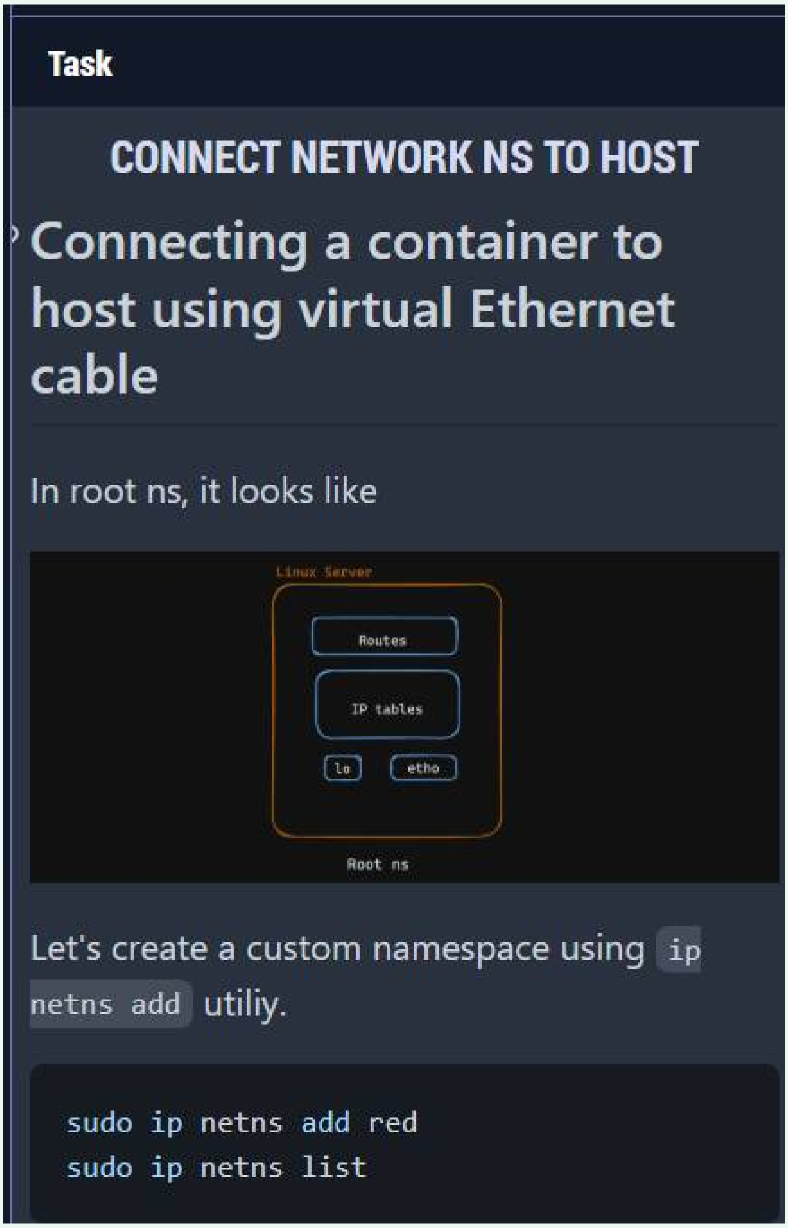
Container Network - Connecting Namespace to Host



