Introducing Linux Network Namespaces

4 September 2013

In this post, I'm going to introduce you to the concept of Linux network namespaces. While it might seem a bit esoteric right now, trust me that there is a reason why I'm introducing you to network namespaces—if you, like me, are on a journey to better understand OpenStack, you'll almost certainly run into network namespaces again.

So what are network namespaces? Generally speaking, an installation of Linux shares a single set of network interfaces and routing table entries. You can modify the routing table entries using policy routing (here's an introduction I wrote and here's a write-up on a potential use case for policy routing), but that doesn't fundamentally change the fact that the set of network interfaces and routing tables/entries are shared across the entire OS. Network namespaces change that fundamental assumption. With network namespaces, you can have different and separate instances of network interfaces and routing tables that operate independent of each other.

This concept is probably best illustrated through some examples. Along the way, I'll introduce a few new ideas as well. First, though, I need to provide some assumptions.

Assumptions

Throughout these examples, I'm using Ubuntu Server 12.04.3 LTS. Please note that support for network namespaces varies between Linux distributions; Ubuntu supports them but Red Hat doesn't. (I'm not sure about Fedora. If you know, speak up in the comments.) If you're thinking about using network namespaces, be sure your Linux distribution includes support.

Further, I'll assume that you're either running as root, or that you will prepend `sudo` to the commands listed here as necessary.

Creating and Listing Network Namespaces

Creating a network namespace is actually quite easy. Just use this command:

```
ip netns add <new namespace name>
```

For example, let's say you wanted to create a namespace called "blue". You'd use this command:

```
ip netns add blue
```

To verify that the network namespace has been created, use this command:

```
ip netns list
```

You should see your network namespace listed there, ready for you to use.

Assigning Interfaces to Network Namespaces

Creating the network namespace is only the beginning; the next part is to assign interfaces to the namespaces, and then configure those interfaces for network connectivity. One thing that threw me off early in my exploration of network namespaces was that you couldn't assign physical interfaces to a namespace. How in the world were you supposed to use them, then?

It turns out you can only assign virtual Ethernet (veth) interfaces to a network namespace. Virtual Ethernet interfaces are an interesting construct; they always come in pairs, and they are connected like a tube—whatever comes in one veth interface will come out the other peer

veth interface. As a result, you can use veth interfaces to connect a network namespace to the outside world via the "default" or "global" namespace where physical interfaces exist.

Let's see how that's done. First, you'd create the veth pair:

```
ip link add veth0 type veth peer name veth1
```

I found a few sites that repeated this command to create `veth1` and link it to `veth0`, but my tests showed that both interfaces were created and linked automatically using this command listed above. Naturally, you could substitute other names for `veth0` and `veth1`, if you wanted.

You can verify that the veth pair was created using this command:

```
ip link list
```

You should see a pair of veth interfaces (using the names you assigned in the command above) listed there. Right now, they both belong to the "default" or "global" namespace, along with the physical interfaces.

Let's say that you want to connect the global namespace to the blue namespace. To do that, you'll need to move one of the veth interfaces to the blue namespace using this command:

```
ip link set veth1 netns blue
```

If you then run the `ip link list` command again, you'll see that the veth1 interface has disappeared from the list. It's now in the blue namespace, so to see it you'd need to run this command:

```
ip netns exec blue ip link list
```

Whoa! That's a bit of a complicated command. Let's break it down:

- The first part, `ip netns exec`, is how you execute commands in a different network namespace.
- Next is the specific namespace in which the command should be run (in this case, the blue namespace).
- Finally, you have the actual command to be executed in the remote namespace. In this case, you want to see the interfaces in the blue namespace, so you run `ip link list`.

When you run that command, you should see a loopback interface and the veth1 interface you moved over earlier.

Configuring Interfaces in Network Namespaces

Now that veth1 has been moved to the blue namespace, we need to actually configure that interface. Once again, we'll use the `ip netns exec` command, this time to configure the veth1 interface in the blue namespace:

```
ip netns exec blue ifconfig veth1 10.1.1.1/24 up
```

As before, the format this command follows is:

```
ip netns exec <network namespace> <command to run
against that namespace>
```

In this case, you're using `ifconfig` to assign an IP address to the veth1 interface and bring that interface up. (Note: you could use the `ip addr`, `ip route`, and `ip link` commands to accomplish the same thing.)

Once the veth1 interface is up, you can verify that the network configuration of the blue namespace is completely separate by just using a few different commands. For example, let's assume that your "global"

namespace has physical interfaces in the 172.16.1.0/24 range, and your veth1 interface is in a separate namespace and assigned something from the 10.1.1.0/24 range. You could verify how network namespaces keep the network configuration separate using these commands:

- `ip addr list` in the global namespace will not show *any* 10.1.1.0/24-related interfaces or addresses.
- `ip netns exec blue ip addr list` will show *only* the 10.1.1.0/24-related interfaces and addresses, and will not show any interfaces or addresses from the global namespace.
- Similarly, `ip route list` in each namespace will show different routing table entries, including different default gateways.

Connecting Network Namespaces to the Physical Network

This part of it threw me for a while. I can't really explain why, but it did. Once I'd figured it out, it was obvious. To connect a network namespace to the physical network, *just use a bridge.* In my case, I used an Open vSwitch (OVS) bridge, but a standard Linux bridge would work as well. Place one or more physical interfaces as well as one of the veth interfaces in the bridge, and—bam!—there you go. Naturally, if you had different namespaces, you'd probably want/need to connect them to different physical networks or different VLANs on the physical network.

So there you go—an introduction to Linux network namespaces. It's quite likely I'll build on this content later, so while it seems a bit obscure right now just hang on to this knowledge. In the meantime, if you have questions, clarifications, or other information worth sharing with other readers, please feel free to speak up in the comments.

Tags: CLI · Linux · Networking

Previous Post: Technology Short Take #35

Next Post: Vendor Meetings at VMworld 2013

Be social and share this post!

Recent Posts

Talking Containers, Virtualization, and Interop 16 Feb 2016

Technology Short Take #61 12 Feb 2016

Using KVM with Libvirt and macvtap Interfaces 09 Feb 2016