

INTRODUCCIÓN Y DESARROLLO DE VPN CON WIREGUARD



Javier Pérez Hidalgo

ÍNDICE

¿ Qué es una VPN ?	3
¿ Son necesarias las VPNs ?	4
¿ Qué es WireGuard ?.....	5
Principales diferencias con OpenVPN	6
Conexión Side to Side.....	7
Escenario real para producción	7
Configuración servidor Debian 11 (primera parte)	7
Configuración cliente Debian 11.....	11
Configuración servidor Debian 11 (añadiendo cliente Linux).....	13
Configuración cliente Windows.....	17
Configuración cliente Android y cliente iOS	19
Conclusión.....	21

A día de hoy, nos encontramos con una situación de monopolio absoluto en el mundo de las VPN, liderado por *OpenVPN*, que ya todos conoceréis, o incluso habréis trabajado con él.

He mencionado el término VPN, pero ¿qué es una VPN?

¿Qué es una VPN?

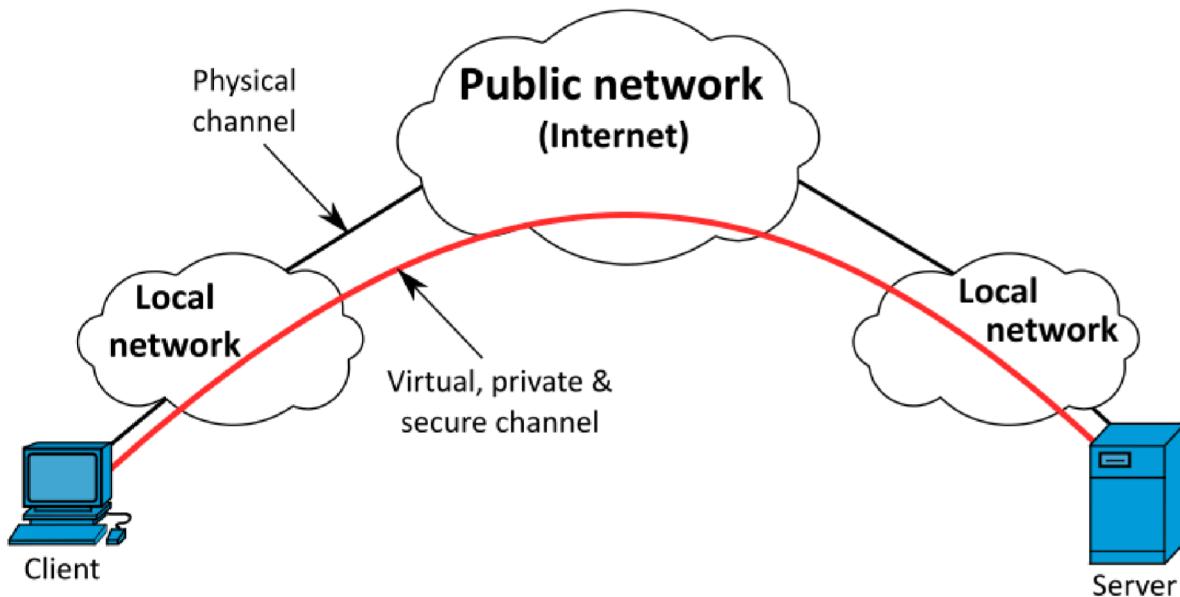
El término VPN viene de las siglas de “Virtual Private Network”, es una tecnología que permite a través de una conexión a internet, una conexión segura a otra red privada.

Permite, además, enviar y recibir datos como si se formase parte de la red local o privada, obteniendo así las funcionalidades que dicha red ofrece.

También se puede hacer referencia a una red privada virtual como un túnel, pues realmente lo que hace es crear un canal para el tráfico entre el cliente y el servidor VPN en el que su contenido corre independientemente del resto y de manera cifrada.

Básicamente, lo que te permite es crear una red local sin necesidad que sus integrantes estén físicamente conectados entre sí. Esto nos proporciona una flexibilidad absoluta, ya que, por ejemplo, podríamos crear una red privada con clientes de una punta del mundo, y clientes de la otra.

Bien, una vez conocemos lo que es una VPN y las ventajas que nos ofrece, es el momento de preguntarnos, ¿necesitamos las VPNs?



¿ Son necesarias las VPNs ?

Habitualmente, cuando navegamos por internet tan solo hacemos uso de nuestra conexión mediante el proveedor contratado, de manera que todo el tráfico de datos, es posible que sea consultado por este proveedor de internet.

¿Esto quiere decir que al usar una VPN nuestro tráfico no pasa por el proveedor?

La respuesta es, no. Al usar una conexión VPN, el tráfico de red sigue yendo del dispositivo al proveedor, pero de este, se dirige directo al servidor VPN, desde donde partirá al destino.

Normalmente la conexión está cifrada, de modo que el proveedor realmente no sabe a qué estás accediendo.