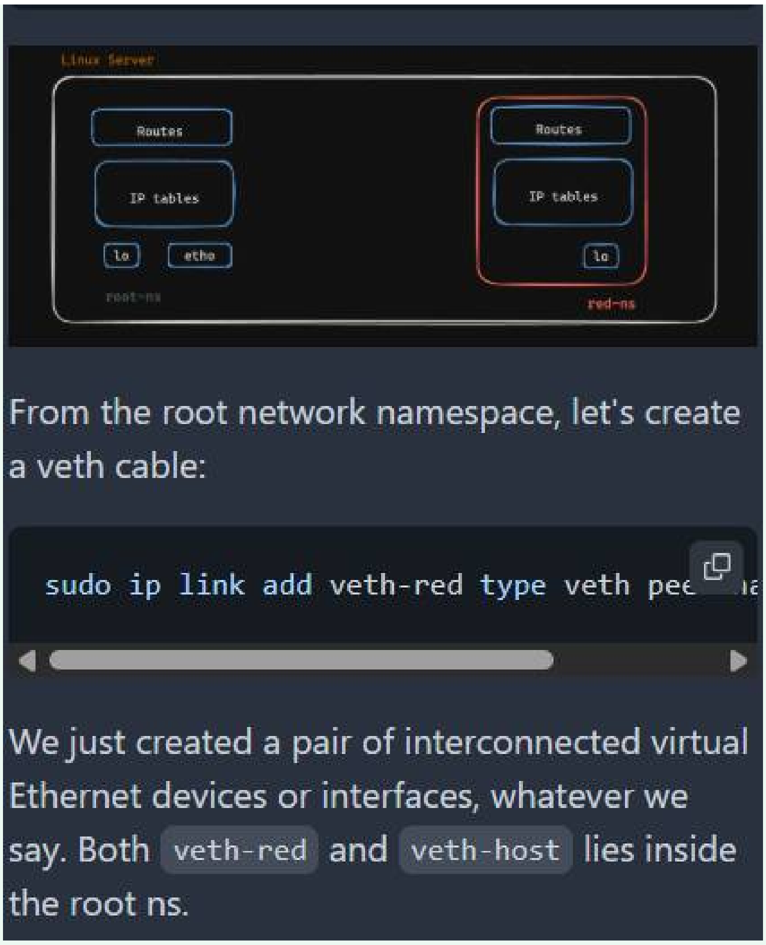
CONNECT NETWORK NS TO HOST

Connecting a container to host using virtual Ethernet cable

In root ns, it looks like

Let's create a custom namespace using `ip netns add` utiliy.

```
sudo ip netns add red
sudo ip netns list
```

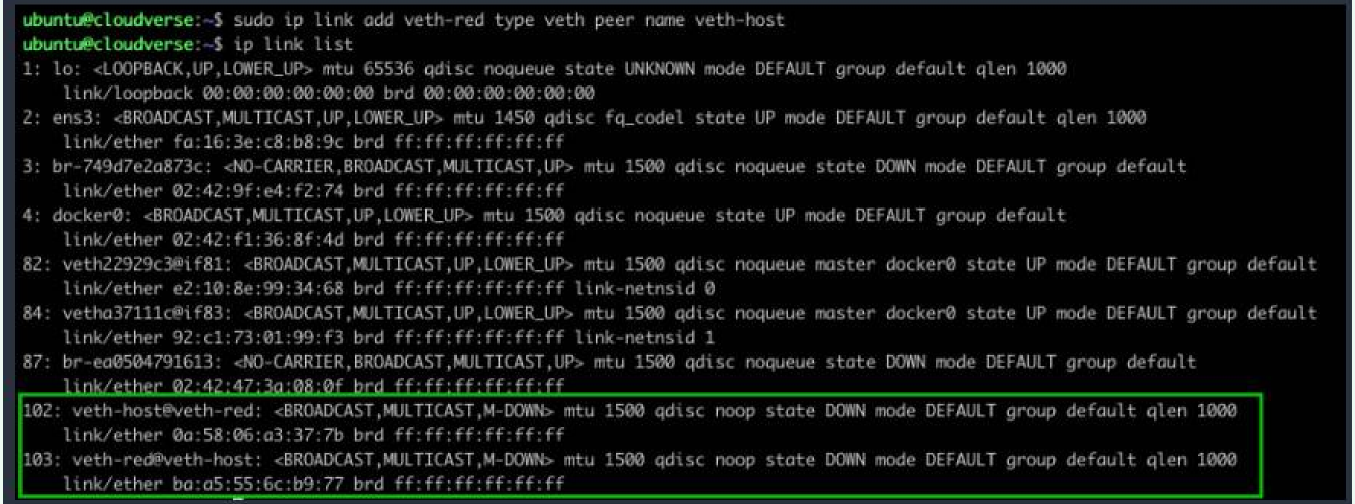


From the root network namespace, let's create a veth cable:

```
sudo ip link add veth-red type veth pee
```

