10. Okt. 2024

Die Version des Orginalskripts, von der ich ausgegangen bin, ist von Mitte Februar 2024 und findet sich in der Datei `original_provider.sh`.
Der Ordner `streams` enthält ein Beispiel-Setup, an dem man ausprobieren kann, wie der Network Creator arbeitet. Dabei handelt es sich um das denkbar einfachste Setup, nämlich ohne VRF, EVPN oder VXLAN's. Es wird auch keine Verbindung zwischen den Providern hergestellt, sondern es geht vorläufig nur darum, die ISP's aufzusetzen. Damit ist es auch für Neueinsteiger, Anfänger, Schüler und Auszubildende gut geeignet. Komplexere Setups sind natürlich für die Zukunft geplant. Wer solange nicht warten möchte, findet im Ordner `changed_and_new` alle geänderten und neu hinzugekommenen Dateien, sodass er sie in seinen eigenen Kontext integrieren kann.
Obwohl letztlich pro Rechner nur ein Node und Provider im Cluster laufen sollen, findet hier im vereinfachten Setup alles auf nur auf einem einzigen Node statt, was den Vorteil hat, dass man auch vorläufig nur einen Rechner, einen DHCP-Server und eine pfSense braucht.

Systemvoraussetzungen

Ein aus 1 pfSense, 1 DHCP-Server und 9 Vyos-Routern bestehender Provider benötigt im fertig aufgesetzem Zustand je nach Provider 9 bis 10 GB RAM. Dabei habe ich die RAM-Größen ziemlich minimalistisch gewählt.
Um ein Upgrade durchführen zu können, benötigt ein Vyos-Router allerdings erheblich mehr RAM als im Normalbetrieb, nämlich knapp 1,5 GB im Gegensatz zu knapp 600 MB. Da während der Installation auch immer ein Upgrade durchgeführt wird, muss bei einem Rechner mit nur 16 GB RAM (egal, ob mit oder ohne SWAP) eine Beschränkung auf die Erstellung von höchstens 6 Routern pro Durchlauf eingehalten werden. Diese Beschränkung kann auch durch Nutzung des *Serial Mode* nicht umgangen werden. Für die Erstellung der restlichen 3 Router gibt es den *Single Router Creator*, der eine Nachinstallation einzelner Router erlaubt. Dabei können auch mehrere Router gleichzeitig abgearbeitet werden, indem in mehreren Terminals der *Single Router Creator* mehrmals (versetzt) parallel gestartet wird.
Ab 20 GB RAM dürfte gerade so genug RAM vorhanden sein, um alle 9 Router in einem Durchgang aufzusetzen (von mir allerdings nicht getestet). Dabei am besten mit SWAP und einer hohen Swappines, z.B:

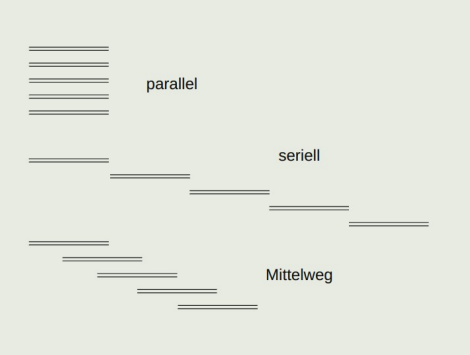
```
nano /etc/sysctl.conf  =>

# Set swap usage level
vm.swappiness=90

reboot
```

Achtung: Niemals SWAP auf SD's oder SSD's anlegen, weil sie dadurch kaputt gehen. (Dasselbe gilt möglicherweise auch für NVME's.) Sondern für SWAP immer HDD's verwenden!

Mit 32 GB RAM ist man auf jeden Fall auf der sicheren Seite, wenn man alle 9 Router in einem einzigen Durchlauf aufsetzen will.

