CONTENTS

// TUTORIAL //

How To Set Up WireGuard on Ubuntu 20.04

Published on August 26, 2021

Security Ubuntu VPN Networking IPv6 Ubuntu 20.04

Jamon Camisso

Not using Ubuntu 20.04?
Choose a different version or distribution.

Ubuntu 20.04

Introduction

WireGuard is a lightweight Virtual Private Network (VPN) that supports IPv4 and IPv6 connections. A VPN allows you to traverse untrusted networks as if you were on a

private network. It gives you the freedom to access the internet safely and securely from your smartphone or laptop when connected to an untrusted network, like the WiFi at a hotel or coffee shop.

WireGuard's encryption relies on public and private keys for peers to establish an encrypted tunnel between themselves. Each version of WireGuard uses a specific cryptographic cipher suite to ensure simplicity, security, and compatibility with peers.

In comparison, other VPN software such as OpenVPN and IPSec use Transport Layer Security (TLS) and certificates to authenticate and establish encrypted tunnels between systems. Different versions of TLS include support for hundreds of different cryptographic suites and algorithms, and while this allows for great flexibility to support different clients, it also makes configuring a VPN that uses TLS more time consuming, complex, and error prone.

In this tutorial, you will set up WireGuard on an Ubuntu 20.04 server, and then configure another machine to connect to it as a peer using both IPv4 and IPv6 connections (commonly referred to as a *dual stack* connection). You'll also learn how to route the peer's Internet traffic through the WireGuard server in a gateway configuration, in addition to using the VPN for an encrypted peer-to-peer tunnel.

For the purposes of this tutorial, we'll configure another Ubuntu 20.04 system as the peer (also referred to as client) to the WireGuard Server. Subsequent tutorials in this series will explain how to install and run WireGuard on Windows, macOS, Android, and iOS systems and devices.

Note: If you plan to set up WireGuard on a DigitalOcean Droplet, be aware that we, like many hosting providers, charge for bandwidth overages. For this reason, please be mindful of how much traffic your server is handling. See this page for more info.

Prerequisites

To follow this tutorial, you will need:

- One Ubuntu 20.04 server with a sudo non-root user and a firewall enabled. To set this up, you can follow our Initial Server Setup with Ubuntu 20.04 tutorial. We will refer to this as the **WireGuard Server** throughout this guide.
- You'll need a client machine that you will use to connect to your WireGuard Server. In this tutorial we'll refer to this machine as the **WireGuard Peer**. For the purposes of this tutorial, it's recommended that you use your local machine as the WireGuard

Peer, but you can use remote servers, or mobile phones as clients if you prefer. If you are using a remote system, be sure to follow all of the optional sections later in this tutorial or you may lock yourself out of the system.

- To use WireGuard with IPv6, you will also need to ensure that your server is configured to support that type of traffic. If you would like to enable IPv6 support with WireGuard and are using a DigitalOcean Droplet, please refer to this documentation page How to Enable IPv6 on Droplets. You can add IPv6 support when you create a Droplet, or afterwards using the instructions on that page.

Step 1 – Installing WireGuard and Generating a Key Pair

The first step in this tutorial is to install WireGuard on your server. To start off, update your WireGuard Server's package index and install WireGuard using the following commands. You may be prompted to provide your sudo user's password if this is the first time you're using `sudo` in this session:

Copy

```
$ sudo apt update
$ sudo apt install wireguard
```

Now that you have WireGuard installed, the next step is to generate a private and public keypair for the server. You'll use the built-in `wg genkey` and `wg pubkey` commands to create the keys, and then add the private key to WireGuard's configuration file.

You will also need to change the permissions on the key that you just created using the `chmod` command, since by default the file is readable by any user on your server.

Create the private key for WireGuard and change its permissions using the following commands:

Copy

```
$ wg genkey | sudo tee /etc/wireguard/private.key
$ sudo chmod go= /etc/wireguard/private.key
```

The `sudo chmod go=...` command removes any permissions on the file for users and groups other than the root user to ensure that only it can access the private key.

You should receive a single line of `base64` encoded output, which is the private key. A copy of the output is also stored in the `/etc/wireguard/private.key` file for future

reference by the `tee` portion of the command. Carefully make a note of the private key that is output since you'll need to add it to WireGuard's configuration file later in this section.

The next step is to create the corresponding public key, which is derived from the private key. Use the following command to create the public key file:

Copy

```
$ sudo cat /etc/wireguard/private.key | wg pubkey | sudo tee /etc/wireguard/
```

This command consists of three individual commands that are chained together using the `|` (pipe) operator:

- `sudo cat /etc/wireguard/private.key`: this command reads the private key file and outputs it to the standard output stream.
- `wg pubkey`: the second command takes the output from the first command as its standard input and processes it to generate a public key.
- `sudo tee /etc/wireguard/public.key`: the final command takes the output of the public key generation command and redirects it into the file named `/etc/wireguard/public.key`.

When you run the command you will again receive a single line of `base64` encoded output, which is the public key for your WireGuard Server. Copy it somewhere for reference, since you will need to distribute the public key to any peer that connects to the server.

Step 2 – Choosing IPv4 and IPv6 Addresses

In the previous section you installed WireGuard and generated a key pair that will be used to encrypt traffic to and from the server. In this section, you will create a configuration file for the server, and set up WireGuard to start up automatically when you server reboots. You will also define private IPv4 and IPv6 addresses to use with your WireGuard Server and peers.

If you plan to use both IPv4 and IPv6 addresses then follow both of these sections. Otherwise, follow the instructions in the appropriate section for your VPN's network needs.

