interface

```
$ sudo ip netns exec red-ns ip link set vpeer-red up
$ sudo ip netns exec blue-ns ip link set vpeer-blue up
```

```
$ ip a | tail -8
```

```
4: veth-1@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default q
len 1000
    link/ether f2:f6:67:72:91:0f brd ff:ff:ff:ff:ff:ff link-netns red-ns
    inet6 fe80::f0f6:67ff:fe72:910f/64 scope link
       valid_lft forever preferred_lft forever
6: veth-2@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default q
len 1000
    link/ether 76:cd:ec:51:56:3a brd ff:ff:ff:ff:ff:ff link-netns blue-ns
    inet6 fe80::74cd:ecff:fe51:563a/64 scope link
       valid_lft forever preferred_lft forever
```

Step: 5 Assigning IP address to NS interfaces

In order to connect to the network, a node must have at least one network interface.

```
$ sudo ip netns exec red-ns ip addr add 10.10.0.10/16 dev vpeer-red
$ sudo ip netns exec blue-ns ip addr add 10.10.0.20/16 dev vpeer-blue
```

Note:

- Still now, we've only assigned network addresses to the interfaces inside the network namespaces (red-ns (vpeer-red), blue-ns (vpeerblue)).
- The host namespaces interfaces do not have an IP assigned (veth1, veth2).

Step: 6 Create a bridge(for L2 communication)

- Bridge name : br0
- Bridge addr : 10.10.0.1

6.1 Setup bridge

```
$ sudo ip link add br0 type bridge
```

```
$ ip a | tail -2
```

```
7: br0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 1e:5d:5d:7b:24:01 brd ff:ff:ff:ff:ff:ff
```

```
$sudo ip link set br0 up
```

```
$ ip a | tail -2
```

```
7: br0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
    link/ether 1e:5d:5d:7b:24:01 brd ff:ff:ff:ff:ff:ff
```

Note: Both side connection is not established so bridge is still **DOWN**

6.2 Assign veth pairs to bridge

```
$ sudo ip link set veth-1 master br0
$ sudo ip link set veth-2 master br0
```

```
$ ip a | tail -2
```

```
7: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 1e:5d:5d:7b:24:01 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::1c5d:5dff:fe7b:2401/64 scope link
       valid_lft forever preferred_lft forever
```

6.3 Setup bridge IP

```
$ sudo ip addr add 10.10.0.1/16 dev br0
```

```
$ ip a | tail -6
```

```
7: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 1e:5d:5d:7b:24:01 brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.1/16 scope global br0
       valid_lft forever preferred_lft forever
    inet6 fe80::1c5d:5dff:fe7b:2401/64 scope link
       valid_lft forever preferred_lft forever
```

6.4 Setup default route to direct traffic to the bridge

Since we have the vETH pairs connected to the bridge, the bridge network address is available to these network namespaces. Let's add a default route to direct the traffic to the bridge.

```
$ sudo ip netns exec red-ns  ip route add default via 10.10.0.1
$ sudo ip netns exec blue-ns  ip route add default via 10.10.0.1
```

Check ping

```
$ sudo ip netns exec red-ns ping -c4 10.10.0.20
```

```
64 bytes from 10.10.0.20: icmp_seq=1 ttl=64 time=0.266 ms
64 bytes from 10.10.0.20: icmp_seq=2 ttl=64 time=0.089 ms
64 bytes from 10.10.0.20: icmp_seq=3 ttl=64 time=0.089 ms
64 bytes from 10.10.0.20: icmp_seq=4 ttl=64 time=0.087 ms
```

```
$ sudo ip netns exec blue-ns ping -c4 10.10.0.10
```

```
64 bytes from 10.10.0.10: icmp_seq=1 ttl=64 time=0.093 ms
64 bytes from 10.10.0.10: icmp_seq=2 ttl=64 time=0.091 ms
64 bytes from 10.10.0.10: icmp_seq=3 ttl=64 time=0.093 ms
64 bytes from 10.10.0.10: icmp_seq=4 ttl=64 time=0.088 ms
```

Check ping from default namespace (main OS)

```
$ ping -c2 10.10.0.10
```

```
64 bytes from 10.10.0.10: icmp_seq=1 ttl=64 time=0.185 ms
64 bytes from 10.10.0.10: icmp_seq=2 ttl=64 time=0.082 ms
```

```
$ ping -c2 10.10.0.20
```

```
64 bytes from 10.10.0.20: icmp_seq=1 ttl=64 time=0.320 ms
64 bytes from 10.10.0.20: icmp_seq=2 ttl=64 time=0.084 ms
```

```
$ ping -c2 10.10.0.1
```

```
64 bytes from 10.10.0.1: icmp_seq=1 ttl=64 time=0.042 ms
64 bytes from 10.10.0.1: icmp_seq=2 ttl=64 time=0.072 ms
```

Step: 7 Allow traffic to go outside the Namespace (MASQUERADE)

We are able to send traffic between the namespaces, but we did not allow sending traffic outside the container. So, we need to masquerade the outgoing traffic from our namespace.

MASQUERADE modifies the source address of the packet, replacing it with the address of a specified network interface. This is similar to SNAT, except that it does not require the machine's IP address to be known in advance.

7.1 Enable ip forwarding

```
$ cat /proc/sys/net/ipv4/ip_forward
      0
$ sudo bash -c 'echo 1 > /proc/sys/net/ipv4/ip_forward'
$ cat /proc/sys/net/ipv4/ip_forward
      1

$ sudo iptables -t nat -A POSTROUTING -s 10.10.0.1/16 ! -o br0 -j
MASQUERADE
```

7.2 Now check finally

a) Namespace “red-ns” to “8.8.8.8”

```
$ sudo ip netns exec red-ns ping -c2 8.8.8.8
```

```
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=27.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=28.1 ms
```

b) Namespace “blue-ns” to “8.8.8.8”

```
$ sudo ip netns exec blue-ns ping -c2 8.8.8.8
```

```
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=27.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=27.9 ms
```

c) Namespace “blue-ns” to “red-ns” IP “10.10.0.10”

```
$ sudo ip netns exec blue-ns ping -c2 10.10.0.10
```

```
64 bytes from 10.10.0.10: icmp_seq=1 ttl=64 time=0.053 ms
64 bytes from 10.10.0.10: icmp_seq=2 ttl=64 time=0.094 ms
```

d) Namespace “red-ns” to “blue-ns” IP “10.10.0.20”

```
$ sudo ip netns exec red-ns ping -c2 10.10.0.20
```

```
64 bytes from 10.10.0.20: icmp_seq=1 ttl=64 time=0.025 ms
64 bytes from 10.10.0.20: icmp_seq=2 ttl=64 time=0.089 ms
```