We just created a pair of interconnected virtual Ethernet devices or interfaces, whatever we say. Both `veth-red` and `veth-host` lies inside the root ns.

```
ip link list
```



```
ubuntu@cloudverse:~$ sudo ip link add veth-red type veth peer name veth-host
ubuntu@cloudverse:~$ ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether fa:16:3e:c8:b8:9c brd ff:ff:ff:ff:ff:ff
3: br-749d7e2a873c: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:9f:e4:f2:74 brd ff:ff:ff:ff:ff:ff
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT group default
    link/ether 02:42:f1:36:8f:4d brd ff:ff:ff:ff:ff:ff
82: veth22929c3@if81: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP mode DEFAULT group default
    link/ether e2:10:8e:99:34:68 brd ff:ff:ff:ff:ff:ff link-netnsid 0
84: vetha37111c@if83: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP mode DEFAULT group default
    link/ether 92:c1:73:01:99:f3 brd ff:ff:ff:ff:ff:ff link-netnsid 1
87: br-ea0504791613: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:47:3a:08:0f brd ff:ff:ff:ff:ff:ff
102: veth-host@veth-red: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether 0a:58:06:a3:37:7b brd ff:ff:ff:ff:ff:ff
103: veth-red@veth-host: <BROADCAST,MULTICAST,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
    link/ether ba:a5:55:6c:b9:77 brd ff:ff:ff:ff:ff:ff
```

To connect the root namespace with the `red` namespace, we need to keep one end of the cable in the root namespace and move another one end into the `red` ns.

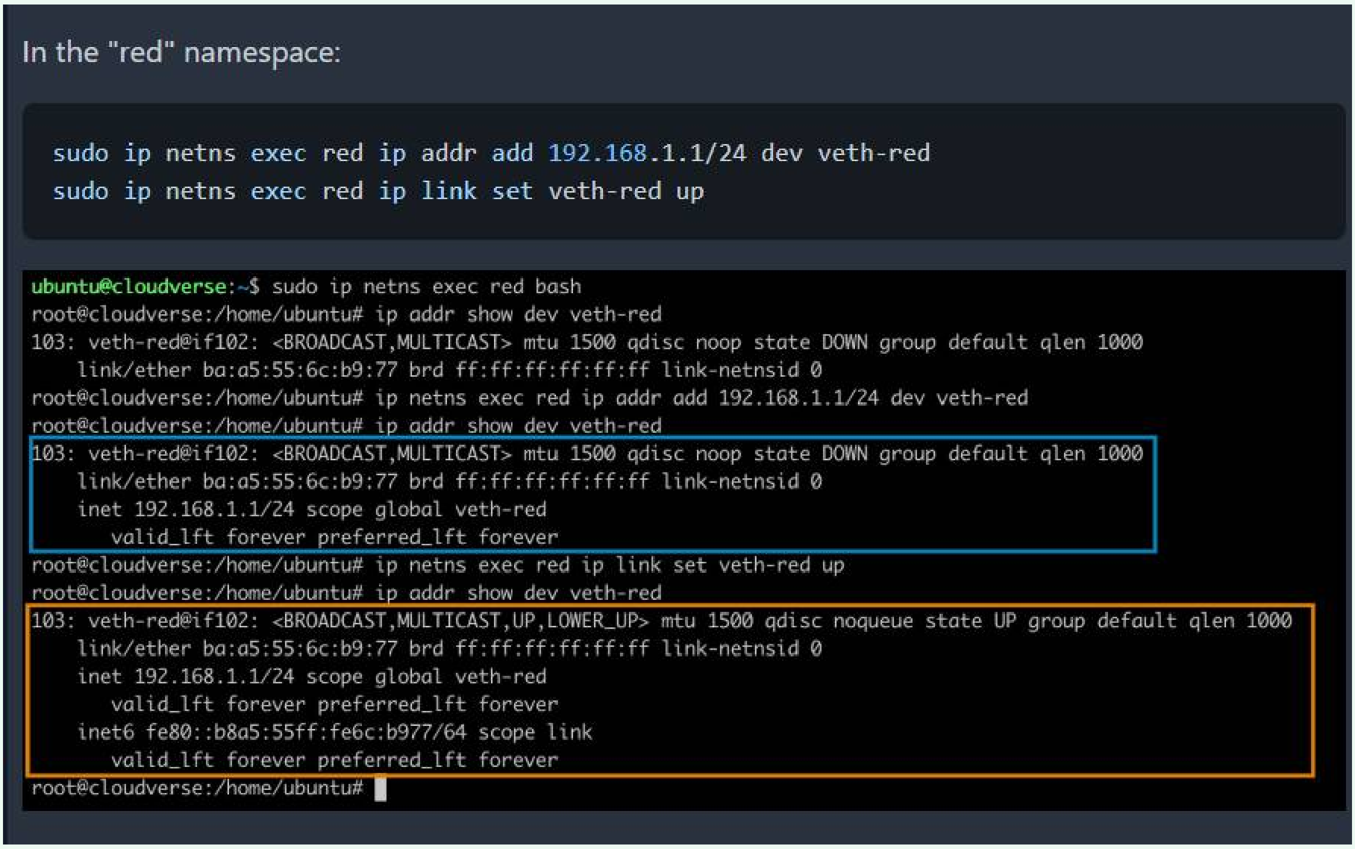
```
sudo ip link set veth-red netns red
```