Step 2(a) – Choosing an IPv4 Range

If you are using your WireGuard server with IPv4 peers, the server needs a range of private IPv4 addresses to use for clients, and for its tunnel interface. You can choose

any range of IP addresses from the following reserved blocks of addresses (if you would like to learn more about how these blocks are allocated visit the RFC 1918 specification):

- `10.0.0.0` to `10.255.255.255` (10/8 prefix)
- `172.16.0.0` to `172.31.255.255` (172.16/12 prefix)
- `192.168.0.0` to `192.168.255.255` (192.168/16 prefix)

For the purposes of this tutorial we'll use `10.8.0.0/24` as a block of IP addresses from the first range of reserved IPs. This range will allow up to 255 different peer connections, and generally should not have overlapping or conflicting addresses with other private IP ranges. Feel free to choose a range of addresses that works with your network configuration if this example range isn't compatible with your networks.

The WireGuard Server will use a single IP address from the range for its private tunnel IPv4 address. We'll use `10.8.0.1/24` here, but any address in the range of `10.8.0.1` to `10.8.0.255` can be used. Make a note of the IP address that you choose if you use something different from `10.8.0.1/24`. You will add this IPv4 address to the configuration file that you define in Step 3 — Creating a WireGuard Server Configuration.

Step 2(b) – Choosing an IPv6 Range

If you are using WireGuard with IPv6, then you will need to generate a unique local IPv6 unicast address prefix based on the algorithm in RFC 4193. The addresses that you use with WireGuard will be associated with a virtual tunnel interface. You will need to complete a few steps to generate a random, unique IPv6 prefix within the reserved `fd00::/8` block of private IPv6 addresses.

According to the RFC, the recommended way to obtain a unique IPv6 prefix is to combine the time of day with a unique identifying value from a system like a serial number or device ID. Those values are then hashed and truncated resulting in a set of bits that can be used as a unique address within the reserved private `fd00::/8` block of IPs.

To get started generating an IPv6 range for your WireGuard Server, collect a 64-bit timestamp using the `date` utility with the following command:

Copy

```
$ date +%s%N
```

You will receive a number like the following, which is the number of seconds (the `%s` in the `date` command), and nanoseconds (the `%N`) since 1970-01-01 00:00:00 UTC combined together:

```
Output
1628101352127592197
```

Record the value somewhere for use later in this section. Next, copy the `machine-id` value for your server from the `/var/lib/dbus/machine-id` file. This identifier is unique to your system and should not change for as long as the server exists.

Copy

```
$ cat /var/lib/dbus/machine-id
```

You will receive output like the following:

```
/var/lib/dbus/machine-id
20086c25853947c7aeee2ca1ea849d7d
```

Now you need to combine the timestamp with the `machine-id` and hash the resulting value using the SHA-1 algorithm. The command will use the following format:

```
printf <timestamp><machine-id> | sha1sum
```

Run the command substituting in your timestamp and machine identity values:

Copy

```
$ printf 1628101352127592197 20086c25853947c7aeee2ca1ea849d7d | sha1sum
```

You will receive a hash value like the following:

```
Output
4f267c51857d6dc93a0bca107bca2f0d86fac3bc -
```

Note that the output of the `sha1sum` command is in hexadecimal, so the output uses two characters to represent a single byte of data. For example `4f` and `26` in the example output are the first two bytes of the hashed data.

The algorithm in the RFC only requires the least significant (trailing) 40 bits, or 5 bytes, of the hashed output. Use the `cut` command to print the last 5 hexadecimal encoded bytes from the hash:

Copy

```
$ printf 4f267c51857d6dc93a0bca107bca2f0d86fac3bc | cut -c 31-
```

The `-c` argument tells the `cut` command to select only a specified set of characters. The `31-` argument tells `cut` to print all the characters from position 31 to the end of the input line.

You should receive output like the following:

```
Output
0d86fac3bc
```

In this example output, the set of bytes is: `0d 86 fa c3 bc` .

Now you can construct your unique IPv6 network prefix by appending the 5 bytes you have generated with the `fd` prefix, separating every **2** bytes with a `:` colon for readability. Because each subnet in your unique prefix can hold a total of 18,446,744,073,709,551,616 possible IPv6 addresses, you can restrict the subnet to a standard size of `/64` for simplicity.

Using the bytes previously generated with the `/64` subnet size the resulting prefix will be the following:

```
Unique Local IPv6 Address Prefix
 fd0d : 86fa : c3bc ::/64
```

This `fd0d:86fa:c3bc::/64` range is what you will use to assign individual IP addresses to your WireGuard tunnel interfaces on the server and peers. To allocate an IP for the server, add a `1` after the final `::` characters. The resulting address will be `fd0d:86fa:c3bc:: 1 /64` . Peers can use any IP in the range, but typically you'll increment the value by one each time you add a peer e.g. `fd0d:86fa:c3bc:: 2 /64` . Make a note of the IP and proceed configuring the WireGuard Server in the next section of this tutorial.

Step 3 – Creating a WireGuard Server Configuration

Before creating your WireGuard Server's configuration, you will need the following pieces of information:

1. Make sure that you have the private key available from Step 1 — Installing WireGuard and Generating a Key Pair.
2. If you are using WireGuard with IPv4, you'll need the IP address that you chose for the server in Step 2(a) — Choosing an IPv4 Range, which in this example is `10.8.0.1/24` .
3. If you are using WireGuard with IPv6, you'll need the IP address for the server that you generated in Step 2(b) — Choosing an IPv6 Range. In this example the IP is `fd0d:86fa:c3bc::1/64` .

Once you have the required private key and IP address(es), create a new configuration file using `nano` or your preferred editor by running the following command:

Copy

```
$ sudo nano /etc/wireguard/wg0.conf
```

Add the following lines to the file, substituting your private key in place of the highlighted `base64_encoded_private_key_goes_here` value, and the IP address(es) on the `Address` line. You can also change the `ListenPort` line if you would like WireGuard to be available on a different port:

```
/etc/wireguard/wg0.conf
[Interface]
PrivateKey =  base64_encoded_private_key_goes_here
Address =  10.8.0.1/24 ,  fd0d:86fa:c3bc::1/64
ListenPort = 51820
SaveConfig = true
```

The `SaveConfig` line ensures that when a WireGuard interface is shutdown, any changes will get saved to the configuration file.

