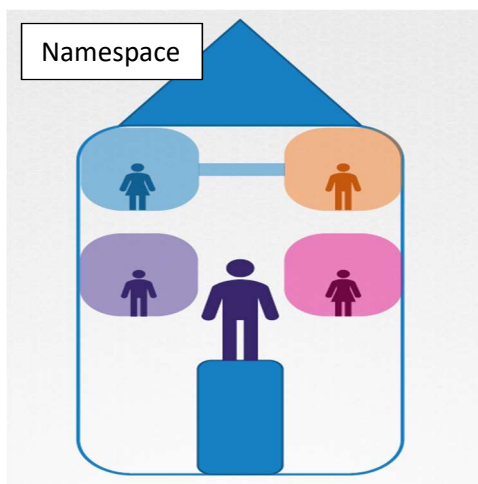


In this video, we get introduced to network namespaces in Linux. Network namespaces are used by containers like Docker to implement network isolation.

We'll start with a simple host. As we know already, containers are separated from the underlying host using namespaces. So what are namespaces? If your host was your house, then namespaces are the rooms within the house that you assign to each of your children. The room helps in providing privacy to each child. Each child can only see what's within his or her room, they cannot see what happens outside their room. As far as they're concerned, they're the only person living in the house. However, as a parent, you have visibility into all the rooms in the house, as well as other areas of the house. If you wish, you can establish connectivity between two rooms in the house.



When you create a container, you want to make sure that it is isolated, that it does not see any other processes on the host, or any other containers. So we create a special room for it on our host using a namespace. As far as the container is concerned, it only sees the processes run by it, and thinks that it is on its own host. The underlying host, however, has visibility into all of the processes, including those running inside the containers. This can be seen when you list the processes from within the container. You see a single process with a process ID of 1. When you list the same processes as a root user from the underlying host, you see all the other processes, along with the process running inside the container, this time with a different process ID. It's the same process running with different process IDs inside and outside the container. That's how namespaces work.

The diagram shows a blue house with a triangular roof. Inside the house, there are four colored circles representing rooms: a blue circle with a person icon, an orange circle with a person icon, a purple circle with a person icon, and a pink circle with a person icon. A central blue figure of a person stands on a blue pedestal in the center of the house. A label 'Namespace' is placed above the house, with a line pointing to the house itself.

```
ps aux (On the container)
```

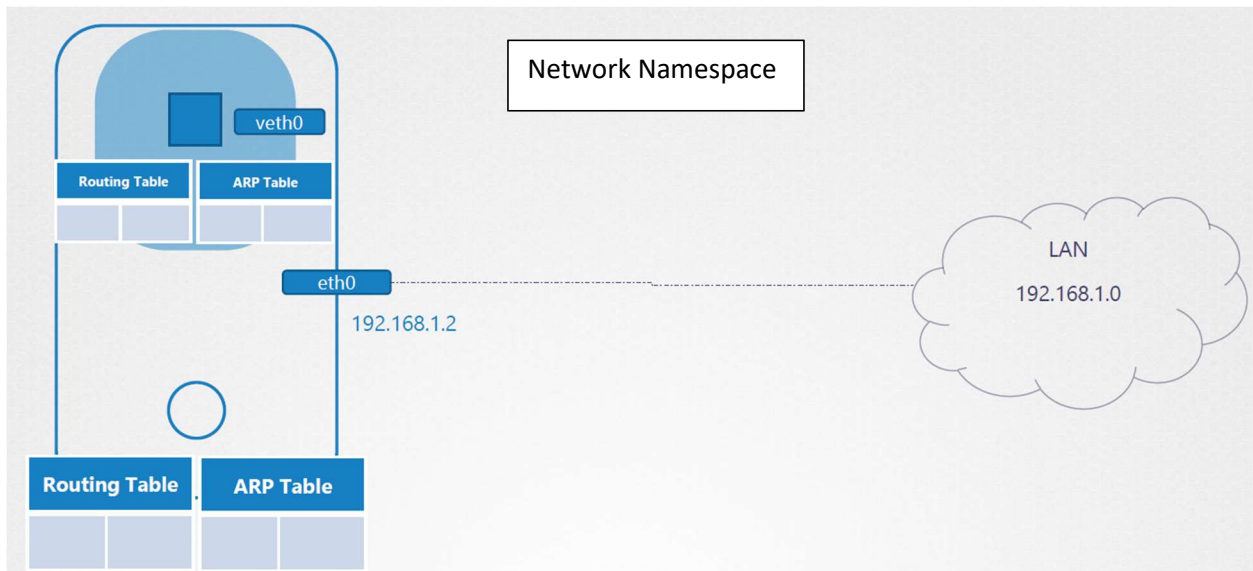
| USER | PID | %CPU | %MEM | VSZ  | RSS | TTY | STAT | START | TIME | COMMAND |
|------|-----|------|------|------|-----|-----|------|-------|------|---------|
| root | 1   | 0.0  | 0.0  | 4528 | 828 | ?   | Ss   | 03:06 | 0:00 | nginx   |

```
ps aux (On the host)
```

