Subject: HowTo: Mehrere virtuelle Netzwerk-Adapter (veth) über bridge (persistent).
Posted by WebWusel on Sun, 29 Aug 2010 16:33:16 GMT
View Forum Message <> Reply to Message

Hallo,

ich nutze OpenVZ aus verschiedenen Gründen schon eine ganze Weile mit virtuellen Ethernet-Devices für die Container. Das Für und Wider soll hier nicht diskutiert werden.

Nun soll Shorewall zum Einsatz kommen und die von mir im Ursprung umgesetzte Art der VETH-Devices stammt noch aus der Zeit von vor 3.0.23 und hat sich bis jetzt so durchgeschlichen. Warum auch nicht, funktionierte ja schließlich.

Für den Einsatz mit einer Firewall bietet sich aber an, die VETH-Devices über eine Bridge zu „bundeln". Und wenn man einer VE mehrere eth-Devices, wie eth0 und eth1 (oder auch noch mehr) zuweisen möchte, die dann wiederum auch auf das interne und externe Netz zugreifen können, dann sind die meisten Dokus unzureichend.

Darum habe ich jetzt etwas gebastelt und platziere ein HowTo an dieser Stelle, nicht zuletzt damit ich mit meinem eigenen HowTo arbeiten kann und das Ganze damit dokumentiert ist.

Was ist gegeben?

Debian (Lenny) Hostsystem mit zwei Netzwerk-Devices:
eth0 mit der Anbindung an das WAN und eth1 mit der Anbindung an das LAN.
Wir nehmen an, dass wir über das Netz 11.22.33.0/25 im WAN verfügen können und nutzen für unsere interne Kommunikation das Netz 192.168.0.0/24.

Was ist gewollt?

Ist das Hostsystem einmal eingerichtet, sollen die Container ein bis zwei virtuelle Devices erhalten können. (Es ist relativ einfach, dieses HowTo auch für mehr als zwei virtuelle Devices abzuändern). Die Konfiguration der Devices soll dabei vollständig durch OVZ-eigene Mechanismen übernommen werden, bzw. bei vzctl start veid sollen alle nötigen Konfigurationen vorgenommen werden. Außerdem soll die Konfiguration sich auf die veid.conf sowie die /etc/network/interfaces des Containers beschränken.

Vorbereitung:

Ich gehe davon aus, dass OVZ auf dem Host bereits läuft. Es werden aber noch die bridge-utils benötigt. Dazu gibt man auf dem Host ein:

apt-get install bridge-utils

Und schon kann es losgehen.

Konfiguration des Hostsystems:

Jetzt wird das Netzwerk des Host konfiguriert. Dazu bearbeiten wir die /etc/network/interfaces auf diese Weise:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# Die Netzwerkkarte am WAN
auto eth0
iface eth0 inet static
        address 11.22.33.104
        netmask 255.255.255.128
        network 11.22.33.0
        broadcast 11.22.33.127
        gateway 11.22.33.1
# die folgenden Einstellungen sind nun wichtig für das richtige Routing
    up sysctl -w net.ipv4.conf.eth0.proxy_arp=1
    up sysctl -w net.ipv4.conf.eth0.forwarding=1
# die erste Bridge wird erstellt und konfiguriert
    up brctl addbr vmbr0
    up ifconfig vmbr0 0
    up sysctl -w net.ipv4.conf.vmbr0.proxy_arp=1

auto eth1
iface eth1 inet static
        address 192.168.0.24
        netmask 255.255.255.0
        network 192.168.0.0
        broadcast 192.168.0.255
# auch hier Einstellungen für das Routing
    up sysctl -w net.ipv4.conf.eth1.proxy_arp=1
    up sysctl -w net.ipv4.conf.eth1.forwarding=1
# die zweite Bridge wird erstellt und konfiguriert
    up brctl addbr vmbr1
    up ifconfig vmbr1 0
    up sysctl -w net.ipv4.conf.vmbr1.proxy_arp=1
```

Nach einem Neustart des Systems stehen nun unsere zwei Bridges (vmbr0 und vmbr1) zur Verfügung. Das hat vorerst keinen weiteren Einfluss auf unser Netzwerk und arbeitet wie bisher.

Nun ergänzen wir die OpenVZ-Scripte noch mit einem eigenen kleinen Script und speichern es als /usr/sbin/vznetaddbrroute mit diesem Inhalt:

```
#!/bin/bash
#script to bring up bridged routing in the CT0

CONFIGFILE=/etc/vz/conf/$VEID.conf
. $CONFIGFILE
VZHOSTIF=`echo $NETIF |sed 's/^.*host_ifname=\(.*\),.*$/\1/g'`

case "$1" in

        start)

if [ -n "$VETH_BR0_ADDR" ]; then

BRIDGE=vmbr0

for IP in $VETH_BR0_ADDR; do
    echo "Adding $IP to $BRIDGE"
    /sbin/ip route add $IP dev $BRIDGE
done

fi

if [ -n "$VETH_BR1_ADDR" ]; then

BRIDGE=vmbr1

for IP in $VETH_BR1_ADDR; do
    echo "Adding $IP to $BRIDGE"
    /sbin/ip route add $IP dev $BRIDGE
done

fi
;;

        stop)

if [ -n "$VETH_BR0_ADDR" ]; then

BRIDGE=vmbr0

  for IP in $VETH_BR0_ADDR; do
    echo "Removing $IP from $BRIDGE"
    /sbin/ip route del $IP
done

fi

if [ -n "$VETH_BR1_ADDR" ]; then
```

```
BRIDGE=vmbr1

  for IP in $VETH_BR1_ADDR; do
    echo "Removing $IP from $BRIDGE"
    /sbin/ip route del $IP
done

fi

;;

esac

echo "Skript ausgeführt als...">/var/log/neteth.log
echo "VZHOSTIF= $VZHOSTIF">>/var/log/neteth.log

exit 0
```

Möchte man mehr als zwei Bridges einsetzen, so kopiert man sich entsprechend die if-Blocks und setzt jeweils die Variable $BRIDGE entsprechend der eigenen Einstellungen in der /etc/network/interfaces neu.

Als nächstes muss die /etc/vz/vznet.conf angelegt oder modifiziert werden und mit dieser Zeile gefüllt werden:

```
EXTERNAL_SCRIPT="/usr/sbin/vznetaddbr"
```

Nicht wundern, dass hier nicht unsere vznetaddbrroute steht, die wird an anderer Stelle aufgerufen, nämlich hier:

Ich habe mich gefragt, wie bekomme ich jetzt die Routen mit einem vzctl start VEID eingerichtet und beim einem Stop wieder entfernt? Dazu kann man hervorragend die VEID.mount und VEID.umount Skripte verwenden. Es handelt sich hierbei tatsächlich um Skripte und nicht, wie sonst im Verzeichnis /etc/vz/conf um Konfigurationsdateien. Darum nicht vergessen das Execute-Bit auf die Dateien zu legen, nachdem diese angelegt wurden!

Nun besteht die Möglichkeit pro Container ein .mount und .umount Script anzulegen. Das erhöht aber wieder den administrativen Aufwand. Deshalb kann man statt VEID.mount und VEID.umount auch vps.mount und vps.umount verwenden. Im letzteren Fall werden diese Skripte für alle eingerichteten Container beim Start oder Stop ausgeführt. Ich habe mich daher für diese bequeme Möglichkeit entschieden und lasse (wie oben im vznetadbrroute gesehen) das aufgerufene Skript mit der Prüfung „if [! -n "$VETH_BR0_ADDR"]; then" entscheiden, ob es Routen erstellen soll oder nicht.

Wir legen also zwei Dateien unter /etc/vz/conf wie folgt an:

Datei vps.mount:

```
#!/bin/bash
/usr/sbin/vznetaddbrroute start $VEID
```

Datei vps.umount:

```
#!/bin/bash
/usr/sbin/vznetaddbrroute stop $VEID
```

Eigentlich müsste $VEID gar nicht übergeben werden. Ich hatte allerdings noch andere Ideen, die ich aber noch nicht umgesetzt habe. Also, es geht auch ohne $VEID

Damit sind die grundsätzlichen Arbeiten an OpenVZ abgeschlossen. Ab sofort kann OpenVZ auch nach einem Host-Neustart mit VETH over Bridge dauerhaft umgehen.

Konfiguration des Containers

Auch hier gehe ich von einem Debian-Gastsystem aus, wobei dies nur für den Abschnitt „Netzwerkkonfiguration des Containers" zutrifft. Hier ist dann entsprechend der eingesetzten Distribution entsprechend anzupassen!

Nachdem ein neuer Container angelegt wurde, liegt unter /etc/vz/conf eine Konfigurationsdatei für eben den angelegten Container.

Als Beispiel konfiguriere ich jetzt den Container mit der VEID 101 mit folgendem Netzwerk:

eth0 mit den Adressen 11.22.33.111, 11.22.33.112, 11.22.33.113, und 11.22.33.114.
eth1 mit den Adressen 192.168.0.111 und 192.168.0.112

ACHTUNG! Wir starten den Container jetzt NICHT!

Es besteht die Möglichkeit, die virtuellen Ethernet-Devices mit von OpenVZ generierten MAC-Adressen zu versehen, oder durch eigene, selbst gewählte oder erstellte. Ich bevorzuge aus organisatorischen Gründen die Variante mit eigenen MAC-Adressen. Dazu gibt es ein nettes Skript und dieses kann hier herunter geladen werden:
http://www.easyvmx.com/downloads.shtml

Nachdem wir das Skript auf den Server geladen haben, generieren wir im Verzeichnis von easymac.sh mit ./easymac.sh -R eine zufällig generierte MAC-Adresse. Ich erhalte jetzt z. B. „00:0C:29:9A:D0:02".

Nun benötigen wir für einen Container mit zwei virtuellen Netzwerk-Devices vier MAC Adressen. Dazu nehme ich die generierte und baue noch weitere drei Adressen auf diese Weise:

Für eth0 im Container und veth101.0 auf dem Host:

00:0C:29:9A:D0:02
00:0C:29:9A:D0:03

Für eth1 im Container und veth101.1 auf dem Host:

00:0C:29:9A:D1:02
00:0C:29:9A:D1:03

Jetzt müssen die VETH-Devices dem Container mit Hilfe von vzctl bekannt gemacht werden. Dazu sind nun folgende Eingaben erforderlich:

vzctl set 101 --netif_add eth0, 00:0C:29:9A:D0:03,veth101.0,00:0C:29:9A:D0:02,vmbr0 --save
und
vzctl set 101 --netif_add eth1, 00:0C:29:9A:D1:03,veth101.1, 00:0C:29:9A:D1:02,vmbr1 --save

Außerdem müssen wir dem Config-File noch sagen, welche IP-Adressen der Container nutzen wird. Dazu wird /etc/vz/101.conf mit folgenden Zeilen ergänzt:

CONFIG_CUSTOMIZED="yes"
VETH_BR0_ADDR="11.22.33.111 11.22.33.112 11.22.33.113"
VETH_BR1_ADDR="192.168.0.111 192.168.0.112"

Ich hätte sicherlich auch die /etc/network/interfaces des Containers durch ein Skript auslesen und damit die Adressen automatisch ermitteln lassen können, die dann in /usr/sbin/vznetaddbrroute umgesetzt würden. Das hätte aber zur Folge, dass jeder Nutzer, der einen Container nutzt, eben beliebige IP's verwenden könnte und somit u. U. das eigene Netz zerschießt. Das können wir so verhindern, weil so nur die aufgeführten IP's geroutet werden.

Ich bin nicht sicher, ob CONFIG_CUSTOMIZED wirklich notwendig ist, aber es schadet auch nicht.

Ich habe mir übrigens (weil ich grundsätzlich sehr bequem bin) ein nettes EXCEL-Arbeitsblatt gebastelt, in dem ich die entsprechenden Parameter eintrage und dann per Copy and Paste die notwendigen Einträge einfach in die Datei schreiben lassen kann. Wer Interesse daran hat, kann mich gerne kontaktieren.

Netwerkkonfiguration des Containers:

Und schließlich bearbeiten wir noch die /etc/network/interfaces des Containers. Ich gehe davon aus, dass jeder selbst weiß, wo die private-Verzeichnisse seiner Container liegen und gebe daher nur den Inhalt der Interfaces unseres Beispiel-Containers an:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 11.22.33.111
        netmask 255.255.255.128
        network 11.22.33.0
```

```
        broadcast 11.22.33.127
        gateway 11.22.33.1

auto eth0:1
iface eth0:1 inet static
        address 11.22.33.112
        netmask 255.255.255.0
        network 11.22.33.0

auto eth0:2
iface eth0:2 inet static
        address 11.22.33.113
        netmask 255.255.255.0
        network 11.22.33.0

auto eth0:3
iface eth0:3 inet static
        address 11.22.33.114
        netmask 255.255.255.0
        network 11.22.33.0

auto eth1
iface eth1 inet static
        address 192.168.0.111
        netmask 255.255.255.0
        network 192.168.0.0

auto eth1:0
iface eth1:0 inet static
        address 192.168.0.112
        netmask 255.255.255.0
        network 192.168.0.0
```

Alles andere, was vorher von OpenVZ hier reingeschrieben wurde, fliegt raus!

Das war's!

Container starten und glücklich sein!

Ich hoffe damit eine klare und einfache Anleitung zur Erstellung von Containern mit persistenten virtuellen Netzwerk-Devices gebaut zu haben.

Ich erlaube mir ein Cross-Posting im Debian-Forum.

Konstruktive Kritik ist willkommen!

Viel Spaß damit,

Oskar

Edit: vznetaddbrroute korrigiert.