Save and close the `/etc/wireguard/wg0.conf` file. If you are using `nano` , you can do so with `CTRL+X` , then `Y` and `ENTER` to confirm. You now have an initial server configuration that you can build upon depending on how you plan to use your WireGuard VPN server.

Step 4 – Adjusting the WireGuard Server's Network Configuration

If you are using WireGuard to connect a peer to the WireGuard Server in order to access services on the **server only**, then you do not need to complete this section. If you would like to route your WireGuard Peer's Internet traffic through the WireGuard Server then you will need to configure IP forwarding by following this section of the tutorial.

To configure forwarding, open the `/etc/sysctl.conf` file using `nano` or your preferred editor:

Copy

```
$ sudo nano /etc/sysctl.conf
```

If you are using IPv4 with WireGuard, add the following line at the bottom of the file:

/etc/sysctl.conf

```
net.ipv4.ip_forward=1
```

If you are using IPv6 with WireGuard, add this line at the bottom of the file:

/etc/sysctl.conf

```
net.ipv6.conf.all.forwarding=1
```

If you are using both IPv4 and IPv6, ensure that you include both lines. Save and close the file when you are finished.

To read the file and load the new values for your current terminal session, run:

Copy

```
$ sudo sysctl -p
```

```
Output
net.ipv6.conf.all.forwarding = 1
net.ipv4.ip_forward = 1
```

Now your WireGuard Server will be able to forward incoming traffic from the virtual VPN ethernet device to others on the server, and from there to the public Internet. Using this

configuration will allow you to route all web traffic from your WireGuard Peer via your server's IP address, and your client's public IP address will be effectively hidden.

However, before traffic can be routed via your server correctly, you will need to configure some firewall rules. These rules will ensure that traffic to and from your WireGuard Server and Peers flows properly.

Step 5 – Configuring the WireGuard Server's Firewall

In this section you will edit the WireGuard Server's configuration to add firewall rules that will ensure traffic to and from the server and clients is routed correctly. As with the previous section, skip this step if you are only using your WireGuard VPN for a machine to machine connection to access resources that are restricted to your VPN.

To allow WireGuard VPN traffic through the Server's firewall, you'll need to enable masquerading, which is an iptables concept that provides on-the-fly dynamic network address translation (NAT) to correctly route client connections.

First find the public network interface of your WireGuard Server using the `ip route` sub-command:

Copy

```
$ ip route list default
```

The public interface is the string found within this command's output that follows the word "dev". For example, this result shows the interface named `eth0`, which is highlighted below:

```
Output
default via 203.0.113.1 dev  eth0  proto static
```

Note your device's name since you will add it to the `iptables` rules in the next step.

To add firewall rules to your WireGuard Server, open the `/etc/wireguard/wg0.conf` file with `nano` or your preferred editor again.

Copy

```
$ sudo nano /etc/wireguard/wg0.conf
```

At the bottom of the file after the `SaveConfig = true` line, paste the following lines:

```
/etc/wireguard/wg0.conf
. . .
PostUp = ufw route allow in on wg0 out on  eth0
PostUp = iptables -t nat -I POSTROUTING -o  eth0  -j MASQUERADE
PostUp = ip6tables -t nat -I POSTROUTING -o  eth0  -j MASQUERADE
PreDown = ufw route delete allow in on wg0 out on  eth0
PreDown = iptables -t nat -D POSTROUTING -o  eth0  -j MASQUERADE
PreDown = ip6tables -t nat -D POSTROUTING -o  eth0  -j MASQUERADE
```

The `PostUp` lines will run when the WireGuard Server starts the virtual VPN tunnel. In the example here, it will add three `ufw` and `iptables` rules:

- `ufw route allow in on wg0 out on  eth0` - This rule will allow forwarding IPv4 and IPv6 traffic that comes in on the `wg0` VPN interface to the `eth0` network interface on the server. It works in conjunction with the `net.ipv4.ip_forward` and `net.ipv6.conf.all.forwarding` sysctl values that you configured in the previous section.
- `iptables -t nat -I POSTROUTING -o eth0 -j MASQUERADE` - This rule configures masquerading, and rewrites IPv4 traffic that comes in on the `wg0` VPN interface to make it appear like it originates directly from the WireGuard Server's public IPv4 address.
- `ip6tables -t nat -I POSTROUTING -o eth0 -j MASQUERADE` - This rule configures masquerading, and rewrites IPv6 traffic that comes in on the `wg0` VPN interface to make it appear like it originates directly from the WireGuard Server's public IPv6 address.

The `PreDown` rules run when the WireGuard Server stops the virtual VPN tunnel. These rules are the inverse of the `PostUp` rules, and function to undo the forwarding and masquerading rules for the VPN interface when the VPN is stopped.

In both cases, edit the configuration to include or exclude the IPv4 and IPv6 rules that are appropriate for your VPN. For example, if you are just using IPv4, then you can exclude the lines with the `ip6tables` commands.

Conversely, if you are only using IPv6, then edit the configuration to only include the `ip6tables` commands. The `ufw` lines should exist for any combination of IPv4 and IPv6 networks. Save and close the file when you are finished.

The last part of configuring the firewall on your WireGuard Server is to allow traffic to and from the WireGuard UDP port itself. If you did not change the port in the server's `/etc/wireguard/wg0.conf` file, the port that you will open is `51820`. If you chose a different port when editing the configuration be sure to substitute it in the following UFW command.

In case you forgot to open the SSH port when following the prerequisite tutorial, add it here too:

Copy

```
$ sudo ufw allow  51820 / udp
$ sudo ufw allow OpenSSH
```

Note: If you are using a different firewall or have customized your UFW configuration, you may need to add additional firewall rules. For example, if you decide to tunnel all of your network traffic over the VPN connection, you will need to ensure that port `53` traffic is allowed for DNS requests, and ports like `80` and `443` for HTTP and HTTPS traffic respectively. If there are other protocols that you are using over the VPN then you will need to add rules for them as well.