| USER    | PID  | %CPU | %MEM | VSZ   | RSS  | TTY   | STAT | START | TIME | COMMAND                             |
|---------|------|------|------|-------|------|-------|------|-------|------|-------------------------------------|
| project | 3720 | 0.1  | 0.1  | 95500 | 4916 | ?     | R    | 06:06 | 0:00 | sshd: project@pts/0                 |
| project | 3725 | 0.0  | 0.1  | 95196 | 4132 | ?     | S    | 06:06 | 0:00 | sshd: project@notty                 |
| project | 3727 | 0.2  | 0.1  | 21352 | 5340 | pts/0 | Ss   | 06:06 | 0:00 | -bash                               |
| root    | 3802 | 0.0  | 0.0  | 8924  | 3616 | ?     | Sl   | 06:06 | 0:00 | docker-containerd-shim -namespace m |
| root    | 3816 | 1.0  | 0.0  | 4528  | 828  | ?     | Ss   | 06:06 | 0:00 | nginx                               |

When it comes to networking, our host has its own interfaces that connect to the local area network. Our host has its own routing and ARP tables with information about rest of the network. We want to seal all of those details from the container.

When the container is created, we create a network namespace for it, that way it has no visibility to any network-related information on the host. Within its namespace, the container can have its own virtual interfaces, routing, and ARP tables. The container has its own interface.



To create a new network namespace on a Linux host, run the **ip netns add** command. In this case, we create two network namespaces. To list the network namespaces, run the **ip netns** command.

```
> ip netns add red
> ip netns add blue
> ip netns
red
blue
```

To list the interfaces on my host, I run the **ip link** command. I see that my host has the loopback interface and the 80 interface. Now, how do we view the same within the network namespace that we created? How do we run the same command within the red or blue namespace? Prefix the command with the command **ip netns exec**, followed by the namespace name, which is red. Now, the **ip link** command will be executed inside the red namespace. Another way to do it is to add the **-n** option to the original **ip link** command. Both of these are the same. The second one is simpler, but remember this only works if you intend to run the **ip** command inside the namespace. As you can see, it only lists the loopback interface, you cannot see the 80 interface on the host.

```

ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc state UP mode DEFAULT qlen 1000
   link/ether 02:42:ac:11:00:08 brd ff:ff:ff:ff:ff:ff

ip netns exec red ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

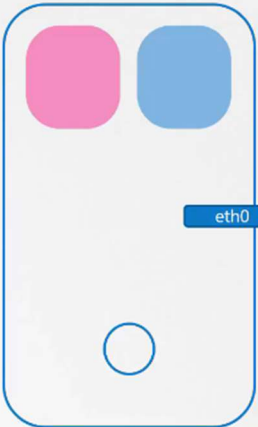
ip -n red link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc state UNKNOWN mode DEFAULT group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

```

```

ip netns exec red ip link
=
ip -n red link

```



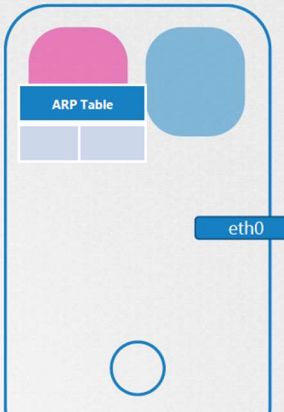
So with namespaces, we have successfully prevented the container from seeing the host interface. The same is true with the ARP table. If you run the ARP command on the host, you see a list of entries, but if you run it inside the container, you see no entries. And the same for routing table.

```

arp
Address HWtype HWaddress Flags Mask Iface
172.17.0.21 ether 02:42:ac:11:00:15 C eth0
172.16.0.8 ether 06:fe:d3:b5:59:65 C eth0
_gateway ether 02:42:d5:7a:84:8e C eth0
host01 ether 02:42:ac:11:00:1c C eth0

ip netns exec red arp
Address HWtype HWaddress Flags Mask Iface