This moves one end of the veth pair (veth-red) into the "red" namespace. The other end (veth-host) remains in the default network namespace.

Now, let's configure IP Addresses to both end of this veth cable and once we turn up the interfaces, the peer device will instantly display any packet that appears on one of the devices.

In the "red" namespace:

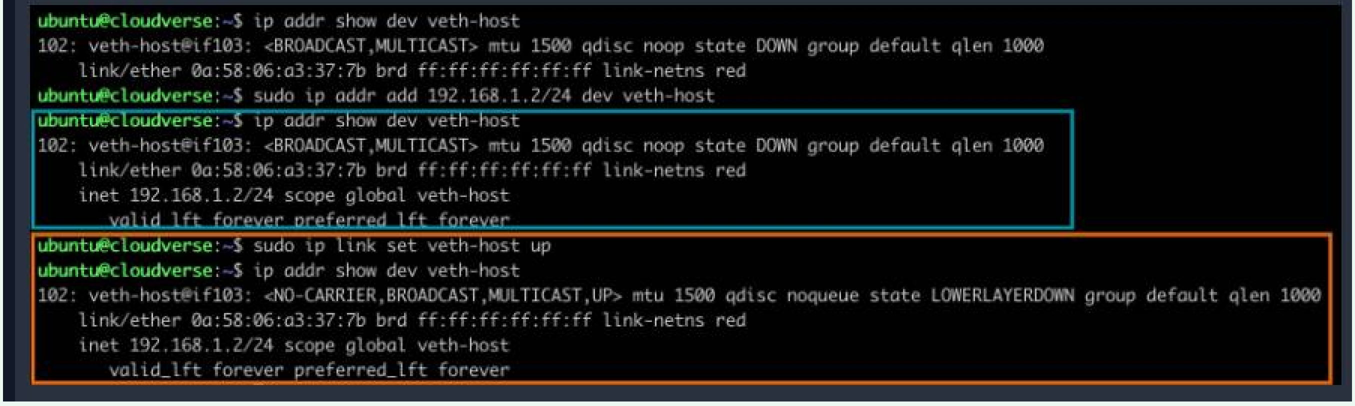
```
sudo ip netns exec red ip addr add 192.168.1.1/24 dev veth-red
sudo ip netns exec red ip link set veth-red up
```



```
ubuntu@cloudverse:~$ sudo ip netns exec red bash
root@cloudverse:/home/ubuntu# ip addr show dev veth-red
103: veth-red@if102: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether ba:a5:55:6c:b9:77 brd ff:ff:ff:ff:ff:ff link-netnsid 0
root@cloudverse:/home/ubuntu# ip netns exec red ip addr add 192.168.1.1/24 dev veth-red
root@cloudverse:/home/ubuntu# ip addr show dev veth-red
103: veth-red@if102: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether ba:a5:55:6c:b9:77 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.1/24 scope global veth-red
       valid_lft forever preferred_lft forever
root@cloudverse:/home/ubuntu# ip netns exec red ip link set veth-red up
root@cloudverse:/home/ubuntu# ip addr show dev veth-red
103: veth-red@if102: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether ba:a5:55:6c:b9:77 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.1.1/24 scope global veth-red
       valid_lft forever preferred_lft forever
    inet6 fe80::b8a5:55ff:fe6c:b977/64 scope link
       valid_lft forever preferred_lft forever
root@cloudverse:/home/ubuntu# █
```