Respondiendo a la primera pregunta, sí, son muy necesarias, desde la seguridad que aportan, hasta la posibilidad que nos ofrece al permitirnos el acceso a contenido restringido en una localización, pasando por el teletrabajo, que tan necesario se ha vuelto actualmente, y la posibilidad de ofrecernos confidencialidad que muchos buscamos.

A efectos prácticos, al hacer uso de una VPN, tu dirección IP de cara a internet, es la del servidor VPN.

Otra de sus ventajas es que al utilizar una conexión VPN, no tenemos ningún tipo de limitación, ya que funciona en todas las aplicaciones además de en un navegador.

Vistas las ventajas y utilidades de las VPNs, vamos a retomar un poco el contenido en sí de este proyecto.

Como he dicho al principio, hoy en día nos encontramos con una gran monopolización en el ámbito de las VPNs de código abierto, donde lidera OpenVPN. He experimentado la creación de un servidor y sus respectivos clientes, con dicho software, y siempre me ha parecido un proceso algo complejo y bastante lento de realizar.

Teniendo en cuenta estos aspectos, me interesé por WireGuard, y empecé a investigar un poco, dándome cuenta rápidamente y quedando gratamente sorprendido al ver sus sencillos procesos de configuración.

¿ Qué es WireGuard ?

WireGuard es un protocolo de comunicación de software libre que implementa técnicas de VPN para crear conexiones seguras punto a punto en configuraciones enrutadas o puenteadas.

Es una aplicación multiplataforma (Linux, Windows, macOS, Android e iOS). Se presenta como una solución más rápida y sencilla que las existentes hasta la fecha, para aquellos usuarios que necesiten o deseen hacer uso de una VPN.

Nació en el año 2016, por lo que es una tecnología muy reciente, aún en desarrollo y no conocida por muchos. Está escrita por Jason A. Donenfeld en C++ y en Go, y publicada bajo GPL.

Su objetivo es convertirse en un estándar en usuarios domésticos y en empresas, en lugar de IPsec u OpenVPN, que son más difíciles y más lentos, esto a consecuencia de que WireGuard consume muy pocos recursos del sistema.

Es compatible con redes IPv4 y con redes IPv6, además de poder encapsular paquetes IPv4 en IPv6 y viceversa.

Uno de los puntos fuertes de este software es que la configuración del cliente y servidor es exactamente igual en los diferentes sistemas operativos que soporta, lo que lo hace muy manejable y adaptable.

Sus creadores, aseguran una gran superioridad frente a OpenVPN, en aspectos como la velocidad de transmisión y la incorporación de un cifrado de siguiente nivel.

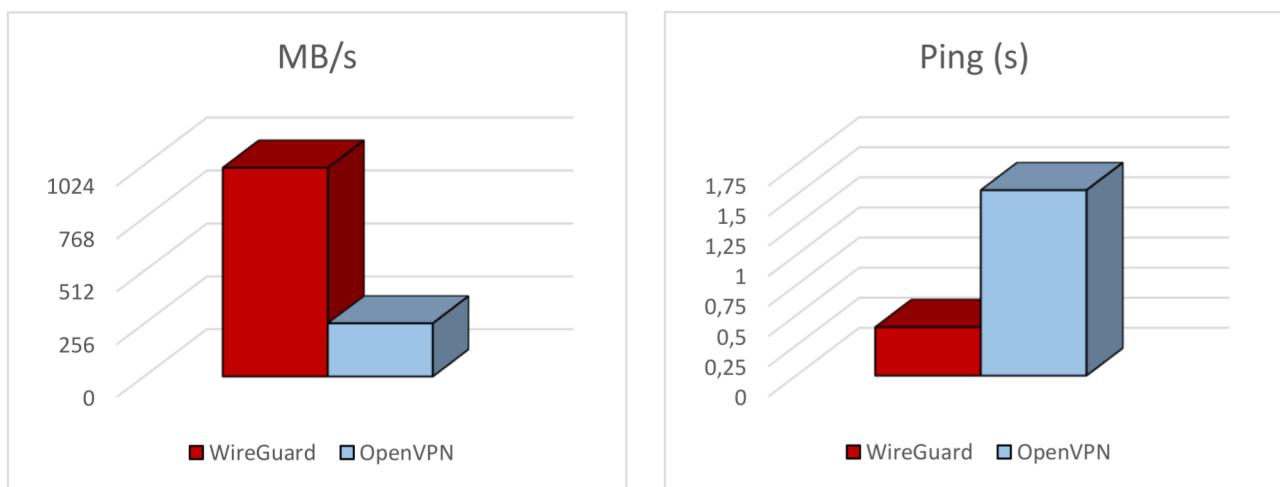
Hace uso del protocolo UDP, normalmente se utiliza el puerto 51820, aunque por defecto WireGuard utiliza puertos dinámicos. En mi caso, recomiendo asignar de manera estática el puerto 51820, ya que es bastante más recomendable si vamos a implantar la aplicación detrás de un cortafuegos.

