keepalived, lvs / haproxy Loadbalancer

Installation

Installation und einrichtung

Beschreibung:

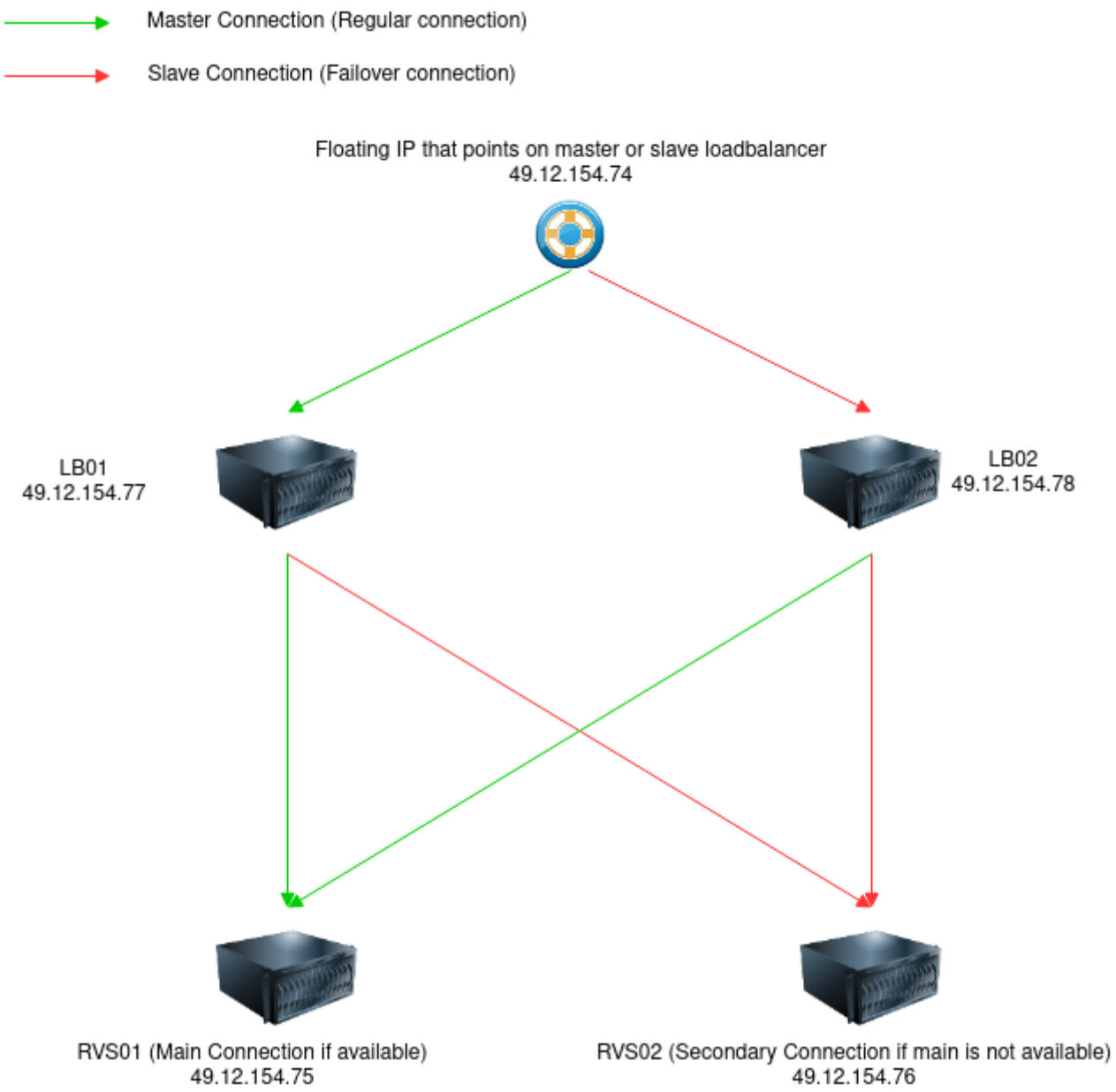
Hier ist ein Tutorial, um zwei Loadbalancer mit einer Floating-IP erreichbar zu machen. Damit nicht nur die Backendserver-Verbindung hochverfügbar ist, sondern auch die Verbindungs-IP (Floating-IP) mithilfe von zwei Loadbalancern anstelle von einem. Zusätzlich überprüfen die Loadbalancer die OpenVPN-Dienste und HTTPS-Dienste.
Es kann natürlich jeder anderer Dienst in betracht gezogen werden

Hinweis:

Bei einem N2N Tunnel funktioniert der LVS nicht richtig, da dann haproxy nehmen. Aber als reiner LB für die floating IPS reicht es.

Vorrausetzungen:

5 x Public IP Adresses, 2 x LB , 2 x RVS and 1 x Floating ip
2 x Machines as LB01 and LB02, with Linux (eg Debian) and ssh access
2 x Machines as RVS01 and RVS02 for example
Ein internes Netz zwischen den Loadbalancern für die Heartbeat funktion. (Hier wird ein broadcast gesendet, der über Öffentliche Netzte nicht funktioniert)
In unserer Testumgebung haben wir diese IPS
Floating IP : 49.12.154.74
LB01 : 49.12.154.77
LB02 : 49.12.154.78
RVS01 : 49.12.154.75
RVS02 : 49.12.154.76



Installation:

Installiere die Software-Anforderungen durch die folgenden Pakete und Konfigurationen auf LB01 und LB02. Es ist notwendig, dass die Netzwerk-IP und SSH korrekt konfiguriert sind für die nächsten Schritte...

```
apt update
apt dist-upgrade
apt install curl wget
apt install keepalived
apt install python3 python-is-python3
```

Das Routing muss in der sysctl auch aktiviert sein.

```
nano /etc/sysctl.conf
```