Übersicht

Der *Provider Creator* und der *Turbo Provider Creator* arbeiten beide insofern parallel, als die Upgrade- und die Konfigurations-Tasks unter Ansible immer alle aufzusetzenden Router in jeweils einem einzigen Durchgang abarbeiten. Der *Serial Mode* ist für besonders schwache CPUs gedacht, für die diese Art der parallelen Verarbeitung eine zu große Belastung darstellt. Im *Serial Mode* werden die Upgrade- und die Konfigurations-Tasks immer nur für einen einzigen Router durchgeführt, d.h. der nächste Router kommt (innerhalb eines Playbooks) erst dann dran, wenn das Playbook für den vorigen Router abgeschlossen ist. Diese Vorgehensweise setzt die CPU-Auslastung sehr erheblich herab, was aber natürlich auch länger dauert.
Der *Serial Mode* ist insofern eine serielle Abarbeitung bzgl. Upgrade und Konfiguration. Den *Single Router Creator* mehrmals hintereinander laufen zu lassen, wäre hingegen serielle Abarbeitung bzgl. des gesamten Erzeugungsprozesses eines Routers. Ein Mittelweg zwischen paralleler und serieller Verarbeitung bezogen auf den gesamten Erzeugungsprozess ergibt sich, wenn man den *Single Router Creator* mehrmals versetzt parallel startet. Natürlich liegt der Gedanke nahe, diese Funktion im Rahmen eines *Multiple Router Creators* zu programmieren, in dem z.B. die Router 7 bis 9 zur Erstellung ausgewählt werden können, sowie ein Verzögerungswert, der den zeitlichen Abstand festlegt, mit dem die Single-Router-Skripte für diese Router versetzt parallel aufgerufen werden.



Da dies aber mit einer erheblichen programmiertechnischen Komplexitätssteigerung verbunden sein dürfte, habe ich das für das gegenwärtige Anfangsstadium als unverhältnismäßig befunden und deshalb vorläufig weggelassen. Außerdem ist es für den Anwender leichter verständlich, den *Single Router Creator* zunächst händisch multipel laufen zu lassen, um dann später den Unterschied zwischen einem automatisierten *Multiple Router Creator* und dem *Provider Creator* leichter nachvollziehen zu können.
Ein Mittelweg – also eine begrenzte Parallelität – bzgl. des gesamten Erzeugungsprozesses ist aber noch nicht das, wo es letztlich hingehen soll, sondern nur der Lösungsansatz: Letztlich soll ein Mittelweg pro jeweils einzelnem Unterprozess per jeweils spezifischem Delay eingestellt werden können, denn Unterprozesse, die wenig CPU-Leistung benötigen, sollen umso stärker parallelisiert, d.h. mit einem geringerem Verzögerungswert ausgeführt werden. Das heißt, man braucht letztlich für jeden einzelnen Unterprozess ein Skript bzw. Playbook, das nur einen einzigen Router bearbeitet plus einen mit diesem Unterprozess verbundenen Verzögerungsparameter, mit dem dieser eine Unterprozess dann für *alle* zu erstellenden Router versetzt aufgerufen wird. Und das dann für *alle* Unterprozesse.

Die Unterschiede zwischen dem Provider Creator und dem Turbo Provider Creator

Das provider_turbo.sh entspricht im Wesentlichen Gerds ursprünglichem provider.sh, das hier unter dem Namen `original_provider.sh` beigefügt ist.
Im provider_turbo.sh gibt es länger anhaltende Lastspitzen insbesondere in der ersten Start-Phase, der Upgrade-Phase und den Reboot-Phasen.

Die langen Lastspitzen beim ersten Start und den Start-Teilen der Reboots ließen sich durch einen frei setzbaren Start Delay beheben, der in einer wählbar langen sleep-Phase am Ende jedes Loop-Durchlaufs besteht. Die Delaywerte werden in der GUI in Sekunden angegeben.

Sehr einfach war auch die Verlangsamung des Shutdowns: Am Ende des `vyos_update_turbo.yml` wird mittels:

```
- name: reboot
  cli_command:
    command: "reboot now"
```

ein stark parallel stattfindender Bulk-Shutdown und -Restart ausgelöst, der eine erhebliche, anhaltende Lastspitze erzeugt. Das habe ich im `provider.sh` durch einen Shutdown mittels einem kurzen Shutdown-Skript ersetzt (ks steht für *kurze Skripte*):

```
sudo  bash ${HOME}/streams/ks/shutdown_isp$2.sh $4
```

Der Loop im kurzen Shutdown-Skript läuft in einem für schwache Rechner genau richtigen, langsamen Tempo, sodass ein Delay-Parameter gar nicht mehr nötig ist.

