HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

# NVS PROJEKT

# WireGuard

**Ausgeführt im Schuljahr 2019/20 von:**

WireGuard für Unterrichtseinsatz aufbereiten
Karlo PERANOVIC 5BHIF-18

**Lehrer:**

Dipl.-Ing. Dr. Günter Kolousek

Wiener Neustadt, am 22. April 2020

---

Abgabevermerk: Übernommen von:

# Inhaltsverzeichnis

# Kapitel 1

# WireGuard


Abbildung 1.1: WireGuard Logo

## 1.1 Einführung

WireGuard ist ein einfaches und dennoch schnelles und modernes VPN-Protkoll, welches eine sichere Lösung für das VPN-Tunneling bieten soll. Es ist darauf ausgelegt, leistungsfähiger, einfacher und nützlicher als die Konkurrenz z.B. IPsec und OpenVPN zu sein. WireGuard ist als Allzweck-VPN konzipiert, das sowohl auf eingebetteten Schnittstellen als auch auf Supercomputern ausgeführt werden kann und für viele verschiedene Umstände geeignet ist.

Ursprünglich wurde WireGuard für den Linux-Kernel veröffentlicht, ist jedoch nun plattformübergreifend (Windows, MacOS, BSD, iOS, Android) weitgehend einsetzbar. Derzeit wird WireGuard stark weiterentwickelt, aber kann jetzt schon als die sicherste, benutzerfreundlichste und einfachste VPN-Lösung in der Branche angesehen werden.

## 1.2 Installation

WireGuard kann wie in Abschnitt 1.1 beschrieben, auf vielen Betriebssystemen eingesetzt werden. Die Installation wird in weiterer Folge für das Betriebssystem Linux erklärt.

Unter Ubuntu $\geq$ 19.10 erfolgt die Installation durch:

```
1 $ sudo apt install wireguard
```

Ubuntu $\leq$ 19.04:

```
1 $ sudo add-apt-repository ppa:wireguard/wireguard
2 $ sudo apt-get update
3 $ sudo apt-get install wireguard
```

Debian:

```
1 # apt install wireguard
```

Arch:

```
1 $ sudo pacman -S wireguard-tools
```

## 1.3 Verwendung

Im folgenden Abschnitt werde ich auf alle Schritte eingehen, die zur Installation von WireGuard notwendig waren. Klar, es gibt immer mehrere Wege zu einem Ziel.

### 1.3.1 Vorbereitung

Um eine sinnvolle Funktionalität von WireGuard zu demonstrieren ist sowohl ein Server, als auch ein Client notwendig. Dazu habe ich in der Oracle VirtualBox zwei virtuelle Maschinen aufgesetzt. Auf einer VM lief ein Ubuntu Server mit der Version 18.04.4, auf der anderen VM lief ein Ubuntu Desktop System mit der Version 19.10. Der Vorgang zum Aufsetzen, erfolgt wie üblich. Ich empfehle lediglich dem Client mehr als 1GB RAM zuzuweisen, da er sonst während der Installation abstürzen kann. Zusätzlich sollte man die neueste Version von VirtualBox installieren, da in älteren Versionen offiziell Bugs bei der Kommunikation zwischen den VMs bestehen. Dies kann sonst einige Stunden Aufwand kosten :) .
Da zur Verwendung von WireGuard eine Kommunikation zwischen Client und Server und zum Internet notwendig ist, müssen ein paar Konfigurationen in der VirtualBox vorgenommen werden. Dazu muss bei beiden VMs auf den Netzwerk Adaptern *NAT* ausgewählt sein (default). Beim Ubuntu Server muss man einen zusätzlichen Netwerkadapter aktivieren und ihn als *Host-only* Adapter einstellen.