Inhalt am Ende anfügen

```
net.ipv4.ip_forward=1
net.ipv6.conf.all.forwarding=1
```

nun anwenden

```
sysctl -p
```

Anforderungen an die RVS-Server für HA mit Loadbalancer und Floating-IP.
Wir müssen nur eine iptables-Regel hinzufügen, weil die RVS-Server zur Floating-IP umgeleitet werden müssen.
Befehl zur direkten Ausführung. Siehe Punkt 6, um dies automatisch beim Start des Servers durchzuführen.

```
iptables -t nat -A PREROUTING -d 49.12.154.74 -j REDIRECT
```

Keepalived-Konfiguration, der lustige Teil.
Wir beginnen mit der Erklärung der Konfiguration von Server 1, die Konfiguration von Server 2 ist fast identisch mit der von Server 1, mit nur wenigen Änderungen.

Wir starten mit Server 1.

Hinweis: Hinweis: In version 3 von Keepalived wurde die Autehntifizierung.

Der Abschnitt .
...
 authentication {   #Validierungsinformationen einstellen, zwei Knoten müssen konsistent sein
        auth_type PASS   #Authentifizierungstypen einrichten, hauptsächlich PASS und AH
        auth_pass 8nzlTBSoSrpJP0i77TgL!   #Das Authentifizierungspasswort einstellen, zwei Knoten müssen unter einem vrrp_instance dasselbe Passwort verwenden

    }
....
kann entfernt werden.
Denn dieser wird ignoriert.
Das bedeutet das für die Heartbeat Verbindung kein Öffentliches Netz mehr gewählt werden sollte, sondern nur Private / dedizierte Verbindungen.

Hinweis: Keepalived Heartbeat ist Standard Multicast, sollte die Verbindung zwischen den LBs kein Multicast unterstützen

dann den Abschnitt weiter unten Unicast anschauen. Ich würde warscheinlich generell Unicast verwenden um Fehler vorzubeugen.

Erstellen Sie eine neue Datei /etc/keepalived/keepalived.conf.
Die Datei hat verschiedene Abschnitte:

global_defs{}: Dies enthält Benachrichtigungen und Identifikationen der Loadbalancer.
vrrp_instance VI_1{}: Enthält die Verbindungsauthentifizierung und die Zuweisung der virtuellen IP (Floating IP).
Hier können wir auch Skripte definieren, die bei Statusänderungen von Master zu Backup oder von Backup zu Master ausgeführt werden.
Zum Beispiel, um eine andere Floating-IP von Hetzner über ein sh-Skript zuzuweisen.

virtual_server{}: Hier wird die virtuelle Server-IP mit Port und Protokoll (UDP, TCP) definiert.
In diesem Abschnitt ist der letzte Abschnittstyp eingebettet.

Die real_servers, die in Punkt d aufgelistet sind.

real_server{}: Hier werden die RVS-Server mit Service-Checks definiert.

Die Global Config

```
#==========Global Config===============#

#Email notification
global_defs {
notification_email {
root@localhost  //Accept email address
}

#Mailserver Settings
notification_email_from keepalived@localhost  //Mailing address
smtp_server 127.0.0.1  //Send mail server IP
smtp_connect_timeout 30  //Mail connection timeout
#Rounting settings #here master on the seconday lb02 it must called be slave
router_id master #An identity that identifies the keepalived server running (type an string)
}

#======== End Globals ===========#
```

vrrp_instance section with virtual ip

Die Authentication wurfde in Version 3 entfernt. Da das hauptmerkmal darauf gelegt wurde, das die interne verbindung für das Heartbeat sicher ist. Sprich firewall regeln dediziertes VPn etc.

```
#======= HA for Floating IP ===#
vrrp_instance VI_1 {          #VRRP instance definition section
   state MASTER              #Gibt an, dass der Knoten der Hauptknoten ist (Großbuchstaben) und der Standby-Knoten BACKUP ist
   interface ens18           #Netzwerkschnittstelle, über die die Interne kommunikation laufen soll, heartbeat. Muss ein Privates netzwerk sein, da hier einj Broadcast gesendet wird.
   virtual_router_id 51      #VRRP Gruppenname, zwei Knoten müssen gleich eingestellt werden, um anzuzeigen, dass jeder Knoten derselben VRRP-Gruppe angehört. Muss eine ganze Zahl sein
   priority 100              #Priorität des Hauptknotens (1-254), Standard 100, beachten Sie, dass die Priorität des sekundären Knotens niedriger sein muss als die des Hauptknotens
   advert_int 1              #Legt das Zeitintervall zwischen Synchronisationsprüfungen zwischen zwei Knoten fest, die beiden Knoten müssen konsistent sein. Wert in Sekunden
   authentication {          #Validierungsinformationen einstellen, zwei Knoten müssen konsistent sein
      auth_type PASS         #Authentifizierungstypen einrichten, hauptsächlich PASS und AH
      auth_pass 8nzlTBSoSrpJP0i77TgL!        #Das Authentifizierungspasswort einstellen, zwei Knoten müssen unter einem vrrp_instance dasselbe Passwort verwenden

   }
   virtual_ipaddress {       #Virtuelle IP (Floating IPs) angeben, beide Knoten müssen gleich eingestellt sein, es kann mehr als eine geben, eine pro Zeile. Als Parameter dev das Netzwerkgerät, dem die IP zusätzlich zugewiesen wurde
      49.12.154.74 dev ens18
   }

#Benachrichtigungsbereich, wann welches Skript im angegebenen Zustand ausgeführt wird (z.B. Skript zur Zuweisung von Floating IP per Skript bei Hetzner oder anderen)
#Nur notify wird bei jedem Zustand aufgerufen.
#Aber die folgenden Parameter werden übergeben
#notify /pfad_zum_skript/skript_allezustände.sh
#Im Skript allezustände gibt es 3 Parameter, die überprüft werden können
#Type enthält den Wert "GROUP" oder "INSTANCE"
#TYPE = $1
#Name enthält den Namen der Gruppe oder Instanz
#NAME = $2
#State enthält den Zustand. Diese sind "MASTER" "BACKUP" "FAULT" mit * kann man einen unbekannten Zustand abfangen
#STATE = $3
```

```
#Beispiel skript_allezustände.sh weiter unten im nächsten Codekasten.
Da wir ein script für alle zustände haben, wenn wir das hetzner script verwenden möchten dann müssen wir alle scripte unten austauschen,

#ansonsten selbt andere scripte definieren

#Diese rufen die Skripte im gegebenen Zustand auf, daher ist keine Überprüfung im sh-Skript erforderlich
#notify_master /pfad_zum_skript/skript_hauptknoten.sh
#    (oder notify_master “ /pfad_zum_skript/skript_hauptknoten.sh)
#notify_backup /pfad_zum_skript/skript_backup.sh
#    (oder notify_backup “ /pfad_zum_skript/skript_backup.sh)
#notify_fault /pfad_zum_skript/skript_fehler.sh
#    (oder notify_fault “ /pfad_zum_skript/skript_fehler.sh)

}
#=== End Ha for Floating IP ========================#
```