Principales diferencias con OpenVPN

Para combatir a su principal rival e incitar al cambio a los clientes, WireGuard se ha visto en la obligación de tener que incorporar una serie de mejoras.

Entre las más significativas, se encuentran las siguientes:

- Notables mejoras de rendimiento: WireGuard casi toca el rendimiento del 100% de una conexión de 1 Gbps, y lo hace sin maximizar la CPU del servidor, esto, gracias a que su código está formado por unas 4 000 líneas de código, frente a las más de 100 000 que forman OpenVPN.
- Mejoras de ping: WireGuard registra un tiempo de ping que es menos de la mitad del de OpenVPN.



- Configuraciones: Las configuraciones de WireGuard son bastante más sencillas y rápidas de realizar que las de OpenVPN. Esto es debido a que en WireGuard las conexiones se realizan mediante pares de claves públicas-privadas, no mediante nodos centrales, como es el caso de OpenVPN.
- Más seguro: WireGuard utiliza métodos criptográficos más recientes, mientras que el cifrado de OpenVPN está un poco des-actualizado, lo que afirma su lugar como el protocolo más seguro de los dos.
- Compatibilidad con roaming: WireGuard está diseñado para poder realizar roaming de manera fácil y rápida, es decir, si nuestro dispositivo cambia de red, y lógicamente cambia de IP pública, la conexión VPN seguirá levantada porque se volverán a autenticar rápidamente con el servidor VPN, de tal forma que siempre estaremos conectados a la VPN.
- Kill-Switch: En WireGuard podremos habilitar el *Kill-Switch* en los dispositivos, de esta forma, si la conexión VPN se interrumpe, el propio software también se encargará de interrumpir todo el tráfico de red hasta que se vuelva a reestablecer la conexión VPN, con el objetivo de que no naveguemos sin la protección que nos brinda esta VPN.

En este punto, podemos hacernos una idea de todo el potencial de WireGuard, y de lo que promete llegar a ser en un futuro no muy lejano. Para verlo con un poco más de profundidad, y como no podía faltar, vamos a ver algunos ejemplos prácticos de este software.

Conexión Side to Side

En primer lugar, para mostrar el funcionamiento de WireGuard, voy a mostrar cómo se realiza una conexión “**Side to Side**” mediante comandos. Esto nos servirá para comprender mejor la comodidad que nos brinda WireGuard, aunque no es lo recomendable para una puesta en producción.

Para ello he grabado el siguiente vídeo donde se puede apreciar con todo detalle el proceso:

[Demo Side to Side - WireGuard](#)

Escenario real para producción

Además del ejemplo visto en el apartado anterior, he preparado un escenario más elaborado y que simularía una puesta en producción mucho más real.

Este escenario está liderado por un servidor que se encontrará en una máquina virtual con un sistema **Debian 11**.

Como clientes voy a incorporar uno con un sistema **Debian 11** y un cliente **Windows**.

Adicionalmente, WireGuard posee una app tanto en *Play Store* para usuarios de *Android*, como en *App Store* para los usuarios de *iOS*. Dada esta posibilidad, también incorporaré un cliente de cada uno de estos tipos. Además, en todos estos dispositivos, las configuraciones las realizaré mediante ficheros, es decir, lo idóneo para una puesta en producción. Hecha esta pequeña introducción vamos a pasar con las propias configuraciones en sí.

Configuración servidor Debian 11 (primera parte)

En primer lugar, empezaremos viendo la configuración del servidor WireGuard, recordemos en una máquina Debian 11. Comenzaremos instalando el paquete en sí. Este paquete estará disponible en repositorios a partir de Debian 11, para versiones anteriores lo podremos encontrar en repositorios *backports*:

```
root@server:~# apt install wireguard -y
```

Una vez instalado deberemos habilitar el bit de forward, para permitir el reenvío de paquetes:

```
root@server:~# nano /etc/sysctl.conf  
...  
net.ipv4.ip_forward = 1  
...  
  
root@server:~# sysctl -p /etc/sysctl.conf  
net.ipv4.ip_forward = 1
```

Hecho esto podremos dirigirnos al directorio de trabajo de WireGuard, que se encuentra en esta ruta:

```
root@server:~# cd /etc/wireguard/
```

Modificaremos la política de permisos en este directorio para que los ficheros que vamos a crear en los pasos posteriores, se creen de manera predeterminada con los permisos adecuados:

```
root@server:/etc/wireguard# umask 077
```

Ahora es el paso de generar nuestro par de claves públicas privadas mediante el siguiente comando que nos facilita WireGuard:

```
root@server:/etc/wireguard# wg genkey | tee serverprivatekey | wg pubkey > serverpublickey  
  
root@server:/etc/wireguard# ls -l  
total 8  
-rw----- 1 root root 45 May 26 11:09 serverprivatekey  
-rw----- 1 root root 45 May 26 11:09 serverpublickey  
  
root@server:/etc/wireguard# cat serverprivatekey  
wOENVR47BOYbfHFUUiYLq2H3xKOSZAe0oNNUXoHXVGk=  
  
root@server:/etc/wireguard# cat serverpublickey  
cgJ6GfgX1x+YCDzW7TyrmuPzxfkJf5798h+NWwmVlmk=
```