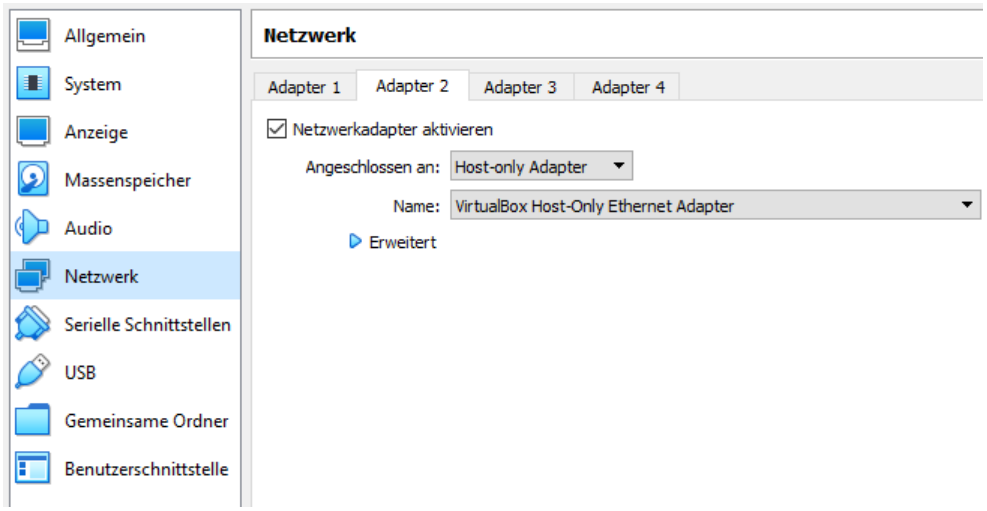


Abbildung 1.2: Host-only Adapter

Es wird davon abgeraten ein eigenes NAT-Network zu konfigurieren, da in VirtualBox somit zwar eine Kommunikation zwischen den VMs funktioniert, jedoch keine Kommunikation zum Internet, wordurch keine Installation von WireGuard möglich ist.

Anschließend muss diesem neu angelegten Adapter im Server eine IP-Adresse zugewiesen werden. Der Name des Adapters kann durch den Befehl *ifconfig -a* identifiziert werden. Unser gewünschter Adapter ist derjenige, der keine IP Adresse besitzt. In meinem Fall war das *enp0s8*. Dazu kann mit dem Befehl

```
1 $ vi /etc/network/interfaces
```

dem Adapter folgenderweise eine IP-Adresse zugewiesen werden:

```
1 auto enp0s8
2 iface enp0s8 inet static
3 address 192.168.56.103
4 netmask 255.255.255.0
```

Zuletzt kann wieder der Befehl *ifconfig* ausgeführt werden, der Folgendes zurückliefert:

```
karlo@server:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe51:61de  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:51:61:de  txqueuelen 1000  (Ethernet)
        RX packets 1117  bytes 894071 (894.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 354  bytes 31037 (31.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.103  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe2d:9e9a  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:2d:9e:9a  txqueuelen 1000  (Ethernet)
        RX packets 4  bytes 1488 (1.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 1328 (1.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Abbildung 1.3: IP Adressen der Netzwerkadapter

### 1.3.2 Server

Sind nun die Schritte aus Abschnitt 1.3.1 ausgeführt, kann die Installation von WireGuard beginnen. Dazu sind der Server und der Client zu starten. Um WireGuard am Server zu installieren ist am Client folgender Befehl auszuführen:

```
1 $ ssh <username>@192.168.56.103
```

Anschließend wurde mit folgenden Befehlen WireGuard am Server installiert:

```
1 add-apt-repository ppa:wireguard/wireguard
2 apt-get update
3 apt-get install wireguard-dkms wireguard-tools linux-headers-$(uname -r)
```

Danach wurde mit folgenden Befehlen ein private und public Key generiert:

```
1 umask 077
2 wg genkey | tee server_private_key | wg pubkey > server_public_key
```

Daraus ergibt sich Folgendes:
**private key**: +ALlz47V8oFmroNDNXB3GusLEpjpl8oHu6/1x/Lmr3Y=
**public key**: oYnqorCOyXKWInOQBKCLu6avKZYtgdPItSNwyQgHGmo=

Um die Konfigurationen langfristig zu speichern kann mit dem Befehl

