Network Namespaces in RHEL7

Finnbarr P. Murphy
(fpm@fpmurphy.com)

Linux *namespaces* are somewhat like Solaris zones in many ways from a user perspective but have significant differences under the hood. The term *namespace isolation* is often used because the purpose of namespaces is to provide a group of processes with the illusion that that they are the only processes on the system. This is an important requirement for implementing *Linux Containers*.

Namespaces were developed over a number of years by Eric W. Biederman (user namespaces), Pavel Emelyanov, Al Viro, Cyrill Gorcunov, *et al*.

Six user namespaces (out of 10 proposed) are implemented in RHEL7:

- *mnt* – mount points, filesystems
- *pid* – processes
- *net* – network stack
- *ipc* – System V IPC
- *uts* – hostname, domainname
- *user* – UIDs, GIDs

The *mnt* namespace first appeared in the Linux kernel in 2002 whereas the *user* namespace work was only completed in the last few years.

Turning now to the network (*(CLONE_NEWNET)*) namespace. A network namespace provides a private view of the network for a group of processes. The namespace includes private network devices and IP addresses, so that each group of processes has its own port number space. Network namespaces can also make packet filtering easier, since each group of processes has its own network device. A network device belongs to exactly one network namespace. Similarly, a socket belongs to exactly one network namespace.

Before we start modifying things, here is my default RHEL7 network configuration:

```
# ifconfig -a
ens33: flags=4163  mtu 1500
       inet 192.168.100.20  netmask 255.255.255.0  broadcast 192.168.100.255
       ether 00:0c:29:cb:9e:30  txqueuelen 1000  (Ethernet)
       RX packets 361  bytes 42328 (41.3 KiB)
       RX errors 0  dropped 0  overruns 0  frame 0
       TX packets 607  bytes 63935 (62.4 KiB)
       TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10
        loop  txqueuelen 0  (Local Loopback)
        RX packets 1351  bytes 123356 (120.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1351  bytes 123356 (120.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

I am actually running RHEL7 as a VM in VMware Workstation.

Looking at the synopsis of the *ip* utility in RHEL7:


```
NAME
       ip - show / manipulate routing, devices, policy routing and tunnels

SYNOPSIS
       ip [ OPTIONS ] OBJECT { COMMAND | help }

       ip [ -force ] -batch filename

       OBJECT := { link | addr | addrlabel | route | rule | neigh | ntable |
                tunnel | tuntap | maddr | mroute | mrule | monitor | xfrm |
                netns | l2tp | tcp_metrics }

       OPTIONS := { -V[ersion] | -s[tatistics] | -r[esolve] | -f[amily] { inet
                | inet6 | ipx | dnet | link } | -o[neline] }
```

Notice the *netns* object. This is used to manage network namespaces. So what can we do with it?

To create and list a network namespace:

```
# ip netns add mynamespace
# ip netns list
mynamespace
#
```

A network namespace is logically another copy of the network stack with its own routes, firewall rules, and network devices.

When a network namespace is created, a bind mount for it is created under */var/run/netns/*. This is to allow the namespace to persist even when no processes are running within it and to facilitate the manipulation of the namespace itself. A new namespace has no opened sockets. By convention a named network namespace is an object at */var/run/netns/NAMESPACE* that can be opened. The file descriptor resulting from opening the object refers to the specified network namespace.

A new network namespace will have a loopback device but no other network devices. The loopback device will be in the down state.

```
# ip netns exec mynamespace ip link list
1: lo:  mtu 65536 qdisc noop state DOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

# ip netns exec mynamespace ping 127.0.0.1
connect: Network is unreachable
# ip netns exec mynamespace ip link set dev lo up
# ip netns exec mynamespace ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.198 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.046 ms
^C
--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.046/0.089/0.198/0.063 ms
#
```

Note that the command sequence *ip netns exec* is how you execute commands in a different network namespace.

To delete a network namespace:

```
# ip netns del mynamespace
# ip netns list
#
```

This command also removes the bind mount referring to the given network namespace. However the namespace itself persists for as long as any processes are running within it.

The default namespace is called the root namespace. So how do you communicate between *mynamespace* and the root namespace? To do that, a pair of virtual Ethernet devices need to be created and configured.