Was den Kopiervorgang beim Upgrade betrifft, wurde das ansible-spezifische `net_put`, das im `upgrade_turbo.sh` verwendet wird, im upgrade.sh durch ein scp ersetzt, wodurch es möglich wurde, in jeder Iteration eine kleine Pause einzubauen:

```
hosts:
    - router

  tasks:
    - name: "Copying {{vyos_file}} to system"
      net_put:
        src: "{{ vyos_dir }}{{ vyos_file }}"
        dest: "{{ vyos_file }}"
```

gegenüber

```
 tasks:
   - name: "Copying {{ vyos_file }} to system via scp"
     shell: "scp {{ vyos_dir }}{{ vyos_file }} {{ ansible_user }}@{{ ansible_host
}}:/home/{{ ansible_user }}/{{ vyos_file }} ; sleep 2"
     delegate_to: localhost   # scp wird auf dem Steuerungsrechner ausgeführt
     become: no   # Keine erhöhten Berechtigungen
     vars:
       ansible_ssh_pass: "{{ ansible_password }}"
```

Das Ansible-Pausenmodul hatte sich zu diesem Zweck nicht als hilfreich erwiesen.

Sonstige Anmerkungen

(1) Die Datei `create-vm-vyos.sh` habe ich dahingehend geändert, dass bei einem Neustart des PVE kein automatischer Start aller Router erfolgt, sondern die Router können über die Start-Funktion unter *General* mit einstellbarer Verzögerung gestart werden, wodurch zu große Lastspitzen bei schwachen Rechnern vermieden werden können.

(2) Die Limitierung im *(Turbo) Provider Creator* gilt ausschließlich für Ansible, nicht aber für die `provider(_turbo).sh` bzw. die `create-vm-vyos(-turbo).sh` Skripte. Das bedeutet, dass ein Limit von z.B. `"-l p1r7v,p1r8v,p1r9v"` (bei 3 zu erstellenden Routern) keinen Sinn macht, denn es werden dann die ersten 3 Router des jeweiligen Providers erstellt und Ansible wird dann vergeblich versuchen die Router 7 – 9 upzugraden und zu konfigurieren. Beim *Provider Creator*, dem *Turbo Provider Creator* und dem *General*-Fenster zählt die eingegebene Anzahl von Routern immer von Router 1 (r1v) an bis zur jeweils eingebenen Zahl (einschließlich).

(3) Der *Network Creator* wird im Terminal durch den Befehl `python3 nwc.py` aufgerufen. (Dazu sind `xorg` und eine `ssh -X` Verbindung erforderlich.) Es empfiehlt sich, in der .bashrc folgenden Alias zu setzen:

```
alias nwc='python3 nwc.py'
```

Abbrechen kann man den *Network Creator* durch Schließen des Terminals oder (das funktioniert m.W. aber nur im Gnome-Terminal) durch:

```
strg+C
Return
Rechts oben auf den x-Knopf zum Schließen klicken
Schließen abbrechen
```

Die letztere Methode hat den Vorteil, dass man im .venv bleibt.

(4) Außer im General-Fenster können die Provider- bzw. der Router Creator wahlweise mittels Mausklick und Returntaste gestartet werden. Im General-Fenster geht nur der Mausklick, weil es mehrere Go-Tasten gibt.
Bevor es dann losgeht, muss im Terminal noch das sudo-Passwort eingegeben werden. Wenn man das nicht so oft machen will, kann man den Zeitraum für die Gültigkeit dieser Eingabe (per default 15 Minuten) folgendermaßen erhöhen:

```
sudo visudo
Defaults timestamp_timeout=60
```

Das empfiehlt sich insbesondere bei Nutzung des *Serial Mode*, der normalerweise deutlich länger als 15 Minuten braucht.

(5) In den Limit-Zeilen können, sowohl mit der Back-Taste als auch mittels der entf-Taste, die sich im Hauptbereich der Tastatur (oben) befindet, Text gelöscht werden. Die entf-Taste rechts unten funktioniert dagegen nur im Provider Creator-Fenster, nicht aber im Turbo Provider Creator-Fenster.