After adding those rules, disable and re-enable UFW to restart it and load the changes from all of the files you've modified:

Copy

```
$ sudo ufw disable
$ sudo ufw enable
```

You can confirm the rules are in place by running the `ufw status` command. Run it, and you should receive output like the following:

Copy

```
$ sudo ufw status
```

```
Output
Status: active

To                         Action      From
--                         ------      ----
51280/udp                  ALLOW       Anywhere
22/tcp                     ALLOW       Anywhere
51280/udp (v6)             ALLOW       Anywhere (v6)
22/tcp (v6)                ALLOW       Anywhere (v6)
```

Your WireGuard Server is now configured to correctly handle the VPN's traffic, including forwarding and masquerading for peers. With the firewall rules in place, you can start the WireGuard service itself to listen for peer connections.

Step 6 – Starting the WireGuard Server

WireGuard can be configured to run as a `systemd` service using its built-in `wg-quick` script. While you could manually use the `wg` command to create the tunnel every time you want to use the VPN, doing so is a manual process that becomes repetitive and error prone. Instead, you can use `systemctl` to manage the tunnel with the help of the `wg-quick` script.

Using a `systemd` service means that you can configure WireGuard to start up at boot so that you can connect to your VPN at any time as long as the server is running. To do this, enable the `wg-quick` service for the `wg0` tunnel that you've defined by adding it to `systemctl`:

Copy

```
$ sudo systemctl enable wg-quick@wg0.service
```

> Notice that the command specifies the name of the tunnel `wg0` device name as a part of the service name. This name maps to the `/etc/wireguard/ wg0 .conf` configuration file. This approach to naming means that you can create as many separate VPN tunnels as you would like using your server.
>
> For example, you could have a tunnel device and name of `prod` and its configuration file would be `/etc/wireguard/prod.conf`. Each tunnel configuration can contain different IPv4, IPv6, and client firewall settings. In this way you can support multiple different peer connections, each with their own unique IP addresses and routing rules.

Now start the service:

Copy

```
$ sudo systemctl start wg-quick@wg0.service
```

Double check that the WireGuard service is active with the following command. You should see `active (running)` in the output:

Copy

```
$ sudo systemctl status wg-quick@wg0.service
```

```
Output
● wg-quick@wg0.service - WireGuard via wg-quick(8) for wg0
     Loaded: loaded (/lib/systemd/system/wg-quick@.service; enabled; vendor p
     Active: active (exited) since Wed 2021-08-25 15:24:14 UTC; 5s ago
       Docs: man:wg-quick(8)
             man:wg(8)
             https://www.wireguard.com/
             https://www.wireguard.com/quickstart/
             https://git.zx2c4.com/wireguard-tools/about/src/man/wg-quick.8
             https://git.zx2c4.com/wireguard-tools/about/src/man/wg.8
    Process: 3245 ExecStart=/usr/bin/wg-quick up wg0 (code=exited, status=0/S
   Main PID: 3245 (code=exited, status=0/SUCCESS)

Aug 25 15:24:14 wg0 wg-quick[3245]: [#] wg setconf wg0 /dev/fd/63
Aug 25 15:24:14 wg0 wg-quick[3245]: [#] ip -4 address add  10.8.0.1/24  dev wg
Aug 25 15:24:14 wg0 wg-quick[3245]: [#] ip -6 address add  fd0d:86fa:c3bc::1/6
Aug 25 15:24:14 wg0 wg-quick[3245]: [#] ip link set mtu 1420 up dev wg0
Aug 25 15:24:14 wg0 wg-quick[3245]: [#] ufw route allow in on wg0 out on eth0
Aug 25 15:24:14 wg0 wg-quick[3279]: Rule added
Aug 25 15:24:14 wg0 wg-quick[3279]: Rule added (v6)
Aug 25 15:24:14 wg0 wg-quick[3245]: [#] iptables -t nat -I POSTROUTING -o eth
Aug 25 15:24:14 wg0 wg-quick[3245]: [#] ip6tables -t nat -I POSTROUTING -o et
Aug 25 15:24:14 wg0 systemd[1]: Finished WireGuard via wg-quick(8) for wg0.
```

The output shows the `ip` commands that are used to create the virtual `wg0` device and assign it the IPv4 and IPv6 addresses that you added to the configuration file. You can use these rules to troubleshoot the tunnel, or with the `wg` command itself if you would like to try manually configuring the VPN interface.

With the server configured and running, the next step is to configure your client machine as a WireGuard Peer and connect to the WireGuard Server.

Step 7 – Configuring a WireGuard Peer

Configuring a WireGuard peer is similar to setting up the WireGuard Server. Once you have the client software installed, you'll generate a public and private key pair, decide on an IP address or addresses for the peer, define a configuration file for the peer, and then start the tunnel using the `wg-quick` script.

You can add as many peers as you like to your VPN by generating a key pair and configuration using the following steps. If you add multiple peers to the VPN be sure to

keep track of their private IP addresses to prevent collisions.

To configure the WireGuard Peer, ensure that you have the WireGuard package installed using the following `apt` commands. On the WireGuard peer run:

Copy

```
$ sudo apt update
$ sudo apt install wireguard
```

Creating the WireGuard Peer's Key Pair

Next, you'll need to generate the key pair on the peer using the same steps as you used on the server. From your local machine or remote server that will serve as peer, proceed and create the private key for the peer using the following commands:

Copy

```
$ wg genkey | sudo tee /etc/wireguard/private.key
$ sudo chmod go= /etc/wireguard/private.key
```

Again you will receive a single line of `base64` encoded output, which is the private key. A copy of the output is also stored in the `/etc/wireguard/private.key`. Carefully make a note of the private key that is output since you'll need to add it to WireGuard's configuration file later in this section.

Next use the following command to create the public key file:

Copy

```
$ sudo cat /etc/wireguard/private.key | wg pubkey | sudo tee /etc/wireguard/
```

You will again receive a single line of `base64` encoded output, which is the public key for your WireGuard Peer. Copy it somewhere for reference, since you will need to distribute the public key to the WireGuard Server in order to establish an encrypted connection.