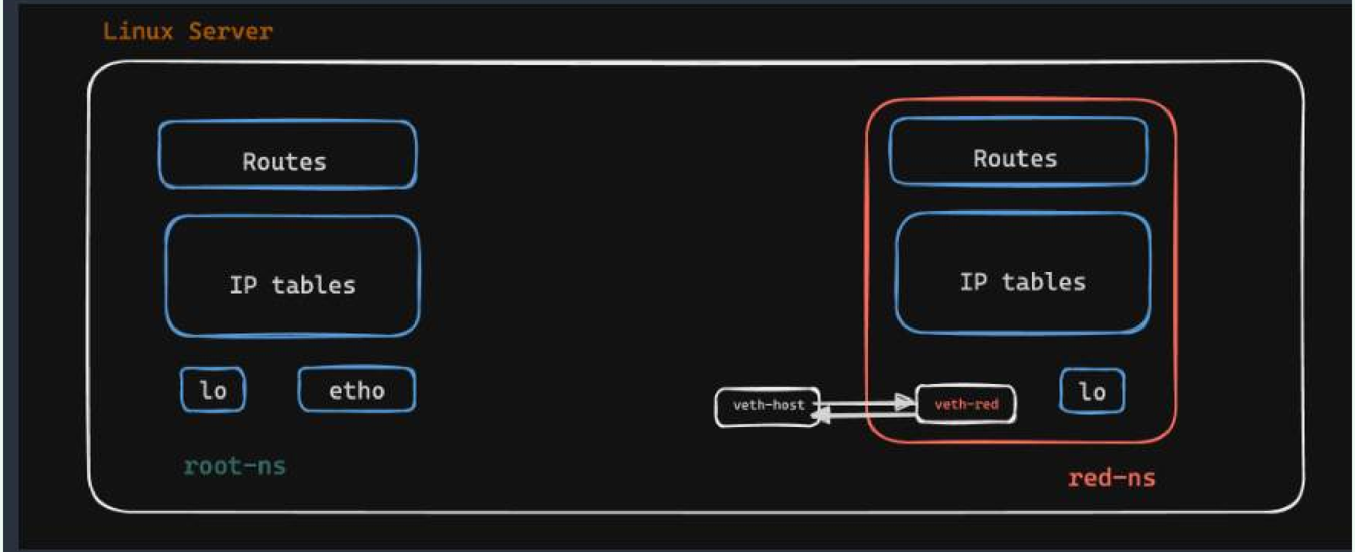
On the host side:

```
sudo ip addr add 192.168.1.2/24 dev veth-host
sudo ip link set veth-host up
```



```
ubuntu@cloudverse:~$ ip addr show dev veth-host
102: veth-host@if103: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 0a:58:06:a3:37:7b brd ff:ff:ff:ff:ff:ff link-netns red
ubuntu@cloudverse:~$ sudo ip addr add 192.168.1.2/24 dev veth-host
ubuntu@cloudverse:~$ ip addr show dev veth-host
102: veth-host@if103: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 0a:58:06:a3:37:7b brd ff:ff:ff:ff:ff:ff link-netns red
    inet 192.168.1.2/24 scope global veth-host
       valid_lft forever preferred_lft forever
ubuntu@cloudverse:~$ sudo ip link set veth-host up
ubuntu@cloudverse:~$ ip addr show dev veth-host
102: veth-host@if103: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state LOWERLAYERDOWN group default qlen 1000
    link/ether 0a:58:06:a3:37:7b brd ff:ff:ff:ff:ff:ff link-netns red
    inet 192.168.1.2/24 scope global veth-host
       valid_lft forever preferred_lft forever
```

