

# wireguard

- [Client Config notes](#)
- [Docker-Compose with Mullvad Wireguard & arbitrary service](#)
- [Troubleshooting](#)
  - [IPTables](#)
- [Wireguard Container + Server](#)

# Client Config notes

Ran this on all the hosts in the cluster; `sysctl -q net.ipv4.conf.all.src_valid_mark=1`

Needed the privileged & net\_admin for tunnel binding

```
- image: linuxserver/wireguard
securityContext:
  privileged: true
  capabilities:
    add: ["NET_ADMIN", "SYS_MODULE"]
env:
- name: PUID
  value: "1000"
- name: PGID
  value: "1000"
name: wireguard
volumeMounts:
- name: config-wg
  mountPath: /config
- name: modules
  mountPath: /lib/modules
```

Spent a lot of time troubleshooting intermittent DNS, this was because of overzealous iptables kill-switches from mullvad blocking the upstream dns server when the cluster DNS didn't have the off hand response, as well as blocking ICMP and other local networks which prevented side-cars from having access from the local net (like qbittorrent over 8080 on 192.168 network)

My final client configuration:

```
[Interface]
PrivateKey = <mullvad provided key>
Address = <mullvad provided IP>/32
```

DNS = <mullvad DNS server>

PostUp = ip route add 192.168.0.0/16 via 169.254.1.1

PreDown = ip route del 192.168.0.0/16 via 169.254.1.1

[Peer]

PublicKey = <mullvad provided key>

AllowedIPs = 0.0.0.0/0 #Actually a take-over ip list

Endpoint = <mullvad server IP>:51820

# Docker-Compose with Mullvad Wireguard & arbitrary service

I was able to make this work really easily in native Kubernetes pods, but lots of folks had been asking questions about getting Wireguard connected to an arbitrary service properly and safely that may not have the means to use that infrastructure. Below are my notes on making that dream a reality with only compose and a few minutes of trial and error.

This compose shows wireguard + qbittorrent with some useful notes in-line. The crux of it though is as follows:

1. Move the exposed ports off the qbittorrent service definition, and into the wireguard definition
2. Add `network_mode: "service:wireguard"` to force the containers to use the same interfaces.

```
version: "3.7"
services:
  wireguard:
    image: linuxserver/wireguard
    container_name: wireguard
    cap_add:
      - NET_ADMIN
      - SYS_MODULE
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Europe/London
    volumes:
      - /appdata/config/wireguard-test/wg:/config
      - /lib/modules:/lib/modules
    ports:
```

- 6881:6881
- 6881:6881/udp
- 8088:8088

sysctls:

- net.ipv4.conf.all.src\_valid\_mark=1

restart: unless-stopped

qbittorrent:

image: linuxserver/qbittorrent

container\_name: qbittorrent

environment:

- PUID=1000
- PGID=1000
- TZ=Europe/London
- UMASK\_SET=022

#Remember to make this the same port as the exposed port

- WEBUI\_PORT=8088

volumes:

- /appdata/config/wireguard-test/qbt:/config
- /appdata/downloads:/downloads

#"ports" moved to wireguard config

restart: unless-stopped

#use the wireguard interfaces instead

network\_mode: "service:wireguard"

In the wireguard `wg0.conf` configuration, you must add a route back to your host network **only if you want to access things** like webUIs from your host. If everything's in the same network, you can just leave this headless, too.

```
PostUp = ip route add 192.168.0.0/16 via $(ip route | grep default | awk '{print $3}')
```

[Interface]

PrivateKey = <MULLVAD KEY>

Address = <MULLVAD ADDRESS>

DNS = <MULLVAD DNS>

PostUp = DROUTE=\$(ip route | grep default | awk '{print \$3}'); HOMENET=192.168.0.0/16;

HOMENET2=10.0.0.0/8; HOMENET3=172.16.0.0/12; ip route add \$HOMENET3 via \$DROUTE; ip route add

\$HOMENET2 via \$DROUTE; ip route add \$HOMENET via \$DROUTE; iptables -I OUTPUT -d \$HOMENET -j

ACCEPT; iptables -A OUTPUT -d \$HOMENET2 -j ACCEPT; iptables -A OUTPUT -d \$HOMENET3 -j ACCEPT; iptables -