```
$ vi /etc/wireguard/wg0.conf
```

eine Konfigurationsdatei erstellt werden, die folgenden Inhalt enthält:

```
[Interface]
Address = 10.100.100.1/24
SaveConfig = true
PrivateKey = +ALlz47V8oFmroNDNXB3GusLEpjpl8oHu6/1x/Lmr3Y=
ListenPort = 51820
PostUp = iptables -A FORWARDls -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

[Peer]
PublicKey = xGshpOf4b256g1ZYxYfnvUq8sdzVKvc12pig2jUJoz4=
AllowedIPs = 10.100.100.2/32
```

In Zeile 4 tragen wir den generierten Private Key des Servers ein. Zeile 11 ignorieren wir vorerst. Wie man sieht werden hier alle Peers mit denen eine Kommunikation aufgebaut werden darf aufgelistet. Dieser wird jedes mal mit einem [Peer] eingeleitet, gefolgt von seinem PublicKey und den erlaubten IP Adressen mit denen kommuniziert werden darf.

Anschließend muss noch IPv4 forwarding erlaubt werden, um den Zugriff auf das gesamte LAN zu ermöglichen. Dazu ist folgender Befehl auszuführen:

```
$ vi /etc/sysctl.conf
```

und in folgender Zeile den Befehl:

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

auszukommenntieren.

Um diese Änderungen zu übernehmen kann entweder der Server mit dem Befehl *reboot* neugestartet werden, oder mit folgenen Befehlen ohne eines Neustarts übernommen werden:

```
sysctl -p
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Soweit so gut. Der Server ist nun konfiguriert.

WireGuard kann nun mit folgenden Befehl gestartet werden:

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

Möchte man, dass WireGuard automatisch gestartet wird, sobald der Server gestartet wird, kann man das mit folgenden Befehl erreichen:

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

### 1.3.3 Client

Zuerst muss WireGuard installiert werden. Dieser Vorgang ist in Abschnitt 1.2 erklärt. Für Ubuntu 19.10 habe ich folgenden Befehl ausgeführt:

```
1 $ sudo apt install wireguard
```

Anschließend wird für den Client ein private und public Key generiert.

```
1 wg genkey | tee client_private_key | wg pubkey > client_public_key
```

Daraus ergibt sich Folgendes:
**private key**: iGIUm+xAylOFtOLEcBKVWKAs29Y8Snj6QuNfN9C301E=
**public key**: xGshpOf4b256g1ZYxYfnvUq8sdzVKvc12pig2jUJoz4=
Nun muss noch in der Server Konfigurationsdatei in Zeile 11 der Public Key des Clients eingetragen werden.

Zuletzt wird wieder eine Konfigurationsdatei angelegt.

```
1 $ vi /etc/wireguard/wg0-client.conf
```

Diese hat folgenden Inhalt:

```
1 [Interface]
2 Address = 10.100.100.2/32
3 PrivateKey = iGIUm+xAylOFtOLEcBKVWKAs29Y8Snj6QuNfN9C301E=
4
5 [Peer]
6 PublicKey = oYnqorCOyXKWInOQBKCLu6avKZYtgdPItSNwyQgHGmo=
7 Endpoint = 192.168.56.103:51820
8 AllowedIPs = 0.0.0.0/0
9 PersistentKeepalive = 21
```

In Zeile 3 wird der Private Key des Clients eingetragen und in Zeile 6 der Public Key des Servers. Zeile 7 enthält zusätzlich noch die IP Adresse des Servers.
Wireguard ist nun am Client konfiguriert und kann mit folgenden Befehl gestartet werden:

```
1 $ sudo wg-quick up wg0-client
```

Überprüft man nun die IP Adresse des Clients, stellt man fest, dass die IP Adresse des WireGuard Servers zu sehen ist, anstelle der ursprünglichen IP Adresse des Clients. Das bedeutet, dass die IP Adresse des Clients nun nicht mehr öffentlich sichtbar ist und unser VPN Server korrekt funktioniert. Näher wird darauf in Abschnitt 1.4.1 eingegangen.

