Using the WireGuard VPN
http://dashohoxha.fs.al/using-wireguard-vpn/

Dashamir Hoxha
dashohoxha@gmail.com
About WireGuard

WireGuard is a simple, fast and modern VPN that utilizes state-of-the-art cryptography. It is quite flexible and can be used in many situations.

- Simple & Easy-to-use
- Cryptographically Sound
- Minimal Attack Surface
- High Performance
- Well Defined & Thoroughly Considered

For more details have a look at its page: https://www.wireguard.com/
We will install a WireGuard server with Docker and `docker-scripts`:

1. Install **Docker**
2. Install `docker-scripts`:
   ```bash
   apt install git make m4
   git clone \
   https://gitlab.com/docker-scripts/ds \
   /opt/docker-scripts/ds
   cd /opt/docker-scripts/ds/
   make install
   
   3. Install the WireGuard container:
   ```bash
   ds pull wireguard
   ds init wireguard @wireguard
   cd /var/ds/wireguard/
   vim settings.sh
   ds make
Settings of the WG container (on `settings.sh`)

➢ **ROUTED_NETWORKS** = "0.0.0.0/0"
Tells the client what to route to the WG interface. In this case all the internet traffic will go through the WG tunnel.

➢ **DNS_SERVERS** = "176.103.130.130, 176.103.130.131"
Tells the client which DNS servers to use. You can use your preferred DNS servers here.

➢ **ALLOW_INTERNET_ACCESS** = yes
Allows the WG server to behave as a NAT server for the clients, providing them access to the Internet.

➢ **CLIENT_TO_CLIENT** = no
Tells the server to block the connections between the clients.

➢ **KEEPALIVE_PERIOD** = 0
Disables the *keepalive* feature, which makes the client to send periodically a packet to the server. Usually 25 is a good value for it.
Test the WireGuard server

1. Create client configurations on the WG server:
   
   ```
   ds client add client1 192.168.100.1
   ds client add client2 192.168.100.2
   ```

2. Send configuration files to each client
   
   ```
   ds share www client1
   ds share www client2
   ```

3. Get config file on the client:
   
   ```
   wget --no-check-certificate -O client1.conf \
   ```

4. Test the VPN connection on the client:
   
   ```
   apt install wireguard
   wg-quick up ./client1.conf
   curl ifconfig.co
   wg-quick down ./client1.conf
   ```

5. Start the VPN connection as a service:
   
   ```
   mv client1.conf /etc/wireguard/wg0.conf
   systemctl enable wg-quick@wg0
   systemctl start wg-quick@wg0
   curl ifconfig.co
   ```
Some of the usecases supported by this WG server are:

1. Securing connections to the Internet
2. Creating a Virtual Private LAN
3. Routing between remote private LANs
4. Accessing clients from a cloud server
This is maybe the most popular reason why people want to use a VPN. If you are connected to an unknown WiFi hotspot (one that it is not under your control), it is quite possible that someone may try to eavesdrop your communications, and maybe try to hack you. The recommended solution is to use a VPN for connecting to the Internet.
Usecase #2: Virtual Private LAN

In this case there are several computers distributed all over the internet, and we want them to communicate with each-other safely and securely, as if they are in a private LAN.

ROUTED_NETWORKS="192.168.10.0/24"
ALLOW_INTERNET_ACCESS=no
CLIENT_TO_CLIENT=yes
KEEPALIVE_PERIOD=25
Use case #3: Routing between remote private LANs

This case is an extension of the second case, in the sense that not only the WG clients can communicate with each other, but also the other clients on their LANs can communicate with each other.

**ROUTED_NETWORKS:**
192.168.11.0/24, 192.168.12.0/24

**ALLOW_INTERNET_ACCESS:** no

**CLIENT_TO_CLIENT:** yes

**KEEPALIVE_PERIOD:** 25
Use case #4: Accessing clients from a cloud server

In this case we want the clients and a containerized application on the cloud to be able to access each-other, but the clients themselves should not be able to access each-other.

```
ROUTED_NETWORKS="172.24.0.0/16"
ALLOW_INTERNET_ACCESS=no
CLIENT_TO_CLIENT=no
KEEPALIVE_PERIOD=25
```
Use case #4 example:
Remote access to Computer Labs, with Guacamole server on a VPS on the cloud

See: http://dashohoxha.fs.al/accessing-computer-labs-remotely/
Testing the usecases of the WG server

Using Docker and docker-scripts we create virtual environments for testing the different usecases of the WireGuard server.

- Test #1: Accessing the internet from a WG client
- Test #2: Two clients cannot ping each-other
- Test #3: Two clients can ping each-other
- Test #4: Virtual private LAN
- Test #5: Routing between two LANs
- Test #6: Accessing clients from a cloud server
Test #1: Accessing the internet from a WG client

The first test is related to the Usecase #1, where a WG client can access the internet through a secure connection to the WG server.

```
apt install highlight expect ./test1.sh
```

The default settings of the server are OK, so we don't need to modify them. After creating configuration files (for the client and for the server), we go to client1 and:

1. start up the interface wg0
2. test the connection to the WG server
3. test the connection to the internet
Test #2: Two clients cannot ping each-other

This test is related to the Usecase #1, where a WG client can access the internet through a secure connection to the WG server. By default, two different clients cannot ping each-other. The setting `CLIENT_TO_CLIENT` by default is commented out (which means disabled).

The default settings of the server are OK, so we don't need to modify them. After creating configuration files:

1. start up `wg0` on client1
2. start up `wg0` on client2
3. try to ping from client1 to client2
4. try to ping from client2 to client1
Test #3: Two clients can ping each-other

This test is related to a customized version of Usecase #1, where WG clients can also ping each-other, besides accessing the internet through the WG server. This requires the setting `CLIENT_TO_CLIENT=yes` (enabled).

Test steps:
1. set `CLIENT_TO_CLIENT=yes` on the server
2. generate configuration files
3. start up `wg0` on client1
4. start up `wg0` on client2
5. ping from client1 to client2
6. ping from client2 to client1
Test #4: Virtual private LAN

This test is related to the Usecase #2, where clients can access each-other, but cannot access the internet through the WG interface (because of the setting `ALLOW_INTERNET_ACCESS=no`).

Settings:

- `ROUTEDNETWORKS="10.100.100.1, 192.168.100.0/24"`
- `ALLOW_INTERNET_ACCESS=no`
- `CLIENT_TO_CLIENT=yes`
- `KEEPALIVE_PERIOD=25`
Test #5: Routing between two LANs

This test is related to the Use case #3, where clients on two different private LANs, can access each other through the WG network.

Settings:

ROUTE\_NETWORKS="10.100.100.1/32, 192.168.100.0/24, 172.25.0.0/16, 172.26.0.0/16"
ALLOW\_INTERNET\_ACCESS=no
CLIENT\_TO\_CLIENT=yes
KEEPALIVE\_PERIOD=25
Test #6: Accessing clients from a cloud server

This test is related to Use-case #4, where a containerized application on the cloud, which is located on the same docker network as the WG server, can access clients that don't have a public IP. At the same time, clients can access the application, but not each other.

Settings:

ROUTED_NETWORKS="10.100.100.1, 172.24.0.0/16"
ALLOW_INTERNET_ACCESS=no
CLIENT_TO_CLIENT=no
KEEPALIVE_PERIOD=25
Thank you for your attention!

Any questions or comments?