A OUTPUT ! -o %i -m mark ! --mark \$(wg show %i fwmark) -m addrtype ! --dst-type LOCAL -j REJECT

PreDown = HOMENET=192.168.0.0/16; HOMENET2=10.0.0.0/8; HOMENET3=172.16.0.0/12; ip route del

```
$HOMENET3 via $DROUTE;ip route del $HOMENET2 via $DROUTE; ip route del $HOMENET via $DROUTE;  
iptables -D OUTPUT ! -o %i -m mark ! --mark $(wg show %i fwmark) -m addrtype ! --dst-type LOCAL -j REJECT;  
iptables -D OUTPUT -d $HOMENET -j ACCEPT; iptables -D OUTPUT -d $HOMENET2 -j ACCEPT; iptables -D OUTPUT  
-d $HOMENET3 -j ACCEPT
```

[Peer]

PublicKey = jHxY2OKpxjqAwWH4r1Pb2K6xDUDt087ivxpM1KpE0Ec=

AllowedIPs = 0.0.0.0/0

Endpoint = <MULLVAD SERVER>:51820

Pretty simple, right? Here's the results of what you came here to see.

```
root@f316f4f274fb:/# curl https://am.i.mullvad.net/connected
```

You are connected to Mullvad (server us32-wireguard). Your IP address is 206.217.xxx.xxx

If you're curious about the nitty gritty, here's the output from each containers interfaces & routes to give an illustration on how this works as if it were on the same host instead of dedicated network stacks:

From Wireguard

```
root@347666d9f127:/# ps -ef  
UID      PID  PPID  C  STIME TTY      TIME CMD  
root      1    0  0 16:52 ?        00:00:00 s6-svscan -t0 /var/run/s6/services  
root     32    1  0 16:52 ?        00:00:00 s6-supervise s6-fdholderd  
root    265    1  0 16:52 ?        00:00:00 s6-supervise coredns  
root    266    1  0 16:52 ?        00:00:00 s6-supervise wireguard  
root    268   266  0 16:52 ?        00:00:00 bash ./run  
root    270   265  0 16:52 ?        00:00:00 /app/coredns -dns.port=53  
root    357   268  0 16:52 ?        00:00:00 sleep infinity  
root    358    0  0 16:59 pts/0    00:00:00 bash  
root    378   358  0 17:00 pts/0    00:00:00 ps -ef
```

```
root@347666d9f127:/# ip route  
default via 172.24.0.1 dev eth0  
172.24.0.0/16 dev eth0 proto kernel scope link src 172.24.0.2  
192.168.0.0/16 via 172.24.0.1 dev eth0
```

```
root@347666d9f127:/# ip address
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
3: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen
1000
link/none
inet 10.67.xxx.xx/32 scope global wg0
valid_lft forever preferred_lft forever
151: eth0@if152: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
link/ether 02:42:ac:18:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
inet 172.24.0.2/16 brd 172.24.255.255 scope global eth0
valid_lft forever preferred_lft forever
```

```
root@347666d9f127:/# iptables-save
```

# Generated by iptables-save v1.6.1 on Sat Aug 8 14:48:00 2020

```
*filter
:INPUT ACCEPT [16:2307]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [17:1615]
-A OUTPUT -d 192.168.0.0/16 -j ACCEPT
-A OUTPUT -d 10.0.0.0/8 -j ACCEPT
-A OUTPUT -d 172.16.0.0/12 -j ACCEPT
-A OUTPUT ! -o wg0 -m mark ! --mark 0xca6c -m addrtype ! --dst-type LOCAL -j REJECT --reject-with icmp-port-
unreachable
COMMIT
```

Completed on Sat Aug 8  
14:48:00 2020

Generated by iptables-save  
v1.6.1 on Sat Aug 8 14:48:00  
2020

```
*mangle
:PREROUTING ACCEPT [16:2307]
:INPUT ACCEPT [16:2307]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [19:1729]
:POSTROUTING ACCEPT [19:1729]
-A PREROUTING -p udp -m comment --comment "wg-quick(8) rule for wg0" -j CONNMARK --restore-mark --nfmask
0xffffffff --ctmask 0xffffffff
-A POSTROUTING -p udp -m mark --mark 0xca6c -m comment --comment "wg-quick(8) rule for wg0" -j
CONNMARK --save-mark --nfmask 0xffffffff --ctmask 0xffffffff
COMMIT
```