```



| ARP Table   |                   |
|-------------|-------------------|
| 172.17.0.21 | 02:42:ac:11:00:15 |
| 172.16.0.8  | 06:fe:d3:b5:59:65 |

```
route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 0.0.0.0 0.0.0.0 UG 202 0 0 eth0
172.17.0.0 0.0.0.0 255.255.0.0 U 202 0 0 eth0
172.17.0.0 0.0.0.0 255.255.255.0 U 0 0 0 docker0

ip netns exec red route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
```

| Route Table |         |
|-------------|---------|
| 172.17.0.0  | 0.0.0.0 |
| 17.18.0.0   | 0.0.0.0 |

Now as of now, these network namespaces have no network connectivity, they have no interfaces of their own and they cannot see the underlying host network. Let's first look at establishing connectivity between the namespaces themselves. Just like how we would connect two physical machines together using a cable to an internet interface on each machine, you can connect two namespaces together using a virtual ethernet pair, or a virtual cable. It's often referred to as a pipe, but I like to call it a virtual cable with two interfaces on either ends.

To create the cable, run the **ip link add** command with a type set to veth and specify the two ends veth red and veth blue.

The next step is to attach each interface to the appropriate namespace. Use the command **ip link set** veth red netns red to do that. Similarly, attach the blue interface to the blue namespace.

We can then assign ip addresses to each of these namespaces. We will use the usual **ip addr** command to assign the ip address, but within each namespace. We will assign the red namespace an ip , 192.168.15.1. We then assigned the blue namespace an ip , 192.168.15.2.

We then bring up the interface using the **ip link setup** command for each device within the respective namespaces. The links are up and the namespaces can now reach each other.

Try a ping from the red namespace to reach the IP of the blue.

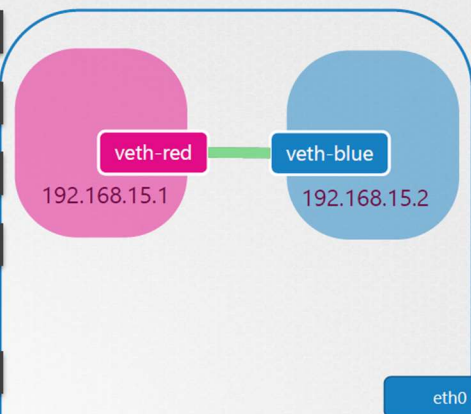
```

▶ ip link add veth-red type veth peer name veth-blue
▶ ip link set veth-red netns red
▶ ip link set veth-blue netns blue
▶ ip -n red addr add 192.168.15.1 dev veth-red
▶ ip -n blue addr add 192.168.15.2 dev veth-blue

▶ ip -n red link set veth-red up
▶ ip -n blue link set veth-blue up

▶ ip netns exec red ping 192.168.15.2
PING 192.168.15.2 (192.168.15.2) 56(84) bytes of data:
64 bytes from 192.168.15.2: icmp_seq=1 ttl=64 time=0.026 ms

```



KODECLOUD

If you look at the ARP table on the red namespace, you see it's identified its blue neighbor at 192.168.15.2 with a mac address. Similarly, if you list the ARP table on the blue namespace, you see it's identified its red neighbor.

If you compare this with the ARP table of the host, you see that the host ARP table has no idea about these new namespaces we have created, and no idea about the interfaces we created in them.

```

▶ ip netns exec red arp

```

| Address      | HWtype | HWaddress         | Flags | Mask | Iface    |
|--------------|--------|-------------------|-------|------|----------|
| 192.168.15.2 | ether  | ba:b0:6d:68:09:e9 | C     |      | veth-red |

```

▶ ip netns exec blue arp