Führt man den Befehl

```
1 $ sudo wg-quick down wg0-client
```

aus erhält man wieder die ursprüngliche IP Adresse des Clients.

## 1.4 Technische Funktionalität

### 1.4.1 VPN

Als VPN (Virtual Private Network) bezeichnet man ein logisches privates Netzwerk, das auf einer öffentlichen Infrastruktur zugänglich ist. Ausschließlich Kommunikationspartner, die zu diesem privaten Netzwerk gehören, können miteinander kommunizieren.
Die Grundsätze von VPNs um eine sichere Kommunikation zu gewährleisten lauten:

- **Authentizität**
  Eigenschaft, die gewährleistet, dass ein Kommunikationspartner tatsächlich derjenige ist, der er vorgibt zu sein.
- **Vertraulichkeit**
  Schutz vor unbefugter Preisgabe von Information. Vertrauliche Daten dürfen ausschließlich den Kommunikationspartnern im privaten Netzwerk zugänglich sein.
- **Integrität**
  Sicherstellung der Korrektheit von Daten.

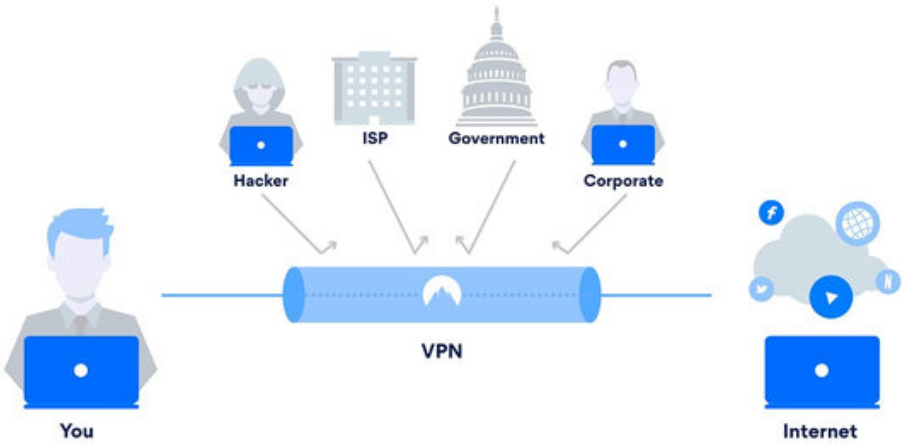


Abbildung 1.4: VPN Funktionsweise

#### Ablauf VPN-Verbindung

1. Ein VPN-Client stellt eine Verbindung zu einem VPN-Server über ein VPN-Protokoll her.
2. Der VPN-Server überprüft die Zugangsdaten, nimmt die Verbindung dementsprechend an und einigt sich mit dem VPN-Client auf einen Schlüssel zur sicheren Kommunikation.
3. Es entsteht eine verschlüsselte Verbindung zwischen dem VPN-Server und dem VPN-Client, die auch als *Tunnel* bezeichnet wird und eine physische Verbindung simuliert.
4. Der VPN-Client kann auf das lokale Netzwerk des VPN Servers zugreifen, als wäre er vor Ort mit diesem Netzwerk verbunden.

## 1.5 Wireguard

WireGuard arbeitet ausschließlich auf der 3. Schicht des OSI Modells und bietet eine Unterstützung für IPv4 und IPv6. Zur Übertragung der verschlüsselten IP-Pakete wird UDP verwendet.

