Ziel ist es, das die VM aus dem ganzen Netzwerk erreichbar ist und nicht nur vom Host.

24.09. – 03.10.23

Vorab

Begrifflichkeiten:

- Host – Der reale Computer auf dem die VM ausgeführt wird
- VM – Virtual Machine – Der Virtuelle Computer
- enp1s0 – Der Netzwerkadapter mit Internetzugriff (bzw. realem Netzwerkzugriff), variiert und wird mit '#`ip a`' ermittelt und muss im Verlauf überall entsprechend angepasst werden. Ebenso sind die erwänten IP-Adressen überall individuell zu ermitteln, zu vergeben.
- Virtual Machine Manager – Grafisches Tool zum bedienen und konfigurieren der VM. Die Bedienung wird hier nicht erklärt.
- Strg+s – Tastenkombination...

Farben:

#`Befehl` - Der blaue Teil wird in einer geöffneten Konsole eingegeben

Internetlink / Informationsquelle

Testen ob darauf verzichtet werden kann

Siehe Begrifflichkeiten enp1s0

Los gehts

Basierend auf Debian 12.1.0

Auf dem Host mit Desktop (KDE) ist Debian nagelneu installiert

Host-Username: keeper ← Muss natürlich überall den eigenen Gegebenheiten angepasst werden

User keeper zu sudo hinzufügen:

Konsole öffnen
als su anmelden: #`su -l`
Rootpasswort eingeben
#`adduser keeper sudo`
Systemneustart: #`reboot`

```
#sudo apt-get install qemu-system-x86
#sudo apt-get install virt-manager
```

Test mit #`kvm` :D – wieder schließen

Damit eine Remoteverbindung durch den 'Virtual Machine Manager' aufgebaut werden kann und beim Start von 'Virtual Machine Manager' auf dem host keine Passworteingabe mehr nötig ist: #`sudo adduser keeper libvirt`

Qemu/libvirt/kvm soll als user ausführbar werden:
https://www.openeuler.org/en/blog/jzy0/2020-08-20-run-qemu-in-nonroot-mode.html

```
#sudo adduser keeper kvm
#sudo chmod 666 /etc/libvirt/qemu.conf
#kwrite /etc/libvirt/qemu.conf
```

Am Anfang einfügen (**selbst Tippen, nicht kopieren!**):

```
user = "keeper"
group = "keeper"
```

speichern.

```
#sudo systemctl restart libvirtd.service
```

Jetzt kann mit dem ‘Virtual Machine Manager’ eine VM erstellt und darauf ein Betriebssystem installiert werden. Dies ging auch vor der qemu.conf-Änderung schon, dann konnte qemu allerdings nicht auf den User-Ordner (/home/keeper/xyz) zugreifen, an dem das HD-Image und das Install.iso liegen sollen.

Der Einfachhalt halber, erst einmal eine VM mit grafischer Oberfläche
Lokal auf dem Host mit dem ‘Virtual Machine Manager’ erstellen und Debian installieren. Wenn die fertige VM Internetzugriff hat, weiter konfigurieren. Diese VM ist jetzt schon vom Host aus „an**ping**bar“.

Netzwerkkonfiguration auf dem Host:
https://spad.uk/really-simple-network-bridging-with-qemu/

Es wird eine freie IP-Adresse im Netzwerk benötigt, ggf. mit #Ping *w.x.y.z* ertesten. Im Verlauf, wenn eine IP auftaucht, die ermittelte IP dort eintragen.

```
#sudo apt-get install bridge-utils
```

Bridge = Netzwerkbrücke.

Temporär (Durch das kapern der bestandsbridge):

```
#sudo ip link set dev virbr0 down
#sudo brctl delbr virbr0
#sudo brctl addbr virbr0
#sudo brctl addif virbr0 enp1s0
#sudo ip addr add 192.168.178.44/24 dev virbr0
   //Ohne kommt der Host nicht mehr ins lokale Netz wenn VM gestartet
#sudo ip link set virbr0 up
#sudo iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT
```

FUNKTIONIERT ! → Aber so nach Host-Neustart alles weg.
Damit die Standartbridge virbr0 wieder auftaucht nach den Neustart:

```
#sudo virsh net-autostart default
```

Dauerhaft (Wir erstellen uns eine eigene Bridge → qemubr0. Wurde Temporär durchgeführt, erst einen #reboot durchführen):

Einen Service in systemd anlegen:

```
#kwrite /lib/systemd/system/qemu-network.service
```

Dort einfügen: {

```
[Unit]
Description=Qemu Netzwerkkonfiguration
After=network-online.target

[Service]
Type=oneshot
Restart=on-failure
ExecStart=brctl addbr qemubr0
ExecStart=brctl addif qemubr0 enp1s0
ExecStart=ip addr add 192.168.178.44/24 dev qemubr0
ExecStart=ip link set qemubr0 up
ExecStart=iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT

[Install]
WantedBy=multi-user.target
```

}

Strg+s → Passwort von keeper → kwrite schließen

Den neuen Service aktivieren:
#sudo systemctl enable qemu-network.service

Den Service dazu bringen das er vor dem Start der VM ausgeführt wird, damit die bridge auch schon bereit ist wenn Qemu damit arbeiten will:
#kwrite /lib/systemd/system/libvirtd.service
Dort unter die „After=*xyz*“-Zeilen eine weitere einfügen mit:
After=qemu-network.service
Strg+s → Passwort von keeper → kwrite schließen

Entweder Host neu starten oder #sudo systemctl start qemu-network.service um den neuen Service auszuführen bzw. die neue bridge zu erzeugen.

Da wir nun eine eigene Netzwerkbrücke erstellt haben, müssen wir diese der VM noch bekannt machen:
Im ‘Virtual Machine Manager’ in der Netzwerkkonfiguration der VM muss nun ‚Bridge‘ ausgewählt und bei Name ‚qemubr0‘ eingetragen werden.

That’s it!!!

Jede weitere VM kann jetzt parallel und gleichzeitig mit dieser qemubr0 arbeiten.

Helferlein:

Damit die VM automatisch bei Hoststart mit bootet in der VM-Konfiguration bei „Boot Options“ das Häkchen bei Autostart setzen.

Netzwerk neu starten:
#sudo /etc/init.d/networking restart

Grafische Netzwerkkonfiguration in der Konsole: #nmtui

Netzwerkbridges anzeigen:
#sudo brctl show

Basic-Netzwerkkonfiguration für statische IP (z.B. in der VM):
/etc/network/interfaces

unten einfügen: {
```
auto enp1s0
iface enp1s0 inet static
address 192.168.178.99/24
gateway 192.168.178.1

}
```
Falls schon eine Zeile mit enp1s0 vorhanden ist (meist DHCP), diese Zeile löschen.

Virsh:
#sudo virsh net-list –all

#sudo virsh net-start default

Systemd:

Tabelle 4: Wichtige Kommandos für Systemd

Beschreibung	Kommando
Aktive Service-Units auflisten	systemctl --type=service
Alle Service-Units auflisten	systemctl --type=service --all
Unit starten	systemctl start foobar.service
Unit stoppen	systemctl stop foobar.service
Status einer Unit anzeigen	systemctl status foobar.service
Unit beim Boot automatisch starten	systemctl enable foobar.service
Unit beim Boot nicht automatisch starten	systemctl disable foobar.service

Grafik von: www.linux-magazin.de

--