```

| Address      | HWtype | HWaddress         | Flags | Mask | Iface     |
|--------------|--------|-------------------|-------|------|-----------|
| 192.168.15.1 | ether  | 7a:9d:9b:c8:3b:7f | C     |      | veth-blue |

```

▶ arp

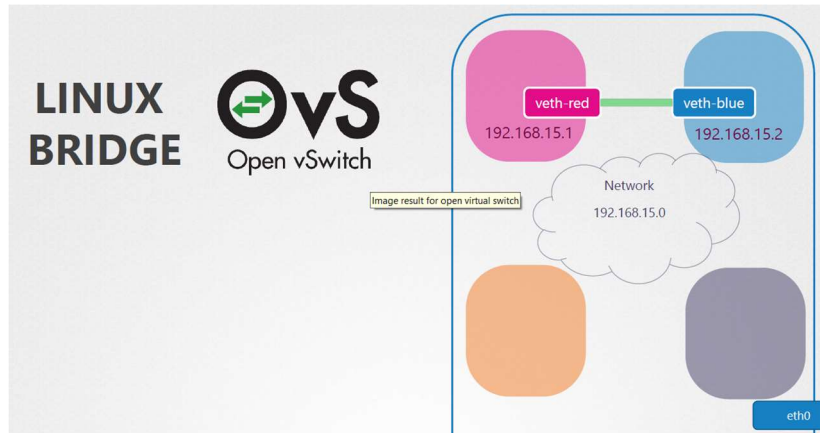
```

| Address     | HWtype | HWaddress         | Flags | Mask | Iface |
|-------------|--------|-------------------|-------|------|-------|
| 192.168.1.3 | ether  | 52:54:00:12:35:03 | C     |      | eth0  |
| 192.168.1.4 | ether  | 52:54:00:12:35:04 | C     |      | eth0  |

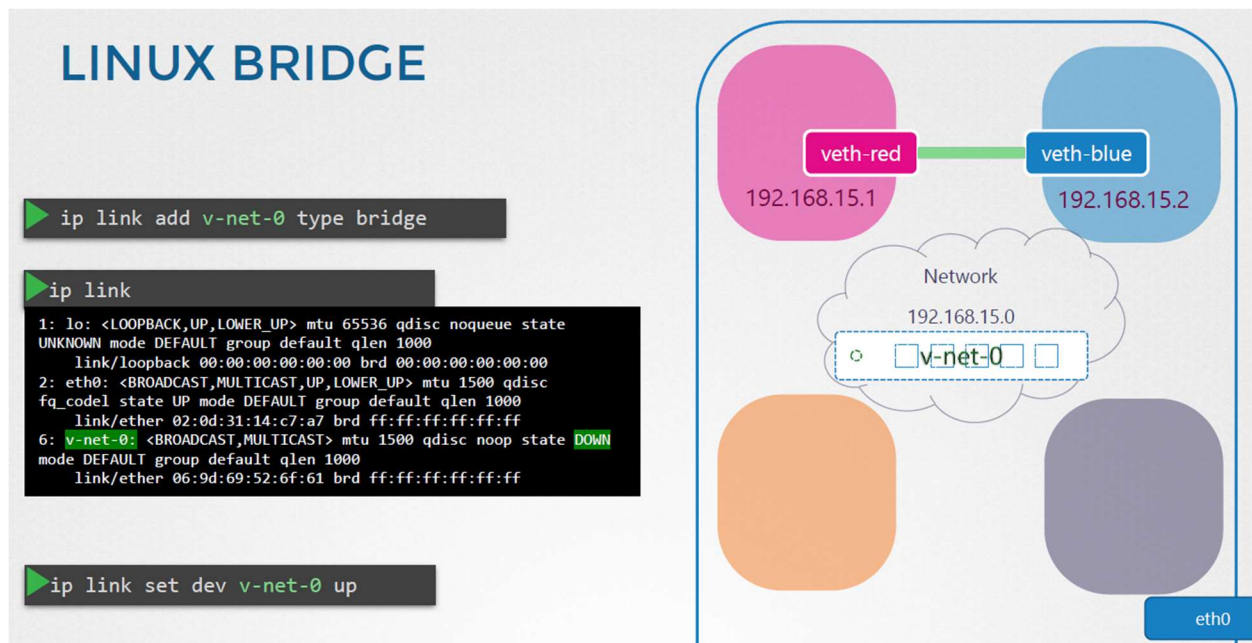
Now, that worked when you had just two namespaces, what do you do when you have more of them? How do you enable all of them to communicate with each other? Just like in the physical world, you create a virtual network inside your host. Create a network, you need a switch, so to create a virtual network,



you need a virtual switch. So you create a virtual switch within our host, and connect the namespaces to it. But how do you create a virtual switch within a host? There are multiple solutions available, such as the native solution, called as Linux bridge, and the open V switch, etc. In this example, we will use the Linux bridge option.

