19. 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.
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: RAM

Ein aus 1 pfSense, 1 DHCP-Server und 9 Vyos-Routern bestehender Provider benötigt im fertig aufgesetztem 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 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.
Ab 20 GB RAM dürfte gerade so genug RAM vorhanden sein, um alle 9 Router in einem Durchgang aufzusetzen (von mir allerdings nicht getestet).
Den Network Creator niemals mit Swap verwenden, weil das leicht zu Programmabstürzen führt.

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.

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 `python app.py` aufgerufen. Es empfiehlt sich, in der .bashrc folgenden Alias zu setzen:

```
alias nwc=‘python app.py’
```

Abbrechen kann man den *Network Creator* durch `strg+C .`

(4) 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.

(5) 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
```

(6) *Vermutlich* ist es bei der rein lokalen Nutzung im Heimnetz gar nicht nötig, die Datei `generate_secret_key.py` in VSCode auszuführen und den Secret Key in Zeile 6 von `app.py` einzufügen:

```
6  app.secret_key = 'y0ur$ecr3t_k3y_w1th_r@ndomCh@racters'
```

Wenn man das aber aus irgendwelchen Gründen machen wollen würde, wird man den Secret Key nicht im Klartext ins Skript schreiben, sondern muss ihn in einer Umgebungsvariable ablegen und diese persistent machen und zwar so:

Zeile 6 im Skript ändern zu

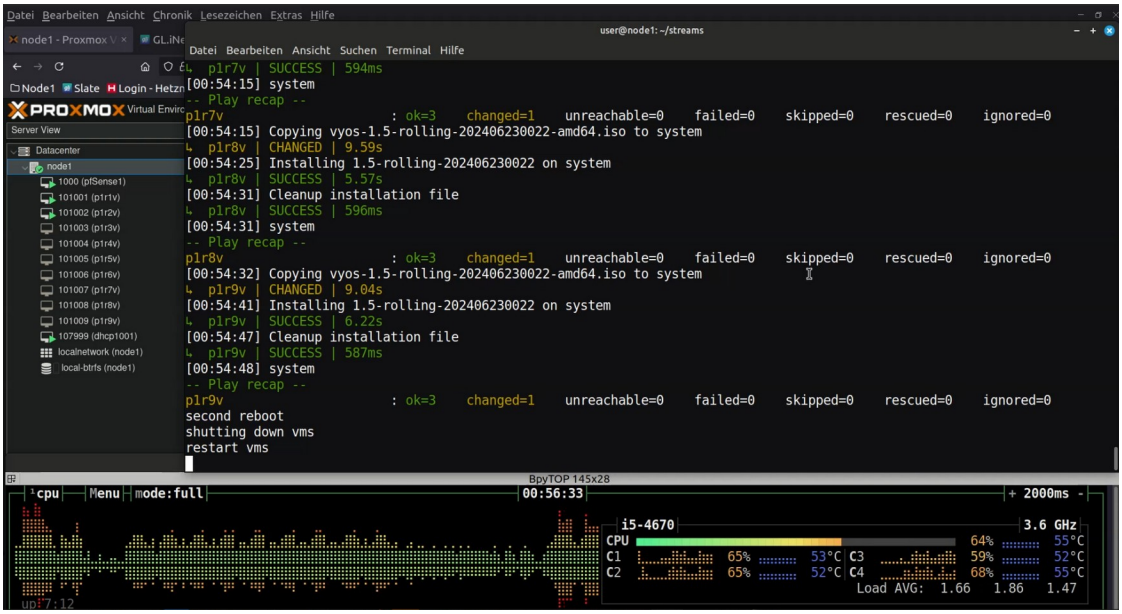
```
6  app.secret_key = os.getenv('SECRET_KEY', 'default-secret-key')
```

Folgende Zeile in die `~/.bashrc` oder `~/.bash_profile` eintragen:

```
export SECRET_KEY='y0ur$ecr3t_k3y_w1th_r@ndomCh@racters'
```

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 m.E. 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*** gibt es eine noch stärkere Erwärmung. Wer das vermeiden möchte kann den Turbo Boost im BIOS deaktivien, sodass eine Taktung von (hier) maximal 3,2 MHz eingehalten wird und dementsprechend auch die Temperatur wesentlich geringer ausfällt.

Fazit: Man könnte grundsätzlich auch sehr schwache Rechner wie z.B. den i5-4570 im Turbo Mode laufen lassen. Dann kommt es über lange Strecken zu 100% CPU-Auslastung und einer starken Erwärmung. Sollte die CPU zu heiß werden, reduziert sie automatisch ihre Taktrate (außer bei extrem alten CPU's aus dem letzten Jahrhundert) und arbeitet dementsprechend langsamer. Demnach kann es vorkommen, dass der Normalmodus - nämlich ohne Drosselung der Taktrate - schneller durchläuft, als der Turbo Mode, wenn er weitgehend mit einer stark reduzierten Taktung läuft.

To-Do-Liste für mögliche 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.