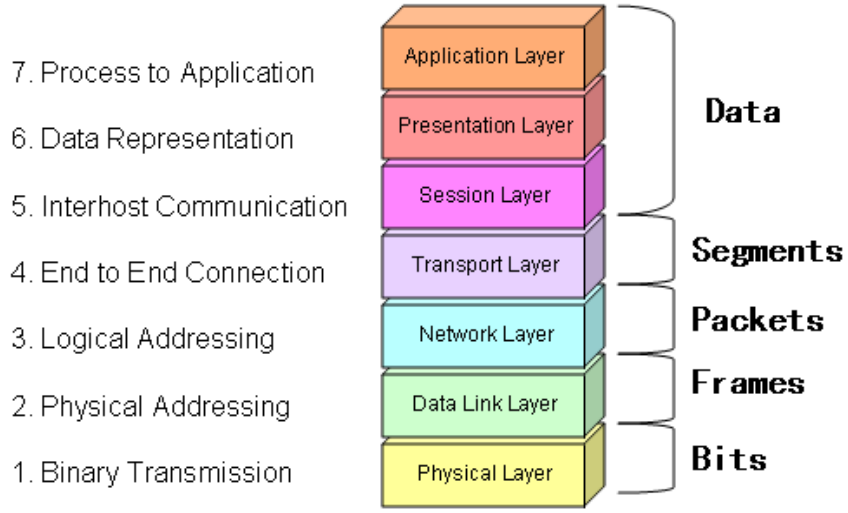


Abbildung 1.5: Osi Modell

Vorteile:

- einfache Nutzbarkeit
- minimale Codebasis
- Open Source
- moderne Kryptographietechniken
- Unterstützung vieler Betriebssysteme
- schneller Verbindungsaufbau
- schnelle Geschwindigkeit

Um ein hohes Sicherheitsniveau bei der Verschlüsselung der Daten zu erreichen, nutzt WireGuard moderne kryptographische Verfahren:

- Curve25519 zum Schlüsselaustausch im Handshake mit Elliptic Curve Diffie-Hellman (ECDH)
- ChaCha20 und Poly1305 für die symmetrische Verschlüsselung und den Austausch der Daten
- BLAKE2s für die Hashfunktion
- Ed25519 für die Zertifikate

Die Identität der VPN-Teilnehmer ist an ihren öffentlichen Schlüssel geknüpft. Damit können sich die Peers untereinander authentifizieren. WireGuard basiert auf einem Konzept namens *Cryptokey-Routing*. Dabei ist für jeden Peer eine Tabelle mit den public Keys und Allowed IPs seiner Gegenstellen enthalten. Daraus leitet WireGuard eine interne Routing-Tabelle ab, die den Weg für jedes Paket kennt. Verbindungen werden ähnlich wie bei SSH durch den Austausch der öffentlichen Schlüssel aufgebaut. Dabei setzt WireGuard nicht auf eine klassische Client-Server Architektur, sondern erzeugt ein Peer-to-Peer Netzwerk. Die Verbindung zwischen zwei Peers realisiert WireGuard über einen einzelnen, frei wählbaren UDP-Port. Wird kein Port angegeben, beginnt WireGuard bei 51820/UDP.

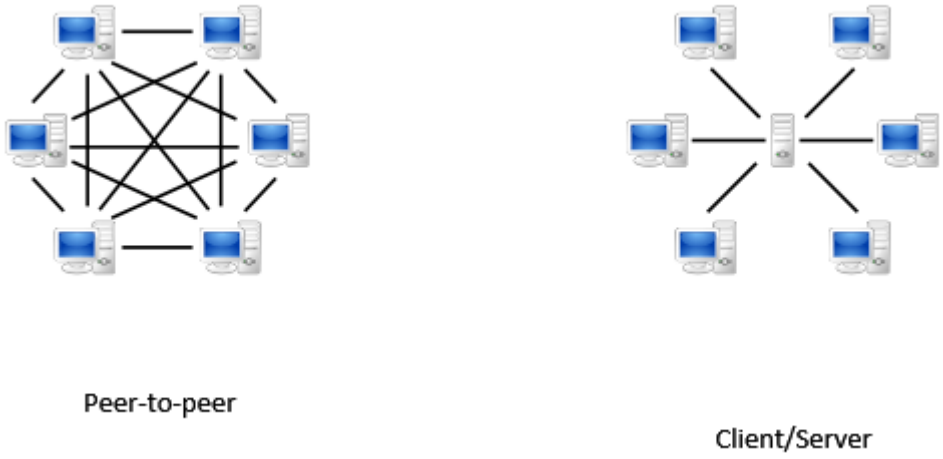


Abbildung 1.6: Peer to Peer vs. Client/Server Architektur

Beispiel für Verschlüsseln eines Pakets:

1. Es soll ein Paket an die IP Adresse 192.168.30.8 geschickt werden. WireGuard überprüft, welchem Peer diese IP Adresse entspricht.
2. Wird dieser Vorgang erfolgreich durchgeführt, erfolgt eine Verschlüsselung des IP Pakets mit dessen Public Key. Sonst wird das Paket verworfen.
3. WireGuard überprüft den öffentlichen Endpoint von diesem Client. Dieser ist beispielsweise Port 53133 mit 216.58.211.110.
4. Das verschlüsselte Paket aus Punkt 2 wird nun an den Endpoint 216.58.211.110:53133 durch UDP weitergeleitet.

Beispiel für Entschlüsseln eines Pakets:

1. Empfängt ein WireGuard Peer ein Paket, wird dieses durch den eigenen private Key entschlüsselt.
2. Wird dieser Vorgang erfolgreich durchgeführt und für einen bekannten Peer X authentifiziert, wird sein letzter öffentlicher Endpoint ermittelt.
3. Das entschlüsselte Paket enthält das Klartext Paket von der IP Adresse 192.168.43.89. Nun wird geprüft, ob der Peer X diesem Peer Pakete der IP Adresse 192.168.43.89 schicken darf.
4. Wenn die Prüfung erfolgreich ist, wird das Paket akzeptiert. Sonst wird das Paket verworfen.

Performance Tests (Abbildung 1.9) zeigen, dass WireGuard wesentlich leistungsfähiger als die Konkurrenten IPsec und OpenVPN abschließt. Im Gegensatz zu diesen Protokollen, welche hunderttausende LOC haben, enthält WireGuard bloß 4000 Zeilen. Dies ist aus zwei Gründen ein großer Sicherheits Vorteil: Zum einen enthält weniger Code auch potenziell weniger Fehler. Zum anderen sind dadurch potentielle Schwachstellen schneller zu finden. Einen weiteren Performance Schub bekommt WireGuard dadurch, dass die Software serverseitig als Linux-Kernelmodul ausgeführt wird. Außerdem wurde WireGuard unter der GPLv2 Lizenz veröffentlicht und ist somit kostenlos einsetzbar.