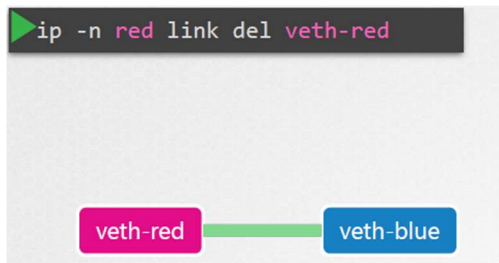


To create an internal bridge network, we add a new interface to the host using the `ip link add` command with the type set to bridge. We will name it `vnet0`. As far as our host is concerned, it is just another interface, just like the `eth0` interface. It appears in the output of the `ip link` command along with the other interfaces. It's currently down, so you need to turn it up. Use `ip link set up` command to bring it up. Now, for the namespaces, this interface is like a switch that it can connect to. So think of it as an interface for the host and a switch for the namespaces.

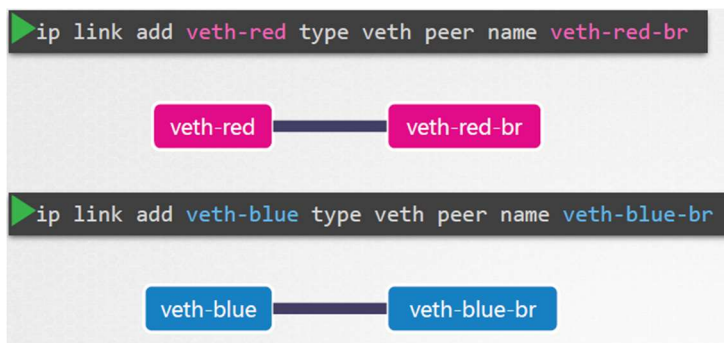


So the next step is to connect the namespaces to this new virtual network switch. Earlier, we created the cable, or the `eth` pair with the `veth red` interface on one end and `blue` interface on the other because we wanted to connect the two namespaces directly. Now, we will be connecting all namespaces to the bridge network. So we need new cables for that purpose. This cable doesn't make sense anymore so we will get

rid of it. Use the **ip link delete** command to delete the cable. When you delete the link with one end the other end gets deleted automatically since they are a pair.



Let us now create new cables to connect the namespaces to the bridge. Run the `ip link add` command and create a pair with veth red on one end, like before, but this time the other end will be named. Veth red VR as it connects to the bridge network. This naming convention will help us easily identify the interfaces that associate to the red namespace. Similarly, create a cable to connect the blue namespace to the bridge network.



Now that we have the cables ready, it's time to get them connected to the namespaces. To attach one end of this, of the interface, to the red namespace, run the `ip link set veth red netns red` command. To attach the other end to the bridge network, run the `ip link set` command on the veth red VR end and specify the master for it. As the VNET zero network.

Follow the same procedure to attach the blue cable to the blue namespace and the bridge network. Let us now set ip addresses for these links and turn them up. We will use the same ip addresses that we used before 192.168.15.1, and 192.168.15.2. And finally turn the devices up. The containers can now reach each other over the network.

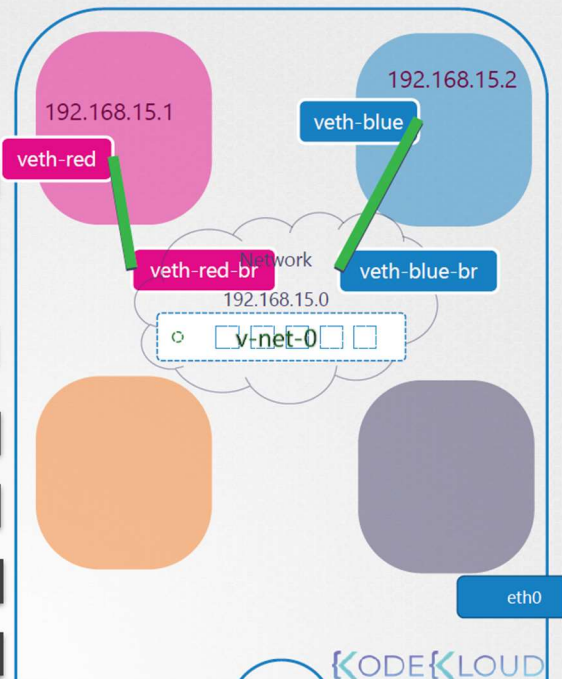
So we follow the same procedure to connect the remaining two namespaces to the same network. We now have all four namespaces connected to our internal bridge network and they can all communicate with each other.