Creating the WireGuard Peer's Configuration File

Now that you have a key pair, you can create a configuration file for the peer that contains all the information that it needs to establish a connection to the WireGuard Server.

You will need a few pieces of information for the configuration file:

- The `base64` encoded private key that you generated on the peer.
- The IPv4 and IPv6 address ranges that you defined on the WireGuard Server.
- The `base64` encoded public key from the WireGuard Server.
- The public IP address and port number of the WireGuard Server. Usually this will be the IPv4 address, but if your server has an IPv6 address and your client machine has an IPv6 connection to the internet you can use this instead of IPv4.

With all this information at hand, open a new `/etc/wireguard/wg0.conf` file on the WireGuard Peer machine using `nano` or your preferred editor:

Copy

```
$ sudo nano /etc/wireguard/wg0.conf
```

Add the following lines to the file, substituting in the various data into the highlighted sections as required:

```
/etc/wireguard/wg0.conf
[Interface]
PrivateKey =  base64_encoded_peer_private_key_goes_here
Address =  10.8.0.2/24
Address =  fd0d:86fa:c3bc::2/64

[Peer]
PublicKey =  U9uE2kb/nrrzsEU58GD3pKFU3TLYDMCbetIsnV8eeFE=
AllowedIPs =  10.8.0.0/24 ,  fd0d:86fa:c3bc::/64
Endpoint =  203.0.113.1 :51820
```

Notice how the first `Address` line uses an IPv4 address from the `10.8.0.0/24` subnet that you chose earlier. This IP address can be anything in the subnet as long as it is different from the server's IP. Incrementing addresses by 1 each time you add a peer is generally the easiest way to allocate IPs.

Likewise, notice how the second `Address` line uses an IPv6 address from the subnet that you generated earlier, and increments the server's address by one. Again, any IP in the range is valid if you decide to use a different address.

The other notable part of the file is the last `AllowedIPs` line. These two IPv4 and IPv6 ranges instruct the peer to only send traffic over the VPN if the destination system has an IP address in either range. Using the `AllowedIPs` directive, you can restrict the VPN on the peer to only connect to other peers and services on the VPN, or you can

configure the setting to tunnel all traffic over the VPN and use the WireGuard Server as a gateway.

If you are only using IPv4, then omit the trailing `fd0d:86fa:c3bc::/64` range (including the `,` comma). Conversely, if you are only using IPv6, then only include the `fd0d:86fa:c3bc::/64` prefix and leave out the `10.8.0.0/24` IPv4 range.

In both cases, if you would like to send all your peer's traffic over the VPN and use the WireGuard Server as a gateway for all traffic, then you can use `0.0.0.0/0`, which represents the entire IPv4 address space, and `::/0` for the entire IPv6 address space.

(Optional) Configuring a Peer to Route All Traffic Over the Tunnel

If you have opted to route all of the peer's traffic over the tunnel using the `0.0.0.0/0` or `::/0` routes and the peer is a remote system, then you will need to complete the steps in this section. If your peer is a local system then it is best to skip this section.

For remote peers that you access via SSH or some other protocol using a public IP address, you will need to add some extra rules to the peer's `wg0.conf` file. These rules will ensure that you can still connect to the system from outside of the tunnel when it is connected. Otherwise, when the tunnel is established, all traffic that would normally be handled on the public network interface will not be routed correctly to bypass the `wg0` tunnel interface, leading to an inaccessible remote system.

First, you'll need to determine the IP address that the system uses as its default gateway. Run the following `ip route` command:

Copy

```
$ ip route list table main default
```

You will receive output like the following:

```
Output
default via  203.0.113.1  dev  eth0  proto static
```

Note the gateway's highlighted IP address `203.0.113.1` for later use, and device `eth0`. Your device name may be different. If so, substitute it in place of `eth0` in the following commands.

Next find the public IP for the system by examining the device with the `ip address show` command:

Copy

```
$ ip -brief address show  eth0
```

You will receive output like the following:

```
Output
eth0             UP             203.0.113.5/20  10.20.30.40/16 2604:a880:400:
```

In this example output, the highlighted `203.0.113.5` IP (without the trailing `/20`) is the public address that is assigned to the `eth0` device that you'll need to add to the WireGuard configuration.

Now open the WireGuard Peer's `/etc/wireguard/wg0.conf` file with `nano` or your preferred editor.

Copy

```
$ sudo nano /etc/wireguard/wg0.conf
```

Before the `[Peer]` line, add the following 4 lines:

```
PostUp = ip rule add table 200 from  203.0.113.5
PostUp = ip route add table 200 default via  203.0.113.1
PreDown = ip rule delete table 200 from  203.0.113.5
PreDown = ip route delete table 200 default via  203.0.113.1

[Peer]
. . .
```

These lines will create a custom routing rule, and add a custom route to ensure that public traffic to the system uses the default gateway.

- `PostUp = ip rule add table 200 from  203.0.113.5` - This command creates a rule that checks for any routing entries in the table numbered `200` when the IP matches the system's public `203.0.113.5` address.
- `PostUp = ip route add table 200 default via  203.0.113.1` - This command ensures that any traffic being processed by the `200` table will use the `203.0.113.1` gateway for routing, instead of the WireGuard interface.

The `PreDown` lines remove the custom rule and route when the tunnel is shutdown.

Note: The table number `200` is arbitrary when constructing these rules. You can use a value between 2 and 252, or you can use a custom name by adding a label to the `/etc/iproute2/rt_tables` file and then referring to the name instead of the numeric value.

For more information about how routing tables work in Linux visit the Routing Tables Section of the Guide to IP Layer Network Administration with Linux.

If you are routing all the peer's traffic over the VPN, ensure that you have configured the correct `sysctl` and `iptables` rules on the WireGuard Server in Step 4 — Adjusting the WireGuard Server's Network Configuration and Step 5 — Configuring the WireGuard Server's Firewall.

(Optional) Configuring the WireGuard Peer's DNS Resolvers

If you are using the WireGuard Server as a VPN gateway for all your peer's traffic, you will need to add a line to the `[Interface]` section that specifies DNS resolvers. If you do not add this setting, then your DNS requests may not be secured by the VPN, or they might be revealed to your Internet Service Provider or other third parties.