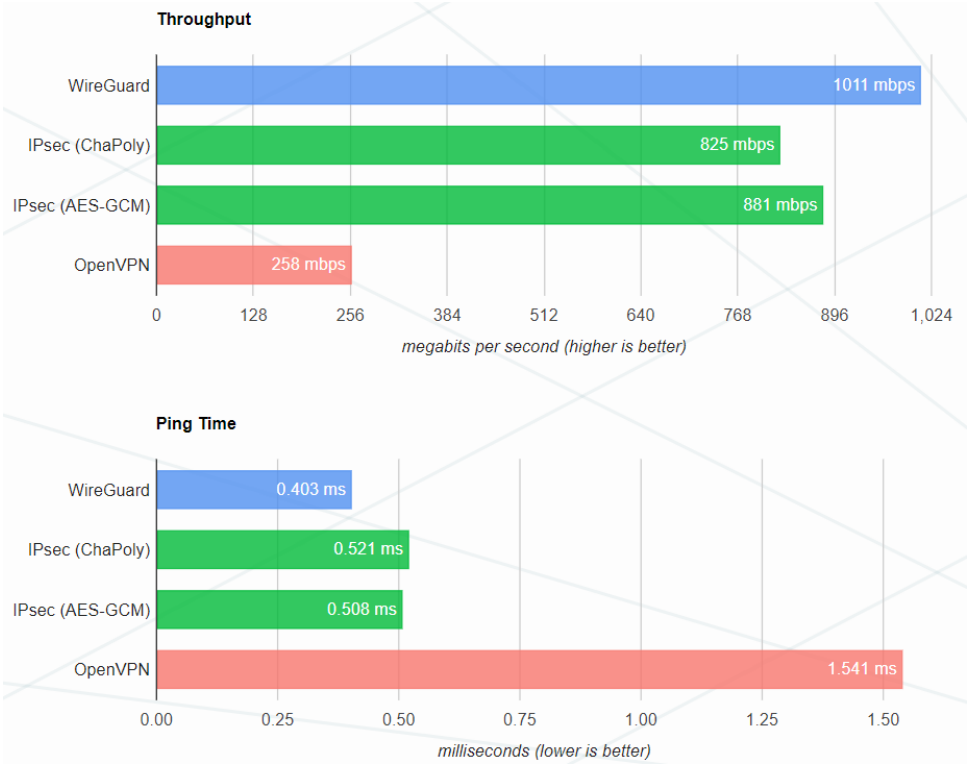


Abbildung 1.7: WireGuard Performancevergleich

Nachteile:

- experimentelle Entwicklung
  Obwohl WireGuard schon benutzbar ist, ist die Entwicklung einer stabilen Version noch nicht abgeschlossen und muss noch Sicherheitsaudits durchlaufen, in denen Experten die Codequalität überprüfen.
- eingeschränkte Anonymität
  WireGuard unterstützt keine dynamische Adressverwaltung, wodurch dem Client im Vorhinein eine vordefinierte IP Adresse zugewiesen werden muss, welche auf jedem Server mit dem entsprechenden Schlüssel verknüpft ist.

## 1.6 Docker

In der heutigen Zeit möchte man Software zwischen unterschiedlichen Plattformen und Infrastrukturen bewegen, ohne ständig die Vorraussetzungen für das Deployment und den Betrieb anzupassen.
Docker gibt uns die Möglichkeit mithilfe von Container-Technologie, Anwendungen in verschiedensten Umgebungen mit den unterschiedlichsten Vorraussetzungen auszuführen, ohne dafür das komplette System virtualisieren zu müssen.

## 1.7 Vorteile

Die Plattform bietet die Möglichkeit unsere Applikation in eine isolierte Umgebung zu verpacken, gennant *Container*. Container benötigen weit weniger Ressourcen als z.B. eine Virtuelle Maschine. Der Grund dafür, wird in der Erklärung der Container-Technologie näher erläutert. Diese Umgebung verändert sich nicht, was bedeutet, dass keine Inkonsistenzen auftreten und die Applikation immer in der exakt gleichen Umgebung läuft. Docker ist somit durch seine Container-Technologie im Hinblick auf Systemressourcen weitaus effizienter und modularer als eine Virtuelle Maschine.

### 1.7.1 Container Technologie

Docker basiert auf der Container-Technologie. Die Grafik 1.8 vergleicht eine Virtuelle Maschine mit einer Container Engine. Bei einer virtuellen Maschine wird ein Hypervisor benötigt, auf welchem die VMs laufen. In einer VM wird das gesamte Betriebssystem virtualisiert und auf diesem Betriebssystem läuft dann unsere Applikation. Die Container Technologie funktioniert so, dass auf dem Host Betriebssystem eine sogenannte Container Engine läuft (in unserem Fall die Docker Engine), auf der sich wiederum die Container befinden. Anstatt also die komplette Hardware zu virtualisieren, wie es bei VMs der Fall ist, werden bei einem Docker Container nur die benötigten Binaries/Libraries virtualisiert, welche für die Anwendung, die am Container laufen soll, benötigt werden.