Beispiel skript_allezustände.sh mit Hetzner Floating-IP-Zuweisungsbeispiel. In unserer Textumgebung verwenden wir keine Hetzner Floating-IP, aber als Beispiel für andere, die es nutzen werden. Dies ist ein Beispiel-Skript für lb01, daher wird die Hetzner Server-ID 5605626 verwendet. das script wertet über Parameter $3 also state aus, was Phase ist, master, backup ,unknown

```
#!/bin/bash

#lb01 ID : 5605626
#lb02 ID : 5605974
#floatingipv4 id =: 245713
#floatingipv6 id =: 246612

TYPE=$1
NAME=$2
STATE=$3

case $STATE in
        "MASTER") curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer <hetzner-api-key>" -d '{"server": 5605626}' 'https://api.hetzner.cloud/v1/floating_ips/245713/actions/assign'
                curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer <hetzner-api-key>" -d '{"server": 5605626}' 'https://api.hetzner.cloud/v1/floating_ips/246612/actions/assign'
                echo "Master"
                exit 0
```

```
        ;;
    "BACKUP") echo "backup"
        exit 0
        ;;
    "FAULT")  echo "fault"
        exit 0
        ;;
    *)       echo "unknown state"
        exit 1
        ;;
esac
```

Zurück in der Konfigurationsdatei nach dem Exkurs zum allstateshscript.
Der Virtualserver-Bereich ist in zwei Abschnitte unterteilt. Der Virtualserver selbst und die Realservers. In den Virtualserver-Einstellungen legen wir den Load-Balancer-Modus, die virtuelle IP und den Port fest, auf den der Server hört.
Im Realservers-Abschnitt befinden sich die Checks und die Serveradresse. So haben wir eine Liste von Realservers. Dies sind die Server, die im Failover ausbalanciert werden...
Die Checks können auch benutzerdefinierte Skripte sein. Für OpenVPN verwenden wir ein Python3-Skript von Icinga2, das die Verfügbarkeit von OpenVPN überprüft. Dieses Skript gibt 0 für Erfolg und 1 für Fehlschlag zurück.
Ideal für unseren OpenVPN-Service-Check. Das openvpn_check-Skript ist hier als Anhang:

check_vpn.py.

Im check_vpn-Skript müssen Sie den Rückgabecode bei "crit" von 2 auf 1 ändern.
Sie müssen das Skript ausführbar machen: chmod +x check_vpn.py.

```
#==========Virtual Server - Definition which floating ip and Ports
#========== Virtual Server for OpenVPN UDP 1194============#
virtual_server 49.12.154.74 1194 {  #Virtueller IP-Dienst
    delay_loop 6                  #Intervall für die Überprüfung des tatsächlichen Servers festlegen
    lvs_sched  fo                 #LVS-Zeitplanalgorithmus angeben (fo, wechselt bei Gewicht, das höchste gewinnt)
    #if Dr Mode dont forget to set iptables nat to the virtual ip on the realservers eg. on 49.12.154.75
    #iptables -t nat -A PREROUTING -d 49.12.154.74 -j REDIRECT
    #its virtual ip : 49.12.154.74
    #LVS direct routing
    lvs_method DR                  #Specify LVS mode, mainly NAT, TUN, DR
    protocol UDP                  #Forwarding protocol is UDP
#Real-Server - here Specify nodes 1-n. Can have an other port, when whished
    #================= Reaslserver UDP 1194 VPN02
    real_server 49.12.154.75 1194  { #Backend Real Server Configuration for VPN01
        weight 10                 #Set the weight value of the server node. The highest is preferred connection, if
```

```
available.
        #this is an Standard TCP check, but we will check UDP VPN we need an own check script, that return 0 or 1.
0 = OK, 1 = fail
    #TCP_CHECK {                #Real Server State Detection Settings section, in seconds
        #    connect_timeout 3       #Connection timeout
        #    nb_get_retry 3          #retry count
        #    delay_before_retry 3    #retry interval
        #    connect_port 80         #Connection Port
        #}
        #This done with, misc check
        MISC_CHECK {
              #misc_path /path/to/check_whatever.sh or py. Hint when parameters given it must be
quoted
              #misc_path "/root/check_vpn.py -p 1194 --tls-auth /root/ta.key 49.12.154.76" this is when tls key is
needed
              misc_path "/root/check_vpn.py -p 1194 49.12.154.76"
              }


    }
    #==================== Realserver UDP 1194 LB02 ==============#
    real_server 49.12.154.76 1194  { #Backend Real Server Configuration for VPN02
        weight 9                    #Set the weight value of the server node. The highest is preferred connection, if
available.
        #this is an Standard TCP check, but we will check UDP VPN we need an own check script, that return 0 or 1.
0 = OK, 1 = fail
        #TCP_CHECK {                #Real Server State Detection Settings section, in seconds
        #    connect_timeout 3       #Connection timeout
        #    nb_get_retry 3          #retry count
        #    delay_before_retry 3    #retry interval
        #    connect_port 80         #Connection Port
        #}
        #This done with, misc check
        MISC_CHECK {
              #misc_path /path/to/check_whatever.sh or py. Hint when parameters given it must be quoted
              #misc_path "/root/check_vpn.py -p 1194 --tls-auth /root/ta.key 49.12.154.76" this is when tls key is
needed
              misc_path "/root/check_vpn.py -p 1194 49.12.154.76"
              }


    }
```

```
    #============= End Realsserver =======================#

 }
#=== End Virtual Server UDP 1194==================#
```