# LINUX BRIDGE

```
➤ ip link set veth-red netns red
➤ ip link set veth-red-br master v-net-0
➤ ip link set veth-blue netns blue
➤ ip link set veth-blue-br master v-net-0

➤ ip -n red addr add 192.168.15.1 dev veth-red
➤ ip -n blue addr add 192.168.15.2 dev veth-blue

➤ ip -n red link set veth-red up
➤ ip -n blue link set veth-blue up
```



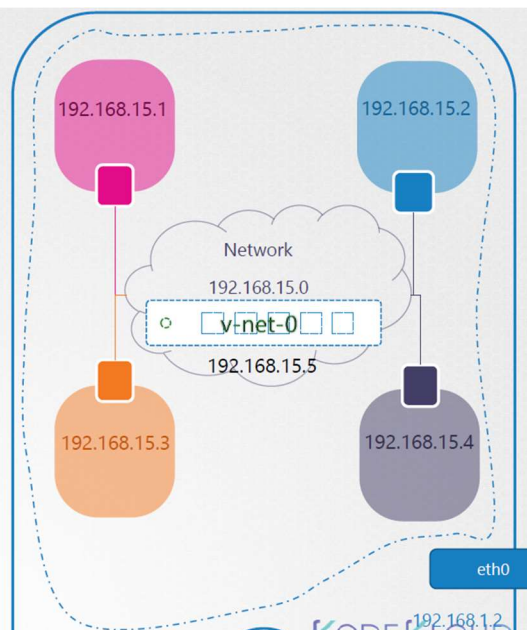
They have all ip addresses. 192.168.15.1, 2, 3, and 4. And remember, we assigned our host the IP 192.168.1.2 from my host. What if I tried to reach one of these interfaces in these namespaces? Will it work? No. My host is on one network and the namespaces are on another. But what if I really want to establish connectivity between my host and these namespaces? Remember we said that the veth switch is actually a network interface for the host. So we do have an interface on the 192.168.15 network on our host. Since this just another interface all we need to do is assign an ip address to it so we can reach the namespaces through it. Run the ip addr command to set the IP 192.168.15.5 to this interface. We can now ping the red namespace from our local host.

# LINUX BRIDGE

```
➤ ping 192.168.15.1
Not Reachable!

➤ ip addr add 192.168.15.5/24 dev v-net-0

➤ ping 192.168.15.1
PING 192.168.15.1 (192.168.15.1) 56(84) bytes of data.
64 bytes from 192.168.15.1: icmp_seq=1 ttl=64 time=0.026 ms
```