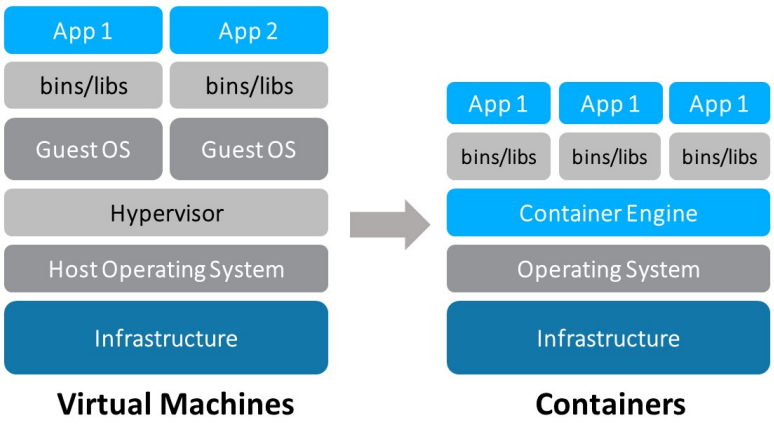


Abbildung 1.8: Docker vs VM

### 1.7.2 Image

Ein Image ist ein Template mit verschiedenen Konfigurationen für einen Container. Meist basiert ein Image auf einem anderen, enthält jedoch zusätzliche Anpassungen. Ein Image wird durch ein sogenanntes "Dockerfile" dargestellt.

### 1.7.3 Container

Ein Container ist eine Instanz eines Images.

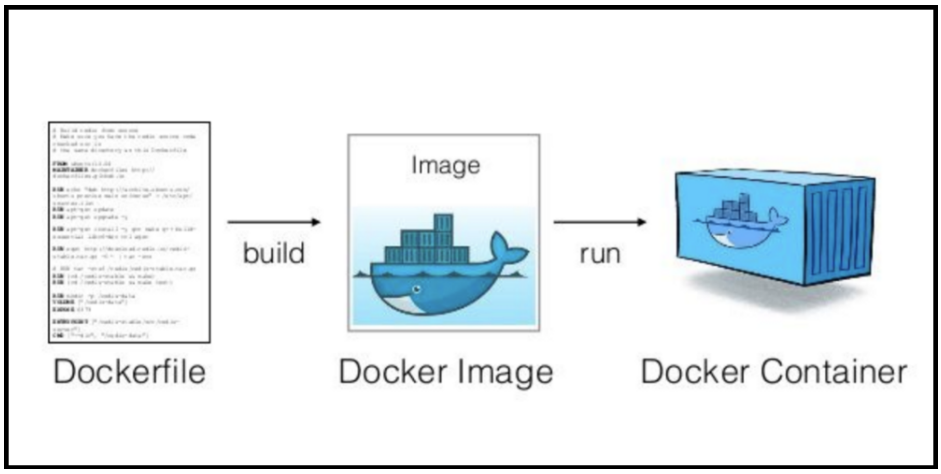


Abbildung 1.9: Docker Build Prozess

## 1.8 Einsatz im Container

Es besteht die Möglichkeit WireGuard in einem Docker Container zu verwenden. In den folgenden Schritten wird dieser Vorgang erklärt:

1. Docker installieren
2. docker-compose Datei erstellen

```
1   version: "2"
2   services:
3    vpn:
4     image: cmulk/wireguard-docker:buster
5     volumes:
6      - data:/etc/wireguard
7     networks:
8      - net
9     ports:
10     - 5555:5555/udp
11    restart: unless-stopped
12    cap_add:
13     - NET_ADMIN
14     - SYS_MODULE
15
16  networks:
17    net:
18
19  volumes:
20   data:
21    driver: local
```

3. Projekt builden

```
1     git clone https://github.com/cmulk/wireguard-docker.git
2     cd wireguard-docker
3     git checkout stretch
4
5     docker build -t wireguard:local .
6
```

4. Projekt zum 1. Mal ausführen

```
1     docker run -it --rm --cap-add sys_module -v /lib/modules:/lib/modules cmulk/
      wireguard-docker:buster install-module
2
```

Nach dem das Projekt bereits einmal ausgeführt wurde, erfolgt dies für die nächsten Runs mit dem Befehl:

```
1 docker run --cap-add net_admin --cap-add sys_module -v <config volume or host dir>:/
      etc/wireguard -p <externalport>:<dockerport>/udp cmulk/wireguard-docker:buster
```