Generados dichas claves, podremos crear nuestro fichero de configuración en el servidor, lo que va a definir su comportamiento / funcionamiento. El nombre que le asignemos, será el nombre que recibirá la interfaz de red de este túnel VPN, normalmente se le asigna el nombre “wg0” seguido de la terminación “.conf”. Dicho archivo poseerá inicialmente el siguiente aspecto:

```
root@server:/etc/wireguard# nano wg0.conf

root@server:/etc/wireguard# cat wg0.conf

# Server config

[Interface]

Address = 10.0.100.1

PrivateKey = wOENVNR47BOYbfHFUUiYLq2H3xKOSZAe0oNNUXoHXVGk=

ListenPort = 51820

PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A FORWARD -o %i -j ACCEPT;
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -D FORWARD -o %i -j ACCEPT;
iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE
```

Podemos apreciar que he añadido una nueva sección llamada *Interface* y dentro de ella disponemos de varios campos:

- **Address:** define la dirección IP que tendrá nuestro servidor.
- **PrivateKey:** especificamos la clave privada que generamos anteriormente.
- **ListenPort:** este campo no es obligatorio, pero sí es recomendable añadirlo para asignar un puerto estático. En mi caso defino el 51820 que es el que normalmente se utiliza en WireGuard.
- **PostUp / PostDown:** definen las reglas de *iptables* que se levantarán y se eliminarán al iniciar o detener el servicio del servidor, de manera que podremos automatizar el proceso y será mucho más eficiente.

Es el momento de activar el servidor y ponerlo en funcionamiento. Para ello, algo que aún no he comentado, se utiliza el comando “wg-quick up/down (interfaz)” que sirve para manejar la actividad del servidor:

```
root@server:/etc/wireguard# wg-quick up wg0

[#] ip link add wg0 type wireguard

[#] wg setconf wg0 /dev/fd/63

[#] ip -4 address add 10.0.100.1 dev wg0

[#] ip link set mtu 1420 up dev wg0

[#] iptables -A FORWARD -i wg0 -j ACCEPT; iptables -A FORWARD -o wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Podemos ver cómo nos ha levantado el túnel, además, de haber aplicado las reglas especificadas, por lo que ahora mismo el servidor se encontraría activo. Esto se puede comprobar con el comando “wg”, que nos da un resumen del estado actual del servicio de WireGuard:

```
root@server:/etc/wireguard# wg

interface: wg0

public key: cgJ6GfgX1x+YCDzW7TyrmuPzxfkJf5798h+NWwmVlmk=
private key: (hidden)

listening port: 51820
```

Vamos a comprobar la nueva interfaz de red, en la podemos ver que le ha asignado la IP correctamente. Como último paso, voy a activar el inicio del servicio de WireGuard en cada arranque del sistema. Esto lo podemos hacer porque WireGuard posee una unidad de systemd, por lo que también podríamos manejar nuestro servidor a través de ella:

```
root@server:/etc/wireguard# ip a show wg0

4: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN
group default qlen 1000

    link/none

    inet 10.0.100.1/32 scope global wg0
        valid_lft forever preferred_lft forever

root@server:/etc/wireguard# systemctl enable wg-quick@wg0

Created symlink /etc/systemd/system/multi-user.target.wants/wg-quick@wg0.service →
/lib/systemd/system/wg-quick@.service.
```

Configuración cliente Debian 11

En este punto vamos a pasar con la configuración del primer cliente. Se trata del cliente Linux.

El inicio del proceso es exactamente el mismo que el que hemos llevado a cabo en el servidor, por lo que omitiré las explicaciones realizadas anteriormente:

```
root@client:~# apt install wireguard -y

root@client:~# cd /etc/wireguard/

root@client:/etc/wireguard# umask 077

root@client:/etc/wireguard# wg genkey | tee clientprivatekey | wg pubkey > clientpublickey

root@client:/etc/wireguard# ls -l

total 8

-rw----- 1 root root 45 May 26 11:12 clientprivatekey

-rw----- 1 root root 45 May 26 11:12 clientpublickey

root@client:/etc/wireguard# cat clientprivatekey

qEFD2evf0hEr/oAWCTReK7BCuRjZ+zeCu45WgSV1QlQ=


root@client:/etc/wireguard# cat clientpublickey

MVD+I0Q7Y4F8dZK6Nl5Lx7C5IDIv1h+Olnf9dBmJNns=
```

De nuevo, crearemos un fichero de configuración, esta vez obviamente para el cliente.

En él, vamos a volver a establecer la sección **Interface** que ya vimos en el servidor, y también añadiremos una nueva, llamada **Peer**. En esta sección es donde aparecerán los datos relativos al servidor:

```
root@client:/etc/wireguard# nano wg0.conf

root@client:/etc/wireguard# cat wg0.conf

[Interface]

Address = 10.0.100.2/32

PrivateKey = qEFD2evf0hEr/oAWCTReK7BCuRjZ+zeCu45WgSV1QlQ=

ListenPort = 51820


[Peer]

PublicKey = cgJ6GfgX1x+YCDzW7TyrmuPzxfkJf5798h+NWwmVlmk=

AllowedIPs = 0.0.0.0/0

Endpoint = 192.168.0.49:51820
```

Podemos ver como en la sección **Interface** los campos son iguales que en el fichero del servidor, por lo que me centraré y trataré de explicar ahora la sección **Peer**.

- **PublicKey:** especificamos la clave pública del servidor.
- **AllowedIPs:** especificamos el rango de IPs que vamos a permitir que se nos asigne.
- **Endpoint:** define la dirección del servidor al que nos conectaremos.

Con esto habríamos terminado la configuración del cliente Linux.

Como podemos ver es un proceso totalmente rápido y sencillo, así que tan solo faltaría iniciar el servicio y esperar a la conexión.

Configuración servidor Debian 11 (añadiendo cliente Linux)

Como he comentado, el cliente Linux ya se encontraría listo para poder navegar a través de nuestro servidor VPN, pero el servidor aún no lo hemos configurado para que sea capaz de tratar a este cliente, por lo que vamos a llevar a cabo dicho proceso.

De nuevo editaremos el fichero de configuración “wg0.conf” y en él, ahora sí, añadiremos la sección *Peer*.

Como ya hemos visto, en la parte de los clientes, tan solo vamos a tener que añadir una sección *Peer* que definirá al propio servidor al que deseemos conectarnos, pero en el caso de los servidores es distinto, ya que tendremos que crear una sección *Peer* por cada cliente que deseamos que se conecte.

Explicado esto, voy a enseñar el contenido del fichero del servidor tras realizar las modificaciones y añadir a este nuevo cliente Linux:

```
root@server:/etc/wireguard# nano wg0.conf

root@server:/etc/wireguard# cat wg0.conf

# Server config

[Interface]

Address = 10.0.100.1

PrivateKey = wOENVR47BOYbfHFUUiYLq2H3xKOSZAe0oNNUXoHXVGk=

ListenPort = 51820

PostUp = iptables -A FORWARD -i %i -j ACCEPT; iptables -A FORWARD -o %i -j ACCEPT;
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

PostDown = iptables -D FORWARD -i %i -j ACCEPT; iptables -D FORWARD -o %i -j ACCEPT;
iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE

# Clients configs

# Debian Client

[Peer]

PublicKey = MVD+I0Q7Y4F8dZK6Nl5Lx7C5IDiv1h+Olnf9dBmJNns=
AllowedIPs = 10.0.100.2/32
PersistentKeepAlive = 25
```

Podemos apreciar que en esta nueva sección volvemos a tener 3 campos.

Nos encontramos con la clave pública esta vez del cliente, nos encontramos con el campo que definirá la dirección IP que se le va a asignar dentro de nuestro de nuestra red privada a este cliente, y nos encontramos con un nuevo campo llamado “*PersistenKeepAlive*” qué es opcional y se encargará de, si en más de 25 segundos no se realiza ninguna transmisión de datos entre el cliente/servidor, enviar un pequeño paquete que verificará que la conexión sigue activa, pero como digo no es obligatorio.

Hecho esto, aplicaremos los cambios a nuestro servidor reiniciando el servicio:

```
root@server:/etc/wireguard# wg-quick down wg0

[#] ip link delete dev wg0

[#] iptables -D FORWARD -i wg0 -j ACCEPT; iptables -D FORWARD -o wg0 -j ACCEPT; iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE


root@server:/etc/wireguard# wg-quick up wg0

[#] ip link add wg0 type wireguard

[#] wg setconf wg0 /dev/fd/63

[#] ip -4 address add 10.0.100.1 dev wg0

[#] ip link set mtu 1420 up dev wg0

[#] ip -4 route add 10.0.100.2/32 dev wg0

[#] iptables -A FORWARD -i wg0 -j ACCEPT; iptables -A FORWARD -o wg0 -j ACCEPT; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE


root@server:/etc/wireguard# wg

interface: wg0
public key: cgJ6GfgX1x+YCDzW7TyrmuPzxflkJf5798h+NWwmVlmk=
private key: (hidden)
listening port: 51820


peer: MVD+I0Q7Y4F8dZK6Nl5Lx7C5IDIV1h+Olnf9dBmJNns=
allowed ips: 10.0.100.2/32
persistent keepalive: every 25 seconds
```

Podemos ver como ahora, en el estado del servidor nos aparece una nueva sección que hace referencia a nuestro cliente. Irán apareciendo más secciones a medida que añadamos más clientes a nuestra red privada.

Ahora sí, nuestro servidor estaría preparado para permitir que el cliente Linux navegue a través de él, por lo que vamos a iniciar el servicio en dicho cliente y comprobar que realmente podemos navegar y tenemos acceso a internet:

```
root@client:/etc/wireguard# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.0.100.2/32 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] wg set wg0 fwmark 51820
[#] 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
[#] nft -f /dev/fd/63

root@client:/etc/wireguard# wg
interface: wg0
public key: MVD+I0Q7Y4F8dZK6Nl5Lx7C5IDIV1h+Olnf9dBmJNns=
private key: (hidden)
listening port: 51820
fwmark: 0xca6c

peer: cgJ6GfgX1x+YCDzW7TyrmuPzxflJf5798h+NWwmVlmk=
endpoint: 192.168.0.49:51820
allowed ips: 0.0.0.0/0
latest handshake: 34 seconds ago
transfer: 156 B received, 260 B sent
```

Se puede apreciar cómo el cliente ya se ha conectado al servidor. Esto se debe a que han aparecido en los campos “*latest handshake*” y “*transfer*”. El primero define el tiempo que lleva en curso la conexión y el segundo la cantidad de datos que se han recibido/enviado.

Si visualizamos la nueva interfaz de red, podemos ver cómo nos la ha creado con la dirección IP especificada, además de comprobar que realmente poseemos conexión a internet:

```
root@client:/etc/wireguard# ip a show wg0

4: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state UNKNOWN
group default qlen 1000

    link/none

    inet 10.0.100.2/32 scope global wg0
        valid_lft forever preferred_lft forever

root@client:/etc/wireguard# ping -c 3 www.google.es

PING www.google.es (172.217.17.3) 56(84) bytes of data.

64 bytes from mad07s09-in-f3.1e100.net (172.217.17.3): icmp_seq=1 ttl=61 time=18.9 ms
64 bytes from mad07s09-in-f3.1e100.net (172.217.17.3): icmp_seq=2 ttl=61 time=20.3 ms
64 bytes from mad07s09-in-f3.1e100.net (172.217.17.3): icmp_seq=3 ttl=61 time=22.0 ms