(6) Wie anfangs bereits erwähnt, ist das hier gegebene Setup der Einfachheit halber für eine Ausführung auf einem einzigen PVE-System gedacht. (Trotzdem kann es natürlich auch zur Vorbereitung einer Clusterbildung auf drei Nodes verwendet werden, was letztlich auch der eigentlich Zweck ist!) Es ist dabei aber zu beachten, dass kontraintuitiverweise die Zahl, die für

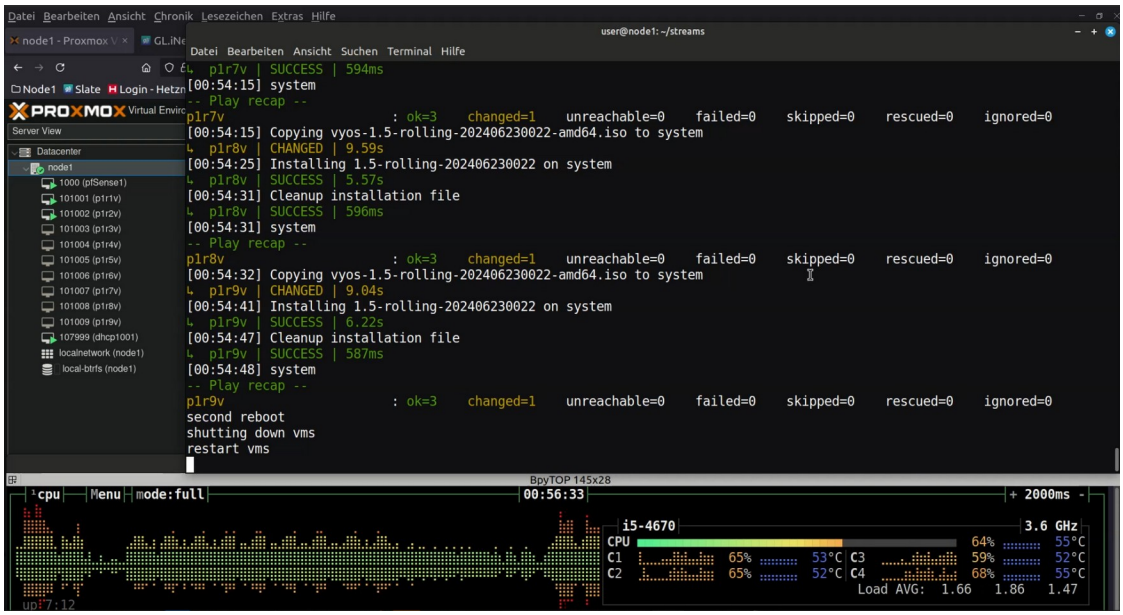
Node in der GUI eingegeben wird, immer gleich der Zahl sein muss, die für *Provider* eingegeben wird. Also auch, wenn man in einem PVE arbeitet, das zufälligerweise z.B. "node1" heißt, muss für *Node* die Zahl 3 eingegeben werden, wenn man *Provider* 3 erstellt. Das liegt daran, dass die Skripte so gemacht sind, dass sie letztlich eben doch in einem Cluster von drei Nodes ausgeführt werden.

(7) Der *Network Creator* übergibt generell den Destroy-Wert 1 an die Skripte, die er jeweils aufruft, d.h. jeder Router, der erstellt werden soll, wird vorher zu zerstören versucht, selbst wenn er gar nicht existiert. In diesem Fall wird eine Meldung ausgegeben, dass eine Konfiguration für diesen (nicht existierenden) Router nicht gefunden wurde, was bedeutet, dass der Destroy-Prozess übersprungen wird:

```
unable to find configuration file for VM 101001 on node 'node1'
Configuration file 'nodes/node1/qemu-server/101001.conf' does not exist
```

Was ist eine optimale CPU-Auslastung?

Das entscheidet natürlich letztlich jeder selber. Mein Ansatz sieht dabei so aus:
Das Grundprinzip ist, dass *idealerweise* eine Auslastung von 100% möglichst selten bis gar nicht vorkommen sollte und wenn dann nur ganz kurz, ansonsten aber eine Auslatung von über 90% sooft und so lange wie möglich angestrebt wird, damit die Erzeugung eines ISP's auch nicht länger dauert als nötig. Wenn man sich daran hält, gibt es immer noch genügend Phasen, in denen die Auslastung auf 50%, 30% oder noch weniger abfällt. Zur Überwachung der CPU-Auslastung und -Temperatur hat sich *bpytop* als gut geeignet erwiesen:



Erfahrungswerte

Hier einige Erfahrungswerte aus meinen Testläufen, um einen ungefähren Eindruck von Größenordnungen zu bekommen:

J4115 Es konnte nicht mal ein einziger Router erstellt werden. => Programm stürzt ab.