```
# ip link add veth0 type veth peer name veth1
# ip link set veth1 netns mynamespace
```

The first command sets up a pair of virtual Ethernet devices that are connected. Packets sent to *veth0* will be received by *veth1* and vice versa. The second command assigns *veth1* to the *mynamespace* namespace.

Veth stands for Virtual ETHernet which is a simple tunnel driver that works at the link layer and works like a pair of Ethernet devices interconnected with each other. *Veth* devices have to be created in pairs Don't confuse a Linux *veth* device with Cisco's *vEth* (Virtual Ethernet) devices. Mainly *veth* is used to communicate between network namespaces but it can be used bridge networking segments. Typically, you will have to load the *veth* kernel module first for virtual Ethernet to work.

Naturally, you can configure and assign IP addresses to *veth* devices:

```
# ip netns exec mynamespace ifconfig veth1 10.0.0.1/24 up
# ifconfig veth0 10.0.0.2/24 up
# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.187 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.062 ms
^C
--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.062/0.124/0.187/0.063 ms

# ip netns exec mynamespace route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 veth1

# ip link
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether 00:0c:29:cb:9e:30 brd ff:ff:ff:ff:ff:ff
4: veth0:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether b2:b4:64:be:5f:82 brd ff:ff:ff:ff:ff:ff

# ip netns exec mynamespace ip link
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
3: veth1:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether 86:be:f9:c7:0e:87 brd ff:ff:ff:ff:ff:ff
```


You can move a physical interface to a network namespace:

```
# ip link set ens33 netns mynamespace
# ip netns exec mynamespace ip link
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33:  mtu 1500 qdisc noop state DOWN mode DEFAULT qlen 1000
    link/ether 00:0c:29:cb:9e:30 brd ff:ff:ff:ff:ff:ff
3: veth1:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether 86:be:f9:c7:0e:87 brd ff:ff:ff:ff:ff:ff
# ip link
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4: veth0:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether b2:b4:64:be:5f:82 brd ff:ff:ff:ff:ff:ff
```

And here is one way of moving the physical device back to the root namespace:

```
# ip netns exec mynamespace ip link set ens33 netns 1
# ip link
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether 00:0c:29:cb:9e:30 brd ff:ff:ff:ff:ff:ff
4: veth0:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether b2:b4:64:be:5f:82 brd ff:ff:ff:ff:ff:ff
#
```

There probably is a more elegant way of doing this but so far I have not come across one.

Can we use a VLAN with a network namespace? Yes, we can, as the following example demonstrates:

```
# ip link add link ens33 name myvlan type vlan id 100

# ip link
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether 00:0c:29:cb:9e:30 brd ff:ff:ff:ff:ff:ff
4: veth0:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether b2:b4:64:be:5f:82 brd ff:ff:ff:ff:ff:ff
5: myvlan@ens33:  mtu 1500 qdisc noqueue state UP mode DEFAULT
    link/ether 00:0c:29:cb:9e:30 brd ff:ff:ff:ff:ff:ff

# ip link set ens33 netns mynamespace

# ip link
1: lo:  mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
4: veth0:  mtu 1500 qdisc pfifo_fast state UP mode DEFAULT qlen 1000
    link/ether b2:b4:64:be:5f:82 brd ff:ff:ff:ff:ff:ff
5: myvlan@if2:  mtu 1500 qdisc noqueue state DOWN mode DEFAULT
    link/ether 00:0c:29:cb:9e:30 brd ff:ff:ff:ff:ff:ff
#
```

Network namespaces are an important underpinning for Linux containers and are fully supported in RHEL7. You should take the time to understand them.