The virtual ethernet pair is now ready.



A route is needed to add on the host to direct traffic destined for 192.168.1.1 through the veth-host interface.

```
sudo ip route add 192.168.1.1 dev veth-host
```

Why Add a Route?

Without adding the route, the system doesn't know how to reach 192.168.1.1. When we ping 192.168.1.1, the system checks its routing table to determine where to send the ping packets. If there's no specific route for 192.168.1.1, it won't know which interface to use.

Why Add a Route?

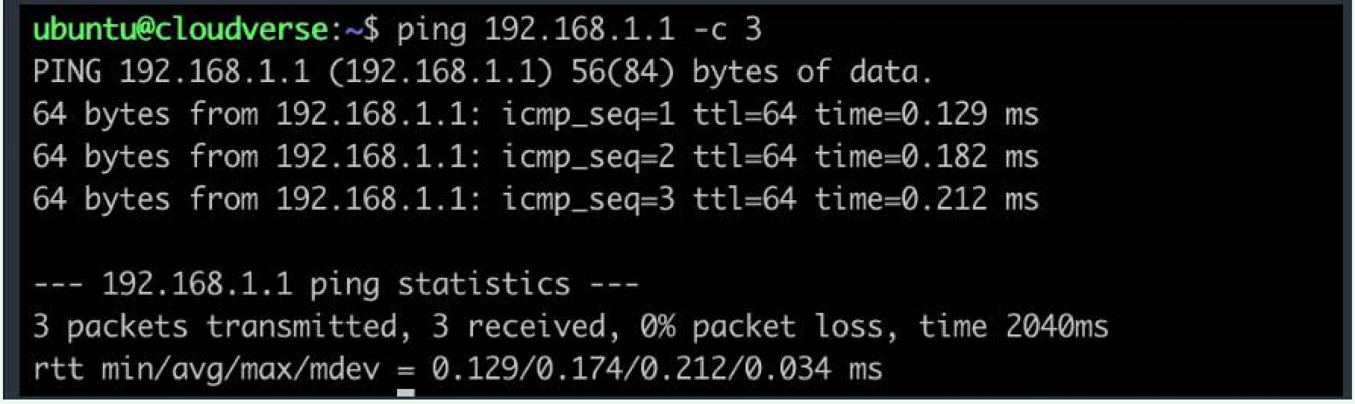
Without adding the route, the system doesn't know how to reach 192.168.1.1. When we ping 192.168.1.1, the system checks its routing table to determine where to send the ping packets. If there's no specific route for 192.168.1.1, it won't know which interface to use.

By adding the route, we are explicitly telling the system that to reach 192.168.1.1, it should send the traffic through the veth-host interface, which is part of the veth pair connected to the "red" namespace.