Completed on Sat Aug 8  
14:48:00 2020



# Generated by iptables-save v1.6.1 on Sat Aug 8 14:48:00 2020

```
*raw
:PREROUTING ACCEPT [16:2307]
:OUTPUT ACCEPT [19:1729]
-A PREROUTING -d 10.67.xxx.xxx/32 ! -i wg0 -m addrtype ! --src-type LOCAL -m comment --comment "wg-quick(8) rule for wg0" -j DROP
COMMIT
```

## Completed on Sat Aug 8 14:48:00 2020

From qbittorrent

```
root@347666d9f127:/# ps -ef
UID      PID  PPID  C STIME TTY      TIME CMD
root      1    0  0 16:52 ?        00:00:00 s6-svscan -t0 /var/run/s6/services
root     32    1  0 16:52 ?        00:00:00 s6-supervise s6-fdholderd
root    250    1  0 16:52 ?        00:00:00 s6-supervise qbittorrent
abc     252  250  0 16:52 ?        00:00:02 /usr/bin/qbittorrent-nox --webui-port=8088
root    276    0  0 17:00 pts/0    00:00:00 bash
root    669  276  0 17:02 pts/0    00:00:00 ps -ef

root@347666d9f127:/# ip route
default via 172.24.0.1 dev eth0
172.24.0.0/16 dev eth0 proto kernel scope link src 172.24.0.2
192.168.0.0/16 via 172.24.0.1 dev eth0
```

```
root@347666d9f127:/# ip address
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
3: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen
1000
link/none
inet 10.67.xx.xx scope global wg0
valid_lft forever preferred_lft forever
151: eth0@if152: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
link/ether 02:42:ac:18:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
inet 172.24.0.2/16 brd 172.24.255.255 scope global eth0
valid_lft forever preferred_lft forever
```

Sources:

<https://nbsoftsolutions.com/blog/routing-select-docker-containers-through-wireguard-vpn>

# Troubleshooting

# IPTables

One user claimed that when they enabled wireguard via a ``docker-compose up`` that all containers lost internet access.

TCPDumps showed that NAT from the bridges to the external interface had been lost at some point, indicating that the iptables may have been dropped or altered in such a way that the docker bridges could no longer properly NAT traffic.

One clue that was given, but missed several times. In the ``iptables-save`` before wireguard came on which broke connectivity showed no mention of legacy tables. After wireguard was started, ``iptables-legacy-save`` was reportedly needed to see all the rules. Following this instruction showed an empty ruleset, a life without nat!

The user simply switched off nftables to "legacy" mode via the openmediavault UI, but presumably a newer debian user could also just run ``update-alternatives --set iptables /usr/sbin/iptables-legacy`` to get the same effect.

No problem:

```
# Completed on Sun Aug 9 21:51:20 2020
root@DK:~#
```

Problem:

```
# Warning: iptables-legacy tables present, use iptables-legacy-save to see them
root@DK:/srv/dev-disk-by-label-HC2/DockerCompose/wireguard#
```

# Wireguard Container + Server

SawToday at 9:18 AM

Evening, maybe someone can help me. I am really not a network professional. I installed the Wireguard container on my rootserver, and on my server here in my own lan. The connection works too, but I can't get the route right, so the docker host can access the lan of the VPN.

OxyTJToday at 9:25 AM

@Saw what I had to do for this issue is first run the following to get your subnet:

```
ip route | awk '!/ (docker0|br-)/ && /src/ {print $1}'
```

After connecting the VPN, I run the following to add the routes, where SUBNET is the ip you got from the previous command:

```
ip route add ${SUBNET} via $(ip route | grep default | awk '{print $3}')
```

```
iptables -A FORWARD -s ${SUBNET} -j ACCEPT
```

```
iptables -A FORWARD -d ${SUBNET} -j ACCEPT
```

```
iptables -A INPUT -s ${SUBNET} -j ACCEPT
```

```
iptables -A OUTPUT -d ${SUBNET} -j ACCEPT
```

Lastly, make sure your exposed ports for other containers are forwarded in the VPN container.

One thing to note, in case you have the same problem, it appears the INPUT and OUTPUT get wiped from the iptables every so often, or whenever the VPN connects. In case you have the same.