If you are only using WireGuard to access resources on the VPN network or in a peer-to-peer configuration then you can skip this section.

To add DNS resolvers to your peer's configuration, first determine which DNS servers your WireGuard Server is using. Run the following command on the **WireGuard Server**, substituting in your ethernet device name in place of `eth0` if it is different from this example:

Copy

```
$ resolvectl dns  eth0
```

You should receive output like the following:

```
Output
Link 2 (eth0): 67.207.67.2 67.207.67.3 2001:4860:4860::8844 2001:4860:4860::8
```

The IP addresses that are output are the DNS resolvers that the server is using. You can choose to use any or all of them, or only IPv4 or IPv6 depending on your needs. Make a note of the resolvers that you will use.

Next you will need to add your chosen resolvers to the WireGuard Peer's configuration file. Back on the **WireGuard Peer**, open `/etc/wireguard/wg0.conf` file using `nano` or your preferred editor:

Copy

```
$ sudo nano /etc/wireguard/wg0.conf
```

Before the `[Peer]` line, add the following:

```
DNS =  67.207.67.2 2001:4860:4860::8844

[Peer]
. . .
```

Again, depending on your preference or requirements for IPv4 and IPv6, you can edit the list according to your needs.

> Once you are connected to the VPN in the following step, you can check that you are sending DNS queries over the VPN by using a site like DNS leak test.com.
>
> You can also check that your peer is using the configured resolvers with the `resolvectl dns` command like you ran on the server. You should receive output like the following, showing the DNS resolvers that you configured for the VPN tunnel:
>
> ```
> Output
> Global: 67.207.67.2 67.207.67.3
> . . .
> ```

With all of these DNS resolver settings in place, you are now ready to add the peer's public key to the server, and then start the WireGuard tunnel on the peer.

Step 8 – Adding the Peer's Public Key to the WireGuard Server

Before connecting the peer to the server, it is important to add the peer's public key to the WireGuard Server. This step ensures that you will be able to connect to and route traffic over the VPN. Without completing this step the WireGuard server will not allow the peer to send or receive any traffic over the tunnel.

Ensure that you have a copy of the `base64` encoded public key for the WireGuard Peer by running:

Copy

```
$ sudo cat /etc/wireguard/public.key
```

```
Output
 PeURxj4Q75RaVhBKkRTpNsBPiPSGb5oQijgJsTa29hg=
```

Now log into the WireGuard server, and run the following command:

Copy

```
$ sudo wg set wg0 peer PeURxj4Q75RaVhBKkRTpNsBPiPSGb5oQijgJsTa29hg= allowed-
```

Note that the `allowed-ips` portion of the command takes a comma separated list of IPv4 and IPv6 addresses. You can specify individual IPs if you would like to restrict the IP address that a peer can assign itself, or a range like in the example if your peers can use any IP address in the VPN range. Also note that no two peers can have the same `allowed-ips` setting.

If you would like to update the `allowed-ips` for an existing peer, you can run the same command again, but change the IP addresses. Multiple IP addresses are supported. For example, to change the WireGuard Peer that you just added to add an IP like `10.8.0.100` to the existing `10.8.0.2` and `fd0d:86fa:c3bc::2` IPs, you would run the following:

Copy

```
$ sudo wg set wg0 peer PeURxj4Q75RaVhBKkRTpNsBPiPSGb5oQijgJsTa29hg= allowed-
```

Once you have run the command to add the peer, check the status of the tunnel on the server using the `wg` command:

Copy

```
$ sudo wg
```

```
Output
interface: wg0
 public key: U9uE2kb/nrrzsEU58GD3pKFU3TLYDMCbetIsnV8eeFE=
 private key: (hidden)
 listening port: 51820

peer: PeURxj4Q75RaVhBKkRTpNsBPiPSGb5oQijgJsTa29hg=
 allowed ips: 10.8.0.2/32, fd0d:86fa:c3bc::/128
```

Notice how the `peer` line shows the WireGuard Peer's public key, and the IP addresses, or ranges of addresses that it is allowed to use to assign itself an IP.

Now that you have defined the peer's connection parameters on the server, the next step is to start the tunnel on the peer.

Step 9 – Connecting the WireGuard Peer to the Tunnel

Now that your server and peer are both configured to support your choice of IPv4, IPv6, packet forwarding, and DNS resolution, it is time to connect the peer to the VPN tunnel.

Since you may only want the VPN to be on for certain use cases, we'll use the `wg-quick` command to establish the connection manually. If you would like to automate starting the tunnel like you did on the server, follow those steps in Step 6 — Starting the WireGuard Server section instead of using the `wq-quick` command.

In case you are routing all traffic through the VPN and have set up DNS forwarding, you'll need to install the `resolvconf` utility on the WireGuard Peer before you start the tunnel. Run the following command to set this up:

Copy

```
$ sudo apt install resolvconf
```

To start the tunnel, run the following on the WireGuard Peer:

Copy

```
$ sudo wg-quick up wg0
```

You will receive output like the following:

```
Output
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add  10.8.0.2/24  dev wg0
[#] ip -6 address add  fd0d:86fa:c3bc::2/64  dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] resolvconf -a tun.wg0 -m 0 -x
```

Notice the highlighted IPv4 and IPv6 addresses that you assigned to the peer.

If you set the `AllowedIPs` on the peer to `0.0.0.0/0` and `::/0` (or to use ranges other than the ones that you chose for the VPN), then your output will resemble the following:

```
Output
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add  10.8.0.2/24  dev wg0
[#] ip -6 address add  fd0d:86fa:c3bc::2/64  dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] resolvconf -a tun.wg0 -m 0 -x
[#] wg set wg0 fwmark 51820
[#] ip -6 route add  ::/0  dev wg0 table 51820
[#] ip -6 rule add not fwmark 51820 table 51820
[#] ip -6 rule add table main suppress_prefixlength 0
[#] ip6tables-restore -n
[#] ip -4 route add  0.0.0.0/0  dev wg0 table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] sysctl -q net.ipv4.conf.all.src_valid_mark=1
[#] iptables-restore -n
```

In this example, notice the highlighted routes that the command added, which correspond to the `AllowedIPs` in the peer configuration.