Der nächste Virtual Server HTTPS 443, hier ist auch vpn auf 443 das sogenannte split tunneling, wird per webrowser angefragt gehts auf einen webserver, aber wir wollen hier den VPN dienst prüfen

```
#===Virtual Server TCP VPN 443
virtual_server 49.12.154.74 443 {  #Virtual IP Service
    delay_loop 6                  #Set interval to check actual server
    lvs_sched  fo                 #Specify LVS Scheduling Algorithm (fo, switched at weight, the heighest wins)
    #f Dr Mode dont forget to set iptables nat to the virtual ip on the realservers eg. on 49.12.154.75
    #iptables -t nat -A PREROUTING -d 192.168.35.127 -j REDIRECT
    #its virtual ip : 49.12.154.74
    lvs_method DR                     #Specify LVS mode, mainly NAT, TUN, DR
#    persistence_timeout 50         #Session Hold Time
    protocol TCP                  #Forwarding protocol is TCP
#Real-Server - here Specify nodes 1-n. Can have an other port, when whished
    #================== Reaslserver TCP 443 VPN02
    real_server 49.12.154.75 443  { #Backend Real Server Configuration for VPN01
        weight 10                  #Set the weight value of the server node. The highest is preferred connection, if
available.
        MISC_CHECK {
                #misc_path /path/to/check_whatever.sh or py. Hint when parameters given it must be quoted
                #misc_path "/root/check_vpn.py -p 443 -t --tls-auth /root/ta.key 49.12.154.75" this is when tls key is
needed
                misc_path "/root/check_vpn.py -p 443 -t 49.12.154.75"
                }
        }
    #==================== Realserver TCP 443 LB02 ==============#
    real_server 49.12.154.76 443  { #Backend Real Server Configuration for VPN02
        weight 9                   #Set the weight value of the server node. The highest is preferred connection, if
available.
        MISC_CHECK {
                #misc_path /path/to/check_whatever.sh or py. Hint when parameters given it must be quoted
                #misc_path "/root/check_vpn.py -p 443 -t --tls-auth /root/ta.key 49.12.154.76" this is when tls key is
needed
                misc_path "/root/check_vpn.py -p 443 -t 49.12.154.76"
```

```
        }
    }
   #============ End Real Server ================#
 }
#================ End Virtual Server TCP 443
```

Möchte man einen TCP check benutzen sähe das ganze so aus, hier enhmen wir https 10443 kann aber auch 443 sein für euer Scenario.

```
#===Virtual Server TCP 10443
virtual_server 49.12.154.74 10443 {  #Virtual IP Service
    delay_loop 6                  #Set interval to check actual server
    lvs_sched  fo                 #Specify LVS Scheduling Algorithm (fo, switched at weight, the heighest wins)
    #f Dr Mode dont forget to set iptables nat to the virtual ip on the realservers eg. on 49.12.154.75
    #iptables -t nat -A PREROUTING -d 192.168.35.127 -j REDIRECT
    #its virtual ip : 49.12.154.74
    lvs_method DR                     #Specify LVS mode, mainly NAT, TUN, DR

#    persistence_timeout 50         #Session Hold Time
    protocol TCP                  #Forwarding protocol is TCP
#Real-Server - here Specify nodes 1-n. Can have an other port, when whished
    #================== Reaslserver TCP 10443 VPN02
    real_server 49.12.154.75 10443  { #Backend Real Server Configuration for VPN01
        weight 10                  #Set the weight value of the server node. The highest is preferred connection, if
available.
        TCP_CHECK {                #Real Server State Detection Settings section, in seconds
            connect_timeout 3       #Connection timeout
            nb_get_retry 3         #retry count
            delay_before_retry 3    #retry interval
            connect_port 10443         #Connection Port
        }



    }
    #==================== Realserver TCP 10443 LB02 ==============#
    real_server 49.12.154.76 10443  { #Backend Real Server Configuration for VPN02
        weight 9                   #Set the weight value of the server node. The highest is preferred connection, if
available.

        TCP_CHECK {                #Real Server State Detection Settings section, in seconds
```

```
        connect_timeout 3       #Connection timeout
        nb_get_retry 3          #retry count
        delay_before_retry 3    #retry interval
        connect_port 10443      #Connection Port
      }


   }
  #============= End Real Server ==================#
 }
#================= End Virtual Server TCP 10443
#========== End Virtual Server - Definition which floating ip and Ports ==================#
#========== End config file ================#
```

Somit hätten wir die Scenarien einmal mit benutzerdefinierten check und TCP check abgedeckt.

Jetzt ist die Konfigurationsdatei für lb01 fertig.

Nun Server 2

Wir können diese Konfigurationsdatei kopieren und müssen nur wenige Dinge ändern.
Globaler Abschnitt.