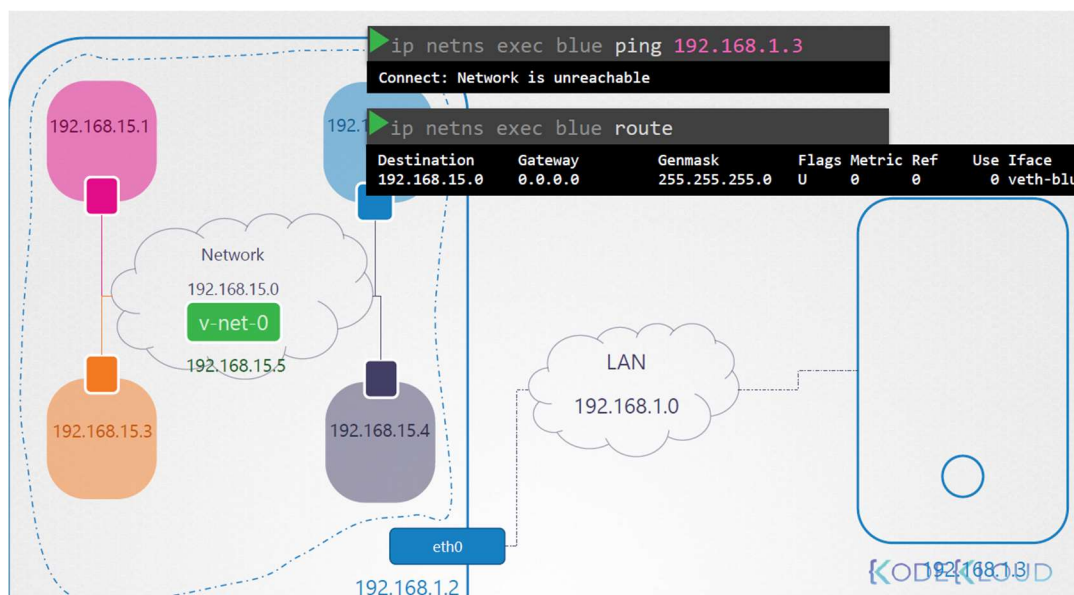




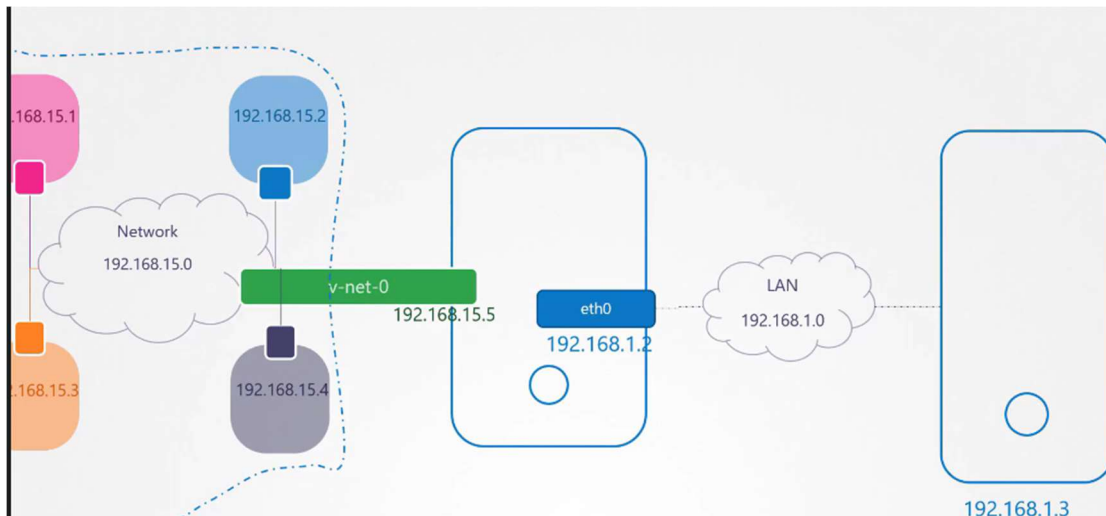
Now remember, this entire network is still private and restricted within the host. From within the namespaces, you can't reach the outside world nor can anyone from the outside world reach the services or applications hosted inside.

The only door to the outside world is the Ethernet port on the host.

So how do we configure this bridge to reach the line network through the Ethernet port? Say there is another host attached to our line network with the address 192.168.1.3. How can I reach this host from within my namespaces? What happens if I try to ping this host from my namespace? The blue namespace sees that I'm trying to reach a network at 192.168.1, which is different from my current network of 192.168.15. So it looks at its routing table to see how to find that network. The routing table has no information about other network. So it comes back saying that the network is unreachable. So we need to add an entry into the routing table to provide a gateway or door to the outside world.



So how do we find that gateway? A door or a gateway, as we discussed before, is a system on the local network that connects to the other network. So what is a system that has one interface on the network local to the blue namespace which is the 192.168.15 network and is also connected to the outside LAN network? Here's a logical view.