You can check the status of the tunnel on the peer using the `wg` command:

Copy

```
$ sudo wg
```

```
Output
interface: wg0
```

```
 public key: PeURxj4Q75RaVhBKkRTpNsBPiPSGb5oQijgJsTa29hg=
 private key: (hidden)
 listening port: 49338
 fwmark: 0xca6c

peer: U9uE2kb/nrrzsEU58GD3pKFU3TLYDMCbetIsnV8eeFE=
 endpoint: 203.0.113.1:51820
 allowed ips: 10.8.0.0/24, fd0d:86fa:c3bc::/64
 latest handshake: 1 second ago
 transfer: 6.50 KiB received, 15.41 KiB sent
```

You can also check the status on the server again, and you will receive similar output.

Verify that your peer is using the VPN by using the `ip route` and `ip -6 route` commands. If you are using the VPN as a gateway for all your Internet traffic, check which interface will be used for traffic destined to CloudFlare's `1.1.1.1` and `2606:4700:4700::1111` DNS resolvers.

> If you are only using WireGuard to access resources on the VPN, substitute a valid IPv4 or IPv6 address like the gateway itself into these commands. For example `10.8.0.1` or `fd0d:86fa:c3bc::1`.

Copy

```
$ ip route get 1.1.1.1
```

```
Output
1.1.1.1 dev  wg0  table 51820 src  10.8.0.2  uid 1000
   cache
```

Notice the `wg0` device is used and the IPv4 address `10.8.0.2` that you assigned to the peer. Likewise, if you are using IPv6, run the following:

Copy

```
$ ip -6 route get 2606:4700:4700::1111
```

```
Output
```

```
2606:4700:4700::1111 from :: dev wg0 table 51820 src  fd0d:86fa:c3bc::2  metri
```

Again note the `wg0` interface, and the IPv6 address `fd0d:86fa:c3bc::2` that you assigned to the peer.

If your peer has a browser installed, you can also visit ipleak.net and ipv6-test.com to confirm that your peer is routing its traffic over the VPN.

Once you are ready to disconnect from the VPN on the peer, use the `wg-quick` command:

Copy

```
$ sudo wg-quick down wg0
```

You will receive output like the following indicating that the VPN tunnel is shut down:

```
Output
[#] ip link delete dev wg0
[#] resolvconf -d tun.wg0 -f
```

If you set the `AllowedIPs` on the peer to `0.0.0.0/0` and `::/0` (or to use ranges other than the ones that you chose for the VPN), then your output will resemble the following:

```
Output
[#] ip rule delete table 200 from 203.0.113.5
[#] ip route delete table 200 default via 203.0.113.1
[#] ip -4 rule delete table 51820
[#] ip -4 rule delete table main suppress_prefixlength 0
[#] ip -6 rule delete table 51820
[#] ip -6 rule delete table main suppress_prefixlength 0
[#] ip link delete dev wg0
[#] resolvconf -d tun.wg0 -f
[#] iptables-restore -n
[#] ip6tables-restore -n
```

To reconnect to the VPN, run the `wg-quick up wg0` command again on the peer. If you would like to completely remove a peer's configuration from the WireGuard Server, you can run the following command, being sure to substitute the correct public key for the peer that you want to remove:

Copy

```
$ sudo wg set wg0 peer PeURxj4Q75RaVhBKkRTpNsBPiPSGb5oQijgJsTa29hg= remove
```

Typically you will only need to remove a peer configuration if the peer no longer exists, or if its encryption keys are compromised or changed. Otherwise it is better to leave the configuration in place so that the peer can reconnect to the VPN without requiring that you add its key and `allowed-ips` each time.

Conclusion

In this tutorial you installed the WireGuard package and tools on both the server and client Ubuntu 20.04 systems. You set up firewall rules for WireGuard, and configured kernel settings to allow packet forwarding using the `sysctl` command on the server. You learned how to generate private and public WireGuard encryption keys, and how to configure the server and peer (or peers) to connect to each other.

If your network uses IPv6, you also learned how to generate a unique local address range to use with peer connections. Finally, you learned how to limit which traffic should go over the VPN by restricting the network prefixes that the peer can use, as well as how to use the WireGuard Server as a VPN gateway to handle all Internet traffic for peers.

If you would like to learn more about WireGuard, including how to configure more advanced tunnels, or use WireGuard with containers, visit the official WireGuard documentation.

About the authors

Jamon Camisso Author

Still looking for an answer? Ask a question

Search for more help

Was this helpful? Yes No

Comments

10 Comments

Leave a comment...

This textbox defaults to using `Markdown` to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

mazuriktc • September 27, 2024

Good day! Should we also restrict access to the file /etc/wireguard/wg.conf using the command “chmod go= ...”? After all, we enter the private key there

Reply

6129343f9fdc44bfa7460ac8ebdb35 • July 27, 2024

Hello! I followed these instructions and was able to create the VPN successfully and have a peer connect, however I am unable to route all traffic through the tunnel on a Windows or iPhone peer. I am using a droplet with Ubuntu 20.04LTS.