i5-4570 (Kein Serial Mode)
9 Router: Start Delay mindestens **12**
6 Router: Start Delay mindestens **6**
Bei Upgrade, Konfiguration und (Re-)Start teilweise ziemlich lang anhaltende CPU-Auslastung von bis zu 100%, allerdings bei geringer Erwärmung der CPU, sodass eine Erstellung von 9 Routern durchaus noch möglich ist. Bei schwächeren CPU's sollte für 9 Router in einem einzigen Durchgang der *Serial Mode* und ein höheres Start Delay gewählt werden.

i7-4770 (Kein Serial Mode)
9 Router: Start Delay mindestens **7**
6 Router: Start Delay mindestens **3**
Unproblematische CPU-Auslastung und niedrige Temperaturen meist zwischen 50 und 60 Grad.

i7-8700 (Kein Serial Mode)
9 Router: Start Delay mindestens **5**
Geringe CPU-Auslastung: Erreicht kaum 80%, meistens zwischen 50% und 60%, aber sehr starke Erwärmung bei Upgrade und (Re-)Start auf bis zu 87 Grad wegen häufigem Turbo Boost der Taktrate auf 4,3 Mhz.
Im ***Turbo Mode*** ist wegen starker Erwärmung (kurz sogar auf 91 Grad) von einer Erstellung von mehr als 4 Routern in einem einzigen Durchgang definitiv abzuraten, sofern der Turbo Boost für die Taktung im BIOS zugelassen ist. Nachdem ich den Turbo Boost im BIOS deaktiviert hatte, sodass eine Taktung von maximal 3,2 MHz eingehalten wurde, konnten 9 Router in einem einzigen Durchgang erstellt werden, ohne dass die CPU heiß geworden wäre! In den Reboot-Phasen wurde zwar über längere Zeit eine CPU-Auslastung von 100% erreicht, was m.E. etwas mehr ist, als wünschenswert, die Temperatur blieb aber selbst da immer deutlich unter 70 Grad, sodass also die parallele Erstellung von 9 Routern in einem einzigen Durchlauf im *Turbo Mode* auf dem i7-8700 durchaus noch machbar ist (vorausgesetzt der Turbo Boost ist deaktiviert!).

Fazit: Die i7-8700 wäre somit die Grenze für die CPU-Mindestanforderung für die Erstellung von 9 Routern im *Turbo Mode* in einem einzigen Durchgang, während die i5-4570 die Grenze für die Mindestanforderung im *Normalmodus* ist. Alles darunter erfordert den *Serial Mode,* wobei natürlich klar ist, dass eine i3 CPU der neuesten Generation mit mehr Kernen und Cache dann *über* einer i5-4570 einzuordnen ist.
'CPU-Mindestanforderung' bedeutet hier natürlich bzgl. einer vernünftigen CPU-Auslastung und Erwärmung. Man könnte natürlich auch z.B. eine i5-4570 CPU im Turbo Modus laufen lassen. Theoretisch würde sie dann schlimmstenfalls zu Throtteln anfangen, d.h. die Taktfrequenz absenken. Praktisch ist es jedoch hinsichtlich der Lebensdauer einer CPU nicht zu empfehlen, sie über lange Strecken an ihre Überhitzungsgrenze zu bringen! Um genau das zu verhindern, gibt es die Delays und den Serial Mode.

Ideen zu möglichen Weiterentwicklungen

(a) Einbindung von Mikrotiks.

(b) Automatisches Aufsetzen der pfSense und des DHCP-Servers. Die Skripte im Ordner Skripte sind ein erster Schritt in diese Richtung.

(c) Automatisches Erstellen der vyos.qcow2 und der seed.iso .

(d) Eine Funktion, die vor dem Update die jeweils neueste Version von Vyos rolling automatisch runterläd und unter /home/user/ansible/vyos-files/ ablegt; ansonsten muss man das händisch machen.

(d) Trotz guten Zuredens meinerseits war ChatGPT nicht in der Lage, die *Create Provider*-, *Create Provider Turbo*-Knöpfe usw. in einen leichten Kontrast (z.B. pseudo-3D-mäßig) zum Hintergrund zu designen. Immerhin werden die Knöpfe aber (meistens) farbig, wenn ihre Funktion gerade ausgeführt wird. Wenn man genau hinsieht, sind noch weitere kleine Schönheitsfehler zu erkennen, aber ich will damit auch keinen Designwettbewerb gewinnen.