It's the local host that have all these namespaces on so you can ping the namespaces. Remember, our local host has an interface to attach the private network so you can ping the namespaces. So our local host is the gateway that connects the two networks together. We can now add a row entry in the blue namespace to say route all traffic to the 192.168.1 network through the gateway at 192.168.15.5. Now remember, our host has two ip addresses; one on the bridge network at 192.168.15.5 and another on the external network at 192.168.1.2. Can you use any in the route? No, because the blue namespace can only reach the gateway in its local network at 192.168.15.5. The default gateway should be reachable from your namespace when you add it to your route.

```
ip netns exec blue ip route add 192.168.1.0/24 via 192.168.15.5
```

| Destination  | Gateway      | Genmask       | Flags | Metric | Ref | Use | Iface     |
|--------------|--------------|---------------|-------|--------|-----|-----|-----------|
| 192.168.15.0 | 0.0.0.0      | 255.255.255.0 | U     | 0      | 0   | 0   | veth-blue |
| 192.168.1.0  | 192.168.15.5 | 255.255.255.0 | UG    | 0      | 0   | 0   | veth-blue |

When you try to ping now, you no longer get the network unreachable message, but you still don't get any response back from the ping.

```
ip netns exec blue ping 192.168.1.3
```

```
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
```

What might be the problem? We talked about a similar situation in one of our earlier lectures where, from our home network, we tried to reach the external internet through our router. Our home network has our internal private ip addresses that the destination network don't know about so they cannot reach back. For this, we need NAT enable on our host acting as a gateway here so that it can send the messages to the LAN in its own name with its own address.

So how do we add NAT functionality to our host? You should do that using IP tables. Add a new rule in the NAT IP table in the post routing chain to masquerade or replace the from address on all packets coming from the source network. 192.168.15.0 with its own ip address. That way, anyone receiving these packets outside the network will think that they're coming from the host and not from within the namespaces. When we try to ping now, we see that we are able to reach the outside world. Finally, say the LAN is connected to the internet. We want the namespaces to reach the internet. So we try to ping a server on

the internet at 8.8.8.8 from the blue namespace. We receive a similar message that the network is unreachable. By now we know why that is. We look at the routing table and see that we have routes to the network, 192.168.1, but not to anything else. Since these namespaces can reach any network our host can reach, we can simply say that, to reach any external network, talk to our host. So we add a default gateway specifying our host. We should now be able to reach the outside world from within these namespaces.

```
ip netns exec blue ping 8.8.8.8
```

Connect: Network is unreachable

```
ip netns exec blue route
```

| Destination  | Gateway      | Genmask       | Flags | Metric | Ref | Use | Iface     |
|--------------|--------------|---------------|-------|--------|-----|-----|-----------|
| 192.168.15.0 | 0.0.0.0      | 255.255.255.0 | U     | 0      | 0   | 0   | veth-blue |
| 192.168.1.0  | 192.168.15.5 | 255.255.255.0 | UG    | 0      | 0   | 0   | veth-blue |

```
ip netns exec blue ip route add default via 192.168.15.5
```

| Destination  | Gateway      | Genmask       | Flags | Metric | Ref | Use | Iface     |
|--------------|--------------|---------------|-------|--------|-----|-----|-----------|
| 192.168.15.0 | 0.0.0.0      | 255.255.255.0 | U     | 0      | 0   | 0   | veth-blue |
| 192.168.1.0  | 192.168.15.5 | 255.255.255.0 | UG    | 0      | 0   | 0   | veth-blue |
| Default      | 192.168.15.5 | 255.255.255.0 | UG    | 0      | 0   | 0   | veth-blue |

```
ip netns exec blue ping 8.8.8.8
```

64 bytes from 8.8.8.8: icmp\_seq=1 ttl=63 time=0.587 ms  
64 bytes from 8.8.8.8: icmp\_seq=2 ttl=63 time=0.466 ms

Now, what about connectivity from the outside world to inside the namespaces? Say for example, the blue namespace hosts a web application on Port 80. As of now, the namespaces are on an internal private network and no one from the outside world knows about them. You can only access these from the host itself. If you try to ping the private IP of the namespace from another host on another network, you will see that it's not reachable, obviously, because that host doesn't know about this private network.

In order to make that communication possible you have two options. The two options that we saw in the previous lecture on that. The first is to give away the identity of the private network to the second host. So we basically add an IP route entry to the second host telling the host that the network 192.168.15 can be reached through the host at 192.168.1.2. But we don't want to do that. The other option is to add a port forwarding role using IP tables to say any traffic coming to Port 80 on the local host is to be forwarded to port 80 on the IP assigned to the blue namespace.