--- www.google.es ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2087ms
rtt min/avg/max/mdev = 18.946/20.402/21.957/1.231 ms
```

En este momento, ya habríamos terminado el proceso de añadir el cliente Linux, así que vamos a pasar con un nuevo cliente, es el turno de Windows.

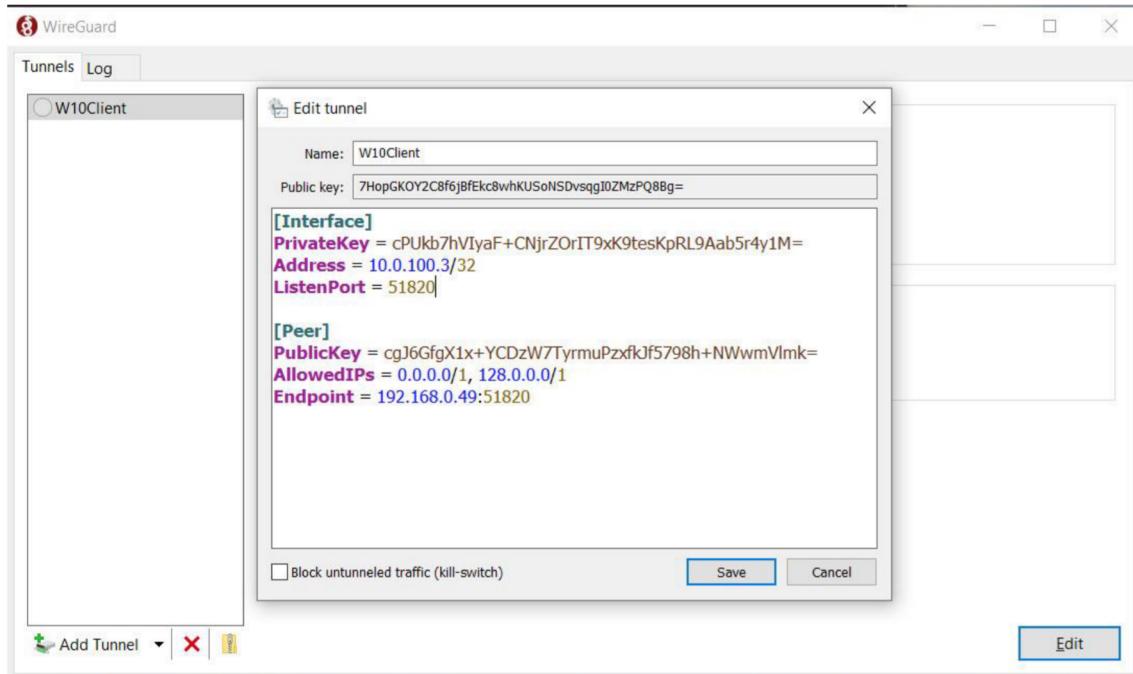
Configuración cliente Windows

Bien, una vez hemos visto cómo se realiza una configuración cliente-servidor, hemos podido darnos cuenta de que consta de 2 partes diferenciadas, la primera llevada a cabo en el cliente, donde realizamos su configuración, y la segunda llevada a cabo en el servidor, donde es necesario añadir a este nuevo cliente. Esto es siempre así, independientemente del sistema operativo que estemos utilizando, algo que ya he comentado como una de las ventajas de WireGuard.

Una vez conocemos el proceso, seremos capaces de realizarlo en todos los clientes necesarios, así que esta parte la voy a tratar un poco más por encima, dando por hecho que ya hemos aprendido los conocimientos vistos en los apartados anteriores.

Para los usuarios de Windows, WireGuard tiene disponible una aplicación de escritorio que podremos descargar desde su página oficial, en la cual realizaremos de manera gráfica su configuración, además de administrar todo lo relacionado con nuestro túnel.

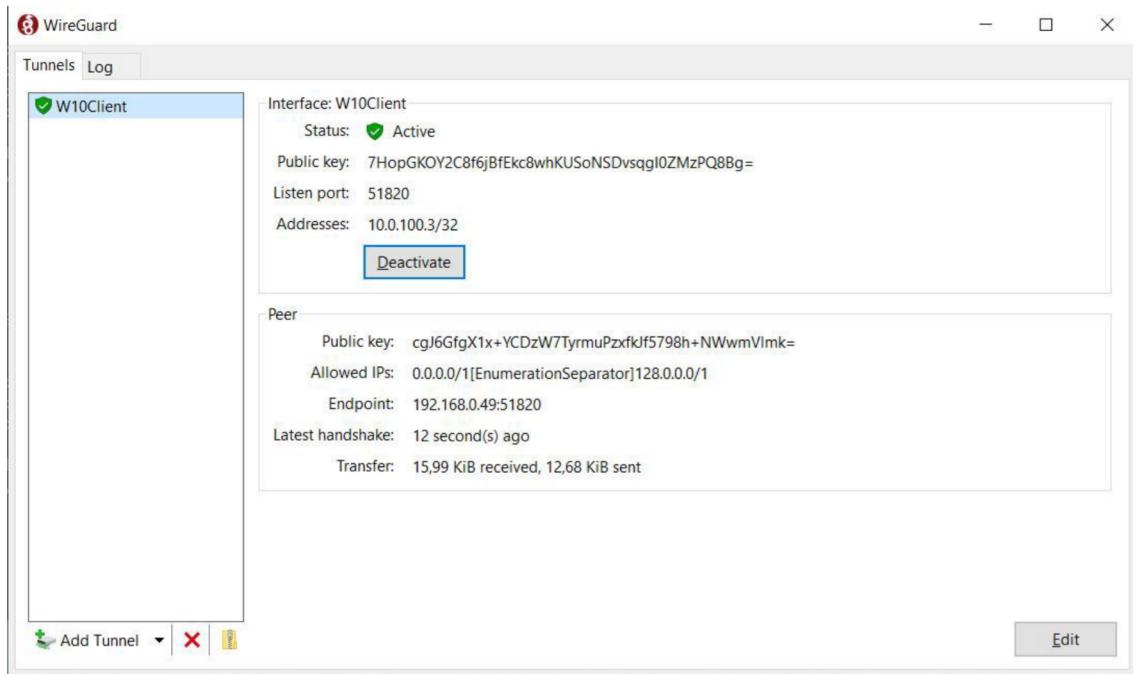
A continuación, voy a mostrar una captura de pantalla en la que visualizaremos la configuración de este nuevo cliente:



Antes de activar la conexión, añadiremos a nuestro servidor esta nueva sección *Peer* y lo reiniciaremos para aplicar cambios:

```
# W10 Client
[Peer]
PublicKey = 7HopGKOY2C8f6jBfEkc8whKUSoNSDvsqgI0ZMzPQ8Bg=
AllowedIPs = 10.0.100.3/32
PersistentKeepAlive = 25
```

Con el servidor listo, podremos activar el túnel desde nuestro cliente y automáticamente podremos visualizar la nueva interfaz con nuestra IP dentro de la red privada, además de poseer conexión a internet:



```
Administrator: Símbolo del sistema
Microsoft Windows [Versión 10.0.19043.1023]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Inma>ipconfig

Configuración IP de Windows

Adaptador desconocido W10Client:

    Sufijo DNS específico para la conexión. . . :
    Dirección IPv4. . . . . : 10.0.100.3
    Máscara de subred . . . . . : 255.255.255.255
    Puerta de enlace predeterminada . . . . . :
```

```
Administrator: Símbolo del sistema
Microsoft Windows [Versión 10.0.19043.1023]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Inma>ping www.google.es