Test connectivity

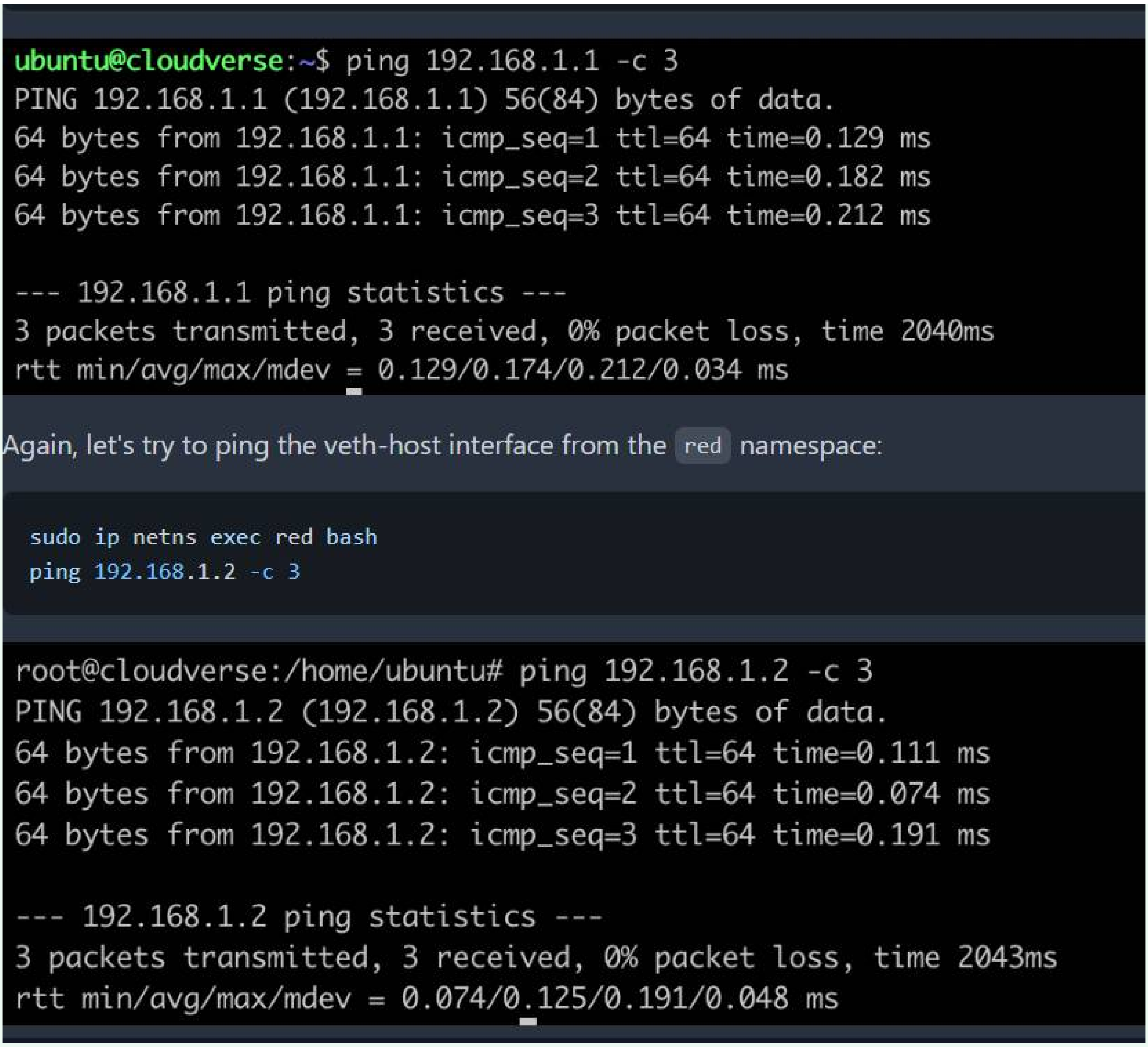
Let's try to ping the `red` ns from the veth-host interface:

```
ping 192.168.1.1 -c 3
```



```
ubuntu@cloudverse:~$ ping 192.168.1.1 -c 3
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.129 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.182 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.212 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2040ms
rtt min/avg/max/mdev = 0.129/0.174/0.212/0.034 ms
```



```
ubuntu@cloudverse:~$ ping 192.168.1.1 -c 3
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.129 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.182 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.212 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2040ms
rtt min/avg/max/mdev = 0.129/0.174/0.212/0.034 ms
```

