# Fixing Docker an VPN IP A ress Conflicts

*by Nate Lampton // August 15, 2019*

### YOU MIGHT ALSO LIKE

> *Creating Default Docker Networks*
> *(https://github.com/docker/compose/issues/4336)*

Occasionally when accessing a client network, I encounter a situation where certain servers are not accessible despite everyone else on the team being able to access the same domain, either by HTTP or SSH. It turns out a frequent culprit of this problem is Docker and it's networking mechanisms.

When you start up Docker, it appropriates some IP addresses for it's own usage. These are usually in the local networking space, which include the following:

> 10.0.0.0 to 10.255.255.255
>
> 172.16.0.0 to 172.31.255.255
>
> 192.168.0.0 to 192.168.255.255

## Discovering Conflicts

You can easily find which IP addresses are being used by your local network or VPN by using `route -n`. Connect to your VPN *first* then run this command.

```
$ route -n

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG    600    0        0
wlp2s0
128.2.5.132     192.168.1.1     255.255.255.255 UGH   600    0        0
wlp2s0
172.16.0.0      0.0.0.0         255.255.0.0     U     50     0        0
vpn0
172.18.6.0      0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.18.11.0     0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.19.0.0      0.0.0.0         255.255.0.0     U     50     0        0
vpn0
172.19.238.0    0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.19.247.0    0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.19.249.0    0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.20.0.0      0.0.0.0         255.255.0.0     U     50     0        0
vpn0
172.20.40.0     0.0.0.0         255.255.254.0   U     50     0        0
vpn0
172.20.42.0     0.0.0.0         255.255.254.0   U     50     0        0
vpn0
172.20.45.0     0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.20.46.0     0.0.0.0         255.255.255.0   U     50     0        0
vpn0
```

```
172.21.0.0      0.0.0.0         255.255.0.0     U     50     0        0
vpn0
172.22.8.0      0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.22.25.0     0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.22.114.0    0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.22.115.0    0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.24.2.0      0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.24.238.0    0.0.0.0         255.255.255.0   U     50     0        0
vpn0
172.29.48.0     0.0.0.0         255.255.240.0   U     50     0        0
vpn0
172.29.80.0     0.0.0.0         255.255.240.0   U     50     0        0
vpn0
```

Looking at this above data, we can derive that the IP address between the ranges of 172.16.*x*.*x* and 172.29.x.*x* are not safe for docker to use.

To determine what IP addresses docker itself is using, we can use the `ip addr` command to see what addresses the networking bridges claim.

Prior to starting docker and any containers:

```
$ ip addr

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default 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: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default qlen 1000
    link/ether 9c:b6:d0:92:d5:b1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.245/24 brd 192.168.1.255 scope global dynamic
noprefixroute wlp2s0
       valid_lft 85363sec preferred_lft 85363sec
    inet6 fe80::69fe:7762:4ecd:d5ae/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: enx8cae4cf13353: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
fq_codel state DOWN group default qlen 1000
    link/ether 8c:ae:4c:f1:33:53 brd ff:ff:ff:ff:ff:ff
4: vpn0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1300 qdisc fq_codel
state UP group default qlen 500
    link/none
    inet 172.31.235.27/16 brd 172.31.255.255 scope global noprefixroute
vpn0
       valid_lft forever preferred_lft forever
    inet6 fe80::5b9c:a58c:5b5a:6b90/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
```

And then after starting docker:

```
$ ip addr

ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default 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: wlp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default qlen 1000
    link/ether 9c:b6:d0:92:d5:b1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.245/24 brd 192.168.1.255 scope global dynamic
noprefixroute wlp2s0
       valid_lft 85977sec preferred_lft 85977sec
    inet6 fe80::69fe:7762:4ecd:d5ae/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: enx8cae4cf13353: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
fq_codel state DOWN group default qlen 1000
    link/ether 8c:ae:4c:f1:33:53 brd ff:ff:ff:ff:ff:ff
4: vpn0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1300 qdisc fq_codel
state UP group default qlen 500
    link/none
    inet 172.31.235.27/16 brd 172.31.255.255 scope global noprefixroute
vpn0
       valid_lft forever preferred_lft forever
    inet6 fe80::5b9c:a58c:5b5a:6b90/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
5: br-05743ccfd659: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN group default
    link/ether 02:42:d8:4d:41:60 brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.1/16 brd 172.20.255.255 scope global br-05743ccfd659
```

```
       valid_lft forever preferred_lft forever
6: br-08e37aab0021: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN group default
    link/ether 02:42:40:a3:a0:63 brd ff:ff:ff:ff:ff:ff
    inet 172.21.0.1/16 brd 172.21.255.255 scope global br-08e37aab0021
       valid_lft forever preferred_lft forever
8: br-70d04f7b2a8c: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN group default
    link/ether 02:42:35:bb:bc:eb brd ff:ff:ff:ff:ff:ff
    inet 172.19.0.1/16 brd 172.19.255.255 scope global br-70d04f7b2a8c
       valid_lft forever preferred_lft forever
9: br-86768d2533cf: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc
noqueue state DOWN group default
    link/ether 02:42:71:49:8e:d7 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-86768d2533cf
       valid_lft forever preferred_lft forever
10: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN group default
    link/ether 02:42:a0:61:83:8f brd ff:ff:ff:ff:ff:ff
    inet 172.240.0.1/24 brd 172.240.0.255 scope global docker0
       valid_lft forever preferred_lft forever
```

We can see from the above that Docker has claimed 172.18.0.1/16 - 172.21.0.1/16. Which will make it so that routes specified by the VPN or local network will not work.

## How to fix this?

First list your current docker networks.

```
$ docker network list
NETWORK ID NAME DRIVER SCOPE
28881c0a72ad cmu_default bridge local
b736a4c00275 host host local
e87ba6af1530 lando_bridge_network bridge local
7d9a9e0a3797 landoproxyhyperion5000gandalfedition_edge bridge
local
4601ca10be74 none null local
```

Those "bridge" entries map to the "br-" prefixed entries seen in the `ip addr` listing.

You can determine which networks are claiming which subnets by using `docker network inspect 28881c0a72ad`. In the JSON output you'll see something similar to this:

```
"Subnet": "172.18.0.0/24",
"Gateway": "172.18.0.1"
```

If you have set up these bridges yourself simply removing the networks and recreating them with a different subnet may be sufficient:

```
docker network rm 05743ccfd659
docker network create --driver=bridge --subnet=192.168.0.0/16 br0
```

Better yet, it's a good idea to prevent docker from automatically claiming subnets that are in conflict with your local network. To fix this you may modify the default daemon.json file used by docker.

```
$ vi /etc/docker/daemon.json
{
  "default-address-pools" : [
    {
      "base" : "172.240.0.0/16",
      "size" : 24
    }
  ]
}
```

Then newly created networks will automatically fit within the specified "base" and "size" parameters.

In my situation, I was using Lando, a docker management tool. So after removing each network and updating the configuration daemon.json configuration file, I had it regenerate all the networks from scratch with:

```
$ lando rebuild
```

Now all my networks are free from conflicts.

Note that individual projects can already specify different subnets in their docker-compose.yml or .lando.yml files. However, with different users needing different networks, control at that level may not be desirable.

More information about setting the default networks can be found in this Docker compose issue: https://github.com/docker/compose/issues/4336 (https://github.com/docker/compose/issues/4336) . Happy networking!

---

**Nate Lam ton**

Nate Lampton is a leader in Open Source with over a decade of contributions to the Drupal project. He joined Lullabot in 2006.