Haciendo ping a www.google.es [142.250.200.99] con 32 bytes de datos:
Respuesta desde 142.250.200.99: bytes=32 tiempo=89ms TTL=125
Respuesta desde 142.250.200.99: bytes=32 tiempo=100ms TTL=125
Respuesta desde 142.250.200.99: bytes=32 tiempo=123ms TTL=125

Estadísticas de ping para 142.250.200.99:
    Paquetes: enviados = 3, recibidos = 3, perdidos = 0
        (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 89ms, Máximo = 123ms, Media = 104ms
```

Además de esto, he preparado una pequeña demo en la que podremos ver en tiempo real tanto el servidor como el cliente, cuando estamos navegando y realizando transmisiones de datos.

[Demo Windows 10 - WireGuard](#)

Configuración cliente Android y cliente iOS

Como ya sabemos perfectamente cómo crear y conectar clientes a nuestro servidor, sería una tontería volver a realizar los mismos procesos, aunque sean para clientes de otros sistemas por lo que, tanto para el cliente Android, como para el cliente iOS, he incorporado una pequeña modificación.

En ambas aplicaciones, disponemos de 2 métodos a la hora de realizar las configuraciones. La primera sería el método tradicional que hemos visto hasta ahora, y la segunda, trata de escanear un código QR que directamente nos importe la configuración al dispositivo.

Lógicamente vamos a ver este nuevo método. Lo voy a explicar por encima ya que en realidad es exactamente lo mismo, pero con la diferencia de que el proceso se realiza entero en la máquina servidor. Básicamente lo que vamos a hacer es, crear en el servidor, el fichero de configuración que deseemos obtener en nuestro cliente, y luego generar a partir de él, un código QR que al escanearlo en nuestros dispositivos nos importe la configuración. Una cosa que he omitido, aunque obviamente hay que tenerla en cuenta es que, será necesario volver a generar para cada cliente un nuevo par de claves.

A continuación, voy a dejar tanto el fichero para el dispositivo Android, como el fichero para el dispositivo iOS:

```
root@server:/etc/wireguard/client_android# cat clientAndroid.conf

[Interface]
Address = 10.0.100.4/32
PrivateKey = qF49LZiAUzPDqMSN5CJja4rH/ljkxOosdyicEa4YkE=
ListenPort = 51820


[Peer]
PublicKey = cgJ6GfgX1x+YCDzW7TyrmuPzxfkJf5798h+NWwmVlmk=
AllowedIPs = 0.0.0.0/0
Endpoint = 192.168.0.49:51820
```

Una vez lo tenemos nos bastaría con instalar en nuestro sistema el paquete “*qrencode*” y generar mediante el siguiente comando, el código QR a escanear en nuestro dispositivo Android:

```
root@server:/etc/wireguard/client_android# qrencode -t ansiutf8 < clientAndroid.conf
```

```
root@server:/etc/wireguard/client_iOS# cat clientiOS.conf

[Interface]

Address = 10.0.100.5/32

PrivateKey = kEmpljarBs76OA9woqrX/wzKzgVn7jWIAvABblkzmU=

ListenPort = 51820

DNS = 8.8.8.8


[Peer]

Publickey = cgJ6GfgX1x+YCDzW7TyrmuPzxfkJf5798h+NWwmVlmk=

AllowedIPs = 0.0.0.0/0

Endpoint = 192.168.0.49:51820
```

Generamos mediante el siguiente comando, el código QR a escanear en nuestro dispositivo iOS:

```
root@server:/etc/wireguard/client_iOS# qrencode -t ansiutf8 < clientiOS.conf
```

En ambos casos, únicamente nos faltaría añadir las secciones pertinentes en nuestro servidor y disfrutar de la conexión. Para ello, en el fichero “wg0.conf”, añadimos las siguientes secciones:

```
# Android Client

[Peer]

Publickey = MWNcn3OkGswxkQslf5MflmbidehNL8K/FnEHmTbnglE=

AllowedIPs = 10.0.100.4/32

PersistentKeepAlive = 25


# iOS Client

[Peer]

Publickey = 539i52F6p08kK/wKkNW5Wi20fTQWigpJbXfoWNZASks=

AllowedIPs = 10.0.100.5/32

PersistentKeepAlive = 25
```

Reiniciado el servidor y activadas las conexiones en ambos clientes, ya podríamos disfrutar de todas las ventajas de nuestra red privada. Para mostrar el funcionamiento de ambos dispositivos, he preparado al igual que anteriormente, dos vídeos en los que muestro a tiempo real el servidor y el cliente en cuestión.

[Demo Android - WireGuard](#)

[Demo iOS - WireGuard](#)

Conclusión

Espero que a través de este proyecto hayáis podido aprender y comprender todas las facilidades y ventajas que nos aporta este software, al igual que he aprendido yo.

También comentar que no hay que olvidar que se trata de una tecnología aún en desarrollo y que no posee todas las funcionalidades que seguramente incorporará más adelante.

Por mi parte, no tengo ninguna duda de que en el futuro se convertirá en una gran alternativa y un software utilizado por muchos usuarios.