Hier ändern wir die id auf backup

```
#==========Global Config==============#


....
#Rounting settings #here master on the seconday lb02 it must called be slave
router_id backup #An identity that identifies the keepalived server running (type an string)
}


#======= End Globals ==========#
```

Nun die HA vrrp_instance section

Hier muss der state auf BACKUP geändert werden.
Die Priority von 100 auf 99.
Sollte die Netzwerkkarte eine andere sein, diese natürlich auch anpassen

```
#======= HA for Floating IP ===#
vrrp_instance VI_1 {          #VRRP instance definition section
```

```
    state BACKUP              #Specify that the node is the primary node (uppercase) and the standby node is BACKUP
    interface ens18           #Netzwerkschnittstelle, über die die Interne kommunikation laufen soll, heartbeat. Muss ein Privates netzwerk sein, da hier einj Broadcast gesendet wird.

    virtual_router_id 51      #VRRP group name, two nodes need to be set the same to indicate that each node belongs to the same VRRP group. Must be an integer
    priority 99               #Priority of the primary node (1-254), default 100, note that the secondary node priority needs to be lower than the primary node
    advert_int 1              #Set the time interval between synchronization checks between two nodes, the two nodes need to be consistent. Value in seconds
    authentication {          #Set validation information, two nodes need to be consistent
        auth_type PASS        #Set up authentication types, mainly PASS and AH
        auth_pass 8nzlTBSoSrpJP0i77TgL!        #Set the authentication password, two nodes must use the same secret under a vrrp_instance

    }
    virtual_ipaddress {       #Specify virtual IP (floating IPs), two nodes need to be set the same, can have more than one, one per line. as paremeter dev the network device were the ip additionally assigned
        49.12.154.74 dev ens18  #it could be an other interface name on lb02
    }


   ....
#=== End Ha for Floating IP ============================#
```

Fertig mit der Konfiguration.

Kopieren Sie die Konfigurationsdateienund starten der Dienste

nach /etc/keepalived und die check_vpn.py nach /root.
Hier sind die kompletten Konfigurationsdateien:

lb01.conf

lb02.conf

check_vpn.py.

Dann auf den Loadbalancern den keepalived service starten

```
service keepalived restart
```

Nun noch auf den Servern die route zur floating ip legen

```
/usr/sbin/iptables -t nat -A PREROUTING -d 49.12.154.74 -j REDIRECT
```

Dieses kann man auch in die interfaces als post-up Befehl integrieren.

Beispiel:

```
auto enp6s18
iface enp6s18 inet static
    address 192.168.178.138
    netmask 255.255.255.0
    gateway 192.168.178.1
    dns-nameservers 8.8.8.8 8.8.4.4
    post-up /usr/sbin/iptables -t nat -A PREROUTING -d 49.12.154.74 -j REDIRECT
```

Testen Sie die Konfiguration.
Wenn der Service gestartet ist, können wir den Status damit anzeigen. Geben Sie auf beiden Loadbalancern ein:
Wir können auch sehen, welchen Status die Überprüfungen hatten.

Befehl:

```
service keepalived status
```

Ausgabe:
Hier sehen wir das der Dienst HTTPS 10443 nicht mehr auf der 76 verfügbar ist und schwenkt zu 75.
Hier ist ein Server ausgefallen, das ist kein Schwenkt des Loadbalancers

```
service keepalived status
Output from lb01
root@lb01:~# service keepalived status
● keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2022-03-16 11:29:36 CET; 21s ago
   Main PID: 2900 (keepalived)
      Tasks: 3 (limit: 2340)
     Memory: 2.0M
        CPU: 634ms
     CGroup: /system.slice/keepalived.service
             ├─2900 /usr/sbin/keepalived --dont-fork
```

```
        ├─2902 /usr/sbin/keepalived --dont-fork
        └─2903 /usr/sbin/keepalived --dont-fork

Mär 16 11:29:40 lb01 Keepalived_healthcheckers[2902]: TCP_CHECK on service [49.12.154.76]:tcp:10443 failed.
Mär 16 11:29:40 lb01 Keepalived_healthcheckers[2902]: Removing service [49.12.154.76]:tcp:10443 to VS
[49.12.154.74]:tcp:10443
Mär 16 11:29:40 lb01 Keepalived_vrrp[2903]: (VI_1) received lower priority (99) advert from 49.12.154.78 -
discarding
Mär 16 11:29:40 lb01 Keepalived_vrrp[2903]: (VI_1) Entering MASTER STATE
Mär 16 11:29:41 lb01 Keepalived_healthcheckers[2902]: TCP connection to [49.12.154.75]:tcp:10443 success.
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: TCP connection to [49.12.154.75]:tcp:443 success.
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: Misc check for [[49.12.154.76]:udp:1194 VS
[49.12.154.74]:udp:1194] by [/root/check_vpn.py] failed with retry disabled (exited with status 1).
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: Removing service [49.12.154.76]:udp:1194 to VS
[49.12.154.74]:udp:1194
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: TCP_CHECK on service [49.12.154.76]:tcp:443 failed.
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: Removing service [49.12.154.76]:tcp:443 to VS
[49.12.154.74]:tcp:443

In this lines wee see that the RVS tcp check 76 is not available on 10443 and is removed
Mär 16 11:29:40 lb01 Keepalived_healthcheckers[2902]: TCP_CHECK on service [49.12.154.76]:tcp:10443 failed.
Mär 16 11:29:40 lb01 Keepalived_healthcheckers[2902]: Removing service [49.12.154.76]:tcp:10443 to VS
[49.12.154.74]:tcp:10443

In this lines we see that RVS 76 misc check openvpn fails and is removed
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: Misc check for [[49.12.154.76]:udp:1194 VS
[49.12.154.74]:udp:1194] by [/root/check_vpn.py] failed with retry disabled (exited with status 1).
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: Removing service [49.12.154.76]:udp:1194 to VS
[49.12.154.74]:udp:1194

In this lines we see that RVS 76 tcp check is not available on 443 and is removed
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: TCP_CHECK on service [49.12.154.76]:tcp:443 failed.
Mär 16 11:29:42 lb01 Keepalived_healthcheckers[2902]: Removing service [49.12.154.76]:tcp:443 to VS
[49.12.154.74]:tcp:443
```