Again, let's try to ping the veth-host interface from the `red` namespace:

```
sudo ip netns exec red bash
ping 192.168.1.2 -c 3
```

```
root@cloudverse:/home/ubuntu# ping 192.168.1.2 -c 3
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.111 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.074 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.191 ms

--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2043ms
rtt min/avg/max/mdev = 0.074/0.125/0.191/0.048 ms
```

explain these for a simple child in a very easiest manner in details with basics also correct any error

It looks like you're trying to understand how to connect a **container (or a separate network namespace) to the host** using a virtual Ethernet cable. Let me explain this in the **simplest way possible** with a step-by-step explanation.

🌍 Imagine This:

- Think of your computer as a **big house** 🏠 with multiple **rooms**.
- Each **room** is a different **network namespace**.
- These rooms do **not** talk to each other directly.
- To **connect two rooms**, you need a **cable** (virtual Ethernet cable or `` `veth` ``).

Step 1: Create a New "Room" (Network Namespace)

🔹 By default, everything is inside one **main room** called the **root namespace** (where your normal network lives).

🔹 Let's create a new **separate room** (a new network namespace) and name it **"red"**.

```
bash

sudo ip netns add red
sudo ip netns list  # Check if it's created
```

🎯 **Now, we have two rooms:**

- The **root namespace** (default network).
- The **red namespace** (isolated network space).

Step 2: Create a Virtual Cable (veth Pair)

🔹 We need a **cable** 🏗️ to connect the two rooms.

```
bash

sudo ip link add veth-red type veth peer name veth-host
```

🎯 **What happens here?**

- We create **two virtual ends** of a cable:
    - **veth-red** (this will go inside the "red" namespace).
    - **veth-host** (this will stay in the root namespace).

🚀 **These two ends act like a network cable connecting two computers!**

Step 3: Place One End of the Cable Inside the "Red" Room

🔹 Now, we need to put one end (**veth-red**) inside the **red namespace**.

```
bash

sudo ip link set veth-red netns red
```

🎯 **Now:**

- ``` `veth-red` ``` is inside the **red** namespace.
- ``` `veth-host` ``` is still inside the **root** namespace.

Step 4: Assign IP Addresses

🔹 Just like in real networks, we need to give each end an **IP address** so they can communicate.

🌟 Assign IP inside the "red" namespace

```bash
sudo ip netns exec red ip addr add 192.168.1.1/24 dev veth-red
sudo ip netns exec red ip link set veth-red up
```

🎯 This means:

- `veth-red` in the "red" namespace has IP `192.168.1.1`.

🌟 Assign IP in the root namespace

```bash
sudo ip addr add 192.168.1.2/24 dev veth-host
sudo ip link set veth-host up
```

🎯 Now:

- `veth-host` in the root namespace has IP `192.168.1.2`.

Step 5: Add a Route for Communication

🔹 The system doesn't **automatically know** where to send packets. We need to **tell** it!

```bash
sudo ip route add 192.168.1.1 dev veth-host
```

🎯 This tells the system:

*"If you want to send data to 192.168.1.1, use the **veth-host** interface."*

Step 6: Test the Connection

🔹 Now, let's **ping** from the host to the "red" namespace:

```bash
ping 192.168.1.1 -c 3
```

🔹 Then, let's **ping from red to the host**:

```bash
sudo ip netns exec red ping 192.168.1.2 -c 3
```

🎯 If both pings work, the connection is successful! 🎉

✅ Recap:

1. **Created a new network namespace** ("red").
2. **Made a virtual Ethernet cable** (`veth-red` ↔ `veth-host`).
3. **Moved one end of the cable into "red" namespace**.
4. **Assigned IP addresses** to both ends.
5. **Added a route** so traffic knows where to go.
6. **Tested the connection** using `ping`.

Now, **your container (red namespace) is connected to the host!** 🚀🔗

Let me know if you need **more clarity** or if you want to go **even simpler!** 😊