My server config is as follows:

```
[Interface]
PrivateKey = $PRIVATE_KEY
Address = 10.8.0.1/24
ListenPort = 51820
SaveConfig = true
PostUp = ufw route allow in on wg0 out on eth0
PostUp = iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
PreDown = ufw route delete allow in on wg0 out on eth0
PreDown = iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE

[Peer]
PublicKey = $PUBLIC_KEY
AllowedIps = 10.8.0.2/32
```

My peer configuration is as follows:

```
[Interface]
PrivateKey = $PRIVATE_KEY
Address = 10.8.0.2/24
DNS = 67.207.67.3
PostUp = ip route add table 200 default via 104.236.0.1
PreDown = ip route delete table 200 default via 104.236.0.1

[Peer]
PublicKey = $PUBLIC_KEY
AllowedIPs = 0.0.0.0/0
Endpoint = $SERVER_IP:51820
```

And I set the following firewall values after init:

```
sudo ufw allow 51820/udp
sudo ufw allow 22/tcp
sudo ufw allow out 53
sudo ufw allow out 80/tcp
sudo ufw allow out 443/tcp
sudo ufw reload
```

Reply

kendofriendo • January 5, 2024

`ip route list table main default` and `ip -brief address show eth0` do not show a pubic ip, they show private IPs. Obviously using the private IPs won't work, and using the public IP of the server didn't work either. I had to remove those lines for it to work at all.

Reply

mrjlodge • September 21, 2023

There appears to be a tpyo in the output of `sudo ufw status` [Ubuntu 20.04] i.e. should read 51820 not 51280. No biggie since the pertinent commands to actually run are correct.

Reply

Florian Bidabe • July 4, 2023

This article was brilliant. Thank you for sharing. One word of caution, the WireGuard Peer is not necessarily an enpoint. In my case, it was an OpenWrt router (GL.iNet) so I had to create the client keys and configuration on the WireGuard server. The commands provided to generate the peer keys would overwrite the server keys so you might want to use a different names or directories.

I can now connect my GLiNet Access Point as a WireGuard Client and have WiFi clients connecting from AWS Public IP Space for my selected regions (WireGuard Servers on EC2 instances). Thank you!

Reply

FloatingLapisRay • April 19, 2023

I really enjoyed this tourial on wireguard. I've been meaning to set this up and I was able to utilize the server setup on my home machine, and the client setup on my android phone. I left off all of the NAT code and had to punch holes through my cable modem and AP, but in the end everything worked nicely. Now I can bring up the wireguard connection and then ssh in from my phone to my home system. Plan to repeat the client setup for my Chromebook as well. Thanks for the great description, especially in noting the "optional" steps that I didn't require.

Reply

jamesthedove • February 20, 2023

If you having configuration errors.

Make sure you didn't copy the `/etc/wireguard/wg0.conf` at the beginning of the configuration.

Check the `/etc/wireguard/wg0.conf` file, and ensure the first line doesn't include `/etc/wireguard/wg0.conf`. This was added to the snippet in the tutorial but it is not part of the configuration

Show replies Reply

Houman Ayazi • December 24, 2022

Hello, When I want to run the service I get this error message:

× wg-quick@wg0.service - WireGuard via wg-quick(8) for wg0 Loaded: loaded (/lib/systemd/system/wg-quick@.service; enabled; vendor preset: enabled) Active: failed (Result: exit-code) since Sat 2022-12-24 08:21:21 UTC; 51s ago Docs: man:wg-quick(8) man:wg(8) https://www.wireguard.com/ https://www.wireguard.com/quickstart/ https://git.zx2c4.com/wireguard-tools/about/src/man/wg-quick.8 https://git.zx2c4.com/wireguard-tools/about/src/man/wg.8 Process: 38627 ExecStart=/usr/bin/wg-quick up wg0 (code=exited, status=1/FAILURE) Main PID: 38627 (code=exited, status=1/FAILURE) CPU: 31ms

Reply

Pinkkter • November 29, 2022

Hello, you said that there can be up to 255 different nodes on an IPv4 subnet. How about IPv6? How many peers can there be on an IPv6 subnet?

Reply

Vanessa Wilcken • November 6, 2022

Hello, I tried several times now and I always get the same error. Anybody an idea? I am a complete banana in this and dont understand much. Any help very much appreciated.

lines 1-22/22 (END)...skipping... × wg-quick@wg0.service - WireGuard via wg-quick(8) for wg0 Loaded: loaded (/lib/systemd/system/wg-quick@.service; enabled; preset: enabled) Active: failed (Result: exit-code) since Sun 2022-11-06 22:36:52 UTC; 18s ago Docs: man:wg-quick(8) man:wg(8) https://www.wireguard.com/ https://www.wireguard.com/quickstart/ https://git.zx2c4.com/wireguard-tools/about/src/man/wg-quick.8 https://git.zx2c4.com/wireguard-tools/about/src/man/wg.8 Process: 2435 ExecStart=/usr/bin/wg-quick up wg0 (code=exited, status=1/FAILURE) Main PID: 2435 (code=exited, status=1/FAILURE) CPU: 18ms

Nov 06 22:36:52 climbingcervino systemd[1]: Starting WireGuard via wg-quick(8) for wg0... Nov 06 22:36:52 climbingcervino wg-quick[2435]: [#] ip link add wg0 type wireguard Nov 06 22:36:52 climbingcervino wg-quick[2435]: [#] wg setconf wg0 /dev/fd/63 Nov 06 22:36:52 climbingcervino wg-quick[2457]: Line unrecognized: `/etc/wireguard/wg0.conf' Nov 06 22:36:52 climbingcervino wg-quick[2457]: Configuration parsing error Nov 06 22:36:52 climbingcervino wg-quick[2435]: [#] ip link delete dev wg0 Nov 06 22:36:52 climbingcervino systemd[1]: wg-quick@wg0.service: Main process exited, code=exited, status=1/FAILURE Nov 06 22:36:52 climbingcervino systemd[1]: wg-quick@wg0.service: Failed with result 'exit-code'. Nov 06 22:36:52 climbingcervino systemd[1]: Failed to start WireGuard via wg-quick(8) for wg0.
~ ~ ~ ~

Show replies Reply

Load more comments

Popular Topics

AI/ML

Ubuntu

Linux Basics

JavaScript

Python

MySQL

Docker

Kubernetes

All tutorials →

Talk to an expert →

Become a contributor for community

Get paid to write technical tutorials and select a tech-focused charity to receive a matching donation.

Sign Up →

DigitalOcean Documentation

Full documentation for every DigitalOcean product.

Learn more →

Resources for startups and SMBs

Company

Products

Resources

Solutions

Contact