Anzeigen des connection status

```
ipvsadm -Ln --stats
```

Ausgabe:

```
root@lb01:~# ipvsadm -Ln --stats
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port               Conns   InPkts  OutPkts  InBytes OutBytes
  -> RemoteAddress:Port
TCP  49.12.154.74:443                   0       0       0       0       0
  -> 49.12.154.75:443                   0       0       0       0       0
TCP  49.12.154.74:10443                 0       0       0       0       0
  -> 49.12.154.75:10443                 0       0       0       0       0
UDP  49.12.154.74:1194                  0       0       0       0       0
  -> 49.12.154.75:1194                  0       0       0       0       0
root@lb01:~#

#Hier können wir sehen, dass die virtuelle IP 74 nur auf einen Server, nämlich 75, zeigt, weil die Überprüfungen auf 76 fehlgeschlagen sind.
#Das ist klar, denn wir haben das LAN-Kabel von 76 zu Testzwecken herausgezogen.
```

Jetzt stecken wir das Kabel wieder ein.
Im Service-Status sehen wir, dass der Service-Status auf 76 automatisch zurückkehrt.

```
root@lb01:~# service keepalived status
● keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2022-03-16 13:56:27 CET; 17min ago
   Main PID: 13902 (keepalived)
      Tasks: 3 (limit: 2340)
     Memory: 2.0M
        CPU: 29.825s
     CGroup: /system.slice/keepalived.service
             ├─13902 /usr/sbin/keepalived --dont-fork
             ├─13903 /usr/sbin/keepalived --dont-fork
             └─13904 /usr/sbin/keepalived --dont-fork

Mär 16 14:06:03 lb01 Keepalived_healthcheckers[13903]: Lost quorum 1-0=1 > 0 for VS [49.12.154.74]:tcp:443
Mär 16 14:06:30 lb01 Keepalived_healthcheckers[13903]: TCP connection to [49.12.154.75]:tcp:443 success.
Mär 16 14:06:30 lb01 Keepalived_healthcheckers[13903]: Adding service [49.12.154.75]:tcp:443 to VS
[49.12.154.74]:tcp:443
Mär 16 14:06:30 lb01 Keepalived_healthcheckers[13903]: Gained quorum 1+0=1 <= 10 for VS
[49.12.154.74]:tcp:443
Mär 16 14:07:44 lb01 Keepalived_healthcheckers[13903]: TCP connection to [49.12.154.76]:tcp:10443 success.
Mär 16 14:07:44 lb01 Keepalived_healthcheckers[13903]: Adding service [49.12.154.76]:tcp:10443 to VS
```

```
[49.12.154.74]:tcp:10443
Mär 16 14:07:44 lb01 Keepalived_healthcheckers[13903]: TCP connection to [49.12.154.76]:tcp:443 success.
Mär 16 14:07:44 lb01 Keepalived_healthcheckers[13903]: Adding service [49.12.154.76]:tcp:443 to VS
[49.12.154.74]:tcp:443
Mär 16 14:12:51 lb01 Keepalived_healthcheckers[13903]: Misc check for [[49.12.154.76]:udp:1194 VS
[49.12.154.74]:udp:1194] by [/root/check_vpn.py] succeeded.
Mär 16 14:12:51 lb01 Keepalived_healthcheckers[13903]: Adding service [49.12.154.76]:udp:1194 to VS
[49.12.154.74]:udp:1194
```

Das ganze nochmlas in

```
ipvsadm -Ln --stats
```

Ausgabe:

```
root@lb01:~# ipvsadm -Ln --stats
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port               Conns   InPkts  OutPkts  InBytes OutBytes
  -> RemoteAddress:Port
TCP  49.12.154.74:443                   0       0       0       0       0
  -> 49.12.154.75:443                   0       0       0       0       0
  -> 49.12.154.76:443                   0       0       0       0       0
TCP  49.12.154.74:10443                20     148       0    8880       0
  -> 49.12.154.75:10443                20     148       0    8880       0
  -> 49.12.154.76:10443                 0       0       0       0       0
UDP  49.12.154.74:1194                  0       0       0       0       0
  -> 49.12.154.75:1194                  0       0       0       0       0
  -> 49.12.154.76:1194                  0       0       0       0       0
```

Was passiert, wenn lb01 ausfällt? Zu Testzwecken haben wir den Keepalive-Service auf lb01 gestoppt.

```
root@lb01:~# service keepalived stop
root@lb01:~# service keepalived status
● keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Wed 2022-03-16 15:38:51 CET; 32min ago
    Process: 16485 ExecStart=/usr/sbin/keepalived --dont-fork $DAEMON_ARGS (code=exited,
status=0/SUCCESS)
   Main PID: 16485 (code=exited, status=0/SUCCESS)
```

```
        CPU: 4min 4.232s

Mär 16 15:15:23 lb01 Keepalived_healthcheckers[16486]: TCP connection to [49.12.154.76]:tcp:10443 success.
Mär 16 15:15:23 lb01 Keepalived_healthcheckers[16486]: Adding service [49.12.154.76]:tcp:10443 to VS
[49.12.154.74]:tcp:10443
Mär 16 15:38:50 lb01 Keepalived[16485]: Stopping
Mär 16 15:38:50 lb01 Keepalived_vrrp[16487]: (VI_1) sent 0 priority
Mär 16 15:38:50 lb01 systemd[1]: Stopping Keepalive Daemon (LVS and VRRP)...
Mär 16 15:38:51 lb01 Keepalived_vrrp[16487]: Stopped
Mär 16 15:38:51 lb01 Keepalived[16485]: Stopped Keepalived v2.1.5 (07/13,2020)
Mär 16 15:38:51 lb01 systemd[1]: keepalived.service: Succeeded.
Mär 16 15:38:51 lb01 systemd[1]: Stopped Keepalive Daemon (LVS and VRRP).
Mär 16 15:38:51 lb01 systemd[1]: keepalived.service: Consumed 4min 4.232s CPU time.
root@lb01:~#

On lb02
root@lb02:~# service keepalived status
● keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2022-03-16 14:32:29 CET; 1h 40min ago
   Main PID: 4733 (keepalived)
      Tasks: 3 (limit: 2340)
     Memory: 7.4M
        CPU: 6min 4.240s
     CGroup: /system.slice/keepalived.service
             ├─4733 /usr/sbin/keepalived --dont-fork
             ├─4734 /usr/sbin/keepalived --dont-fork
             └─4735 /usr/sbin/keepalived --dont-fork

Mär 16 15:15:21 lb02 Keepalived_healthcheckers[4734]: Misc check for [[49.12.154.76]:udp:1194 VS
[49.12.154.74]:udp:1194] by [/root/check_vpn.py] succeeded.
Mär 16 15:15:21 lb02 Keepalived_healthcheckers[4734]: Adding service [49.12.154.76]:udp:1194 to VS
[49.12.154.74]:udp:1194
Mär 16 15:15:23 lb02 Keepalived_healthcheckers[4734]: TCP connection to [49.12.154.76]:tcp:10443 success.
Mär 16 15:15:23 lb02 Keepalived_healthcheckers[4734]: Adding service [49.12.154.76]:tcp:10443 to VS
[49.12.154.74]:tcp:10443
Mär 16 15:15:24 lb02 Keepalived_healthcheckers[4734]: Misc check for [[49.12.154.76]:tcp:443 VS
[49.12.154.74]:tcp:443] by [/root/check_vpn.py] succeeded.
Mär 16 15:15:24 lb02 Keepalived_healthcheckers[4734]: Adding service [49.12.154.76]:tcp:443 to VS
```

[49.12.154.74]:tcp:443
Mär 16 15:38:50 lb02 Keepalived_vrrp[4735]: (VI_1) Backup received priority 0 advertisement
Mär 16 15:38:51 lb02 Keepalived_vrrp[4735]: (VI_1) Entering MASTER STATE

In the last Line we can see he will become master

Now we start on the lb01 again an the status of lb02

root@lb02:~# service keepalived status
● keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2022-03-16 14:32:29 CET; 1h 40min ago
   Main PID: 4733 (keepalived)
      Tasks: 3 (limit: 2340)
     Memory: 7.4M
        CPU: 6min 4.240s
     CGroup: /system.slice/keepalived.service
             ├─4733 /usr/sbin/keepalived --dont-fork
             ├─4734 /usr/sbin/keepalived --dont-fork
             └─4735 /usr/sbin/keepalived --dont-fork

Mär 16 15:15:21 lb02 Keepalived_healthcheckers[4734]: Misc check for [[49.12.154.76]:udp:1194 VS [49.12.154.74]:udp:1194] by [/root/check_vpn.py] succeeded.
Mär 16 15:15:21 lb02 Keepalived_healthcheckers[4734]: Adding service [49.12.154.76]:udp:1194 to VS [49.12.154.74]:udp:1194
Mär 16 15:15:23 lb02 Keepalived_healthcheckers[4734]: TCP connection to [49.12.154.76]:tcp:10443 success.
Mär 16 15:15:23 lb02 Keepalived_healthcheckers[4734]: Adding service [49.12.154.76]:tcp:10443 to VS [49.12.154.74]:tcp:10443
Mär 16 15:15:24 lb02 Keepalived_healthcheckers[4734]: Misc check for [[49.12.154.76]:tcp:443 VS [49.12.154.74]:tcp:443] by [/root/check_vpn.py] succeeded.
Mär 16 15:15:24 lb02 Keepalived_healthcheckers[4734]: Adding service [49.12.154.76]:tcp:443 to VS [49.12.154.74]:tcp:443
Mär 16 15:38:50 lb02 Keepalived_vrrp[4735]: (VI_1) Backup received priority 0 advertisement
Mär 16 15:38:51 lb02 Keepalived_vrrp[4735]: (VI_1) Entering MASTER STATE
Mär 16 16:12:21 lb02 Keepalived_vrrp[4735]: (VI_1) Master received advert from 49.12.154.77 with higher priority 100, ours 99
Mär 16 16:12:21 lb02 Keepalived_vrrp[4735]: (VI_1) Entering BACKUP STATE
root@lb02:~#

In the last Line we can see he will become backup state again

on the lb01 service status

```
root@lb01:~# service keepalived status
● keepalived.service - Keepalive Daemon (LVS and VRRP)
     Loaded: loaded (/lib/systemd/system/keepalived.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2022-03-16 16:12:18 CET; 10s ago
   Main PID: 23921 (keepalived)
      Tasks: 3 (limit: 2340)
     Memory: 2.0M
        CPU: 586ms
     CGroup: /system.slice/keepalived.service
             ├─23921 /usr/sbin/keepalived --dont-fork
             ├─23922 /usr/sbin/keepalived --dont-fork
             └─23923 /usr/sbin/keepalived --dont-fork

Mär 16 16:12:19 lb01 Keepalived_healthcheckers[23922]: TCP connection to [49.12.154.75]:tcp:10443 success.
Mär 16 16:12:19 lb01 Keepalived_vrrp[23923]: (VI_1) received lower priority (99) advert from 49.12.154.78 - discarding
Mär 16 16:12:20 lb01 Keepalived_vrrp[23923]: (VI_1) received lower priority (99) advert from 49.12.154.78 - discarding
Mär 16 16:12:21 lb01 Keepalived_healthcheckers[23922]: Misc check for [[49.12.154.76]:udp:1194 VS [49.12.154.74]:udp:1194] by [/root/check_vpn.py] succeeded.
Mär 16 16:12:21 lb01 Keepalived_vrrp[23923]: (VI_1) received lower priority (99) advert from 49.12.154.78 - discarding
Mär 16 16:12:21 lb01 Keepalived_vrrp[23923]: (VI_1) Entering MASTER STATE
Mär 16 16:12:22 lb01 Keepalived_healthcheckers[23922]: Misc check for [[49.12.154.75]:udp:1194 VS [49.12.154.74]:udp:1194] by [/root/check_vpn.py] succeeded.
Mär 16 16:12:22 lb01 Keepalived_healthcheckers[23922]: Misc check for [[49.12.154.76]:tcp:443 VS [49.12.154.74]:tcp:443] by [/root/check_vpn.py] succeeded.
Mär 16 16:12:23 lb01 Keepalived_healthcheckers[23922]: Misc check for [[49.12.154.75]:tcp:443 VS [49.12.154.74]:tcp:443] by [/root/check_vpn.py] succeeded.
Mär 16 16:12:25 lb01 Keepalived_healthcheckers[23922]: TCP connection to [49.12.154.76]:tcp:10443 success.
root@lb01:~#
```

In this Line

....

Mär 16 16:12:21 lb01 Keepalived_vrrp[23923]: (VI_1) Entering MASTER STATE

....

He will become master again.

Mission accomplished, HA keepalived works!

Unicast wenn das Heartbeat Netz nicht in einem Netz mit Multicast funktionen ist.

Multicast funktioniert nicht über VPN und nicht im Hetzner internen Netz.
Für Unicast sollte allerdings kein Öffentliches Netz gewählt werden, denn die Autehntifizierung wurde in Version 3 entfernt.
Daher nur Private / Dedizierte Verbindungen für Heartbeate verwenden.

Um Unicast zu benutzen in der VVRP einfach folgendes hinzufügen.
Die eigen IP von Privaten / VPN Netz und die IP Adresse vom Privaten / VPN Netz des anderen.
Es können auch mehrer einträge gültig seoin.

```
Einmal mit zwei LBs


10.1.0.3 ist die IP von Knoten selbst, hier lb02
10.1.0.2 ist die ip von lb01


...
 priority 99
    advert_int 1

    unicast_src_ip 10.1.0.3
    unicast_peer {
                    10.1.0.2
                            }



    virtual_ipaddress {
...

Einmal mit zwei LBs


10.1.0.3 ist die IP von Knoten selbst, hier lb02
10.1.0.2 ist die ip von lb01
10.1.0.4 ist die ip von lb03
```

Die ips bei Peer müssen untereinander geschrieben sein.

```
...
 priority 99
    advert_int 1

    unicast_src_ip 10.1.0.3
    unicast_peer {
                    10.1.0.2
                    10.1.0.4
                            }



    virtual_ipaddress {
...
```

lvs_method und lb_kind

n der Konfiguration von Linux Virtual Server (LVS) innerhalb von `keepalived` oder ähnlichen Load Balancing-Lösungen sind `lvs_method` und `lb_kind` zwei verschiedene Konfigurationseinstellungen, die verschiedene Aspekte des Load Balancing-Verhaltens steuern.

lvs_method (Load Balancing Method)

- **Bedeutung**: `lvs_method` bezieht sich auf die Methode, wie der Traffic an die Backend-Server (Real Server) weitergeleitet wird.
- **Typische Methoden**:
  - **NAT (Network Address Translation)**: Der eingehende Traffic wird umgeschrieben, sodass er von der IP des Load Balancers auf die IP des Real Servers umgeleitet wird. Die Antwort des Servers wird ebenfalls umgeschrieben, um vom Load Balancer zu stammen.
  - **DR (Direct Routing)**: Der eingehende Traffic wird direkt an die Backend-Server weitergeleitet, ohne die Pakete umzuschreiben. Die Server müssen so konfiguriert sein, dass sie den Traffic, der für die virtuelle IP-Adresse bestimmt ist, akzeptieren und verarbeiten.
  - **TUN (Tunneling)**: Der Traffic wird in IP-Tunnel eingekapselt und an die Backend-Server gesendet.

lb_kind (Load Balancer Kind)

- **Bedeutung**: `lb_kind` definiert, wie der Load Balancer selbst im Netzwerk funktioniert.
- **Typische Arten**:
  - **NAT**: Der Load Balancer führt eine Netzwerkadressübersetzung durch (wie oben beschrieben).
  - **IP-IP**: Der Load Balancer verwendet IP-IP-Tunneling, wobei die ursprünglichen IP-Pakete in neue IP-Pakete eingekapselt werden.
  - **BRIDGE**: Der Load Balancer arbeitet auf der Datenträgerebene (Layer 2) des OSI-Modells, was bedeutet, dass er wie eine Netzwerkbrücke funktioniert.
- Bei LB kind müss allerdings noch in der sysctl folgender Wert auf 1 gestell werden

```
net.ipv4.ip_nonlocal_bind = 1
```

Zusammenfassung

- **lvs_method** steuert, wie der Traffic an die Backend-Server weitergeleitet wird (z.B. NAT, DR, TUN).
- **lb_kind** definiert, wie der Load Balancer im Netzwerk arbeitet (z.B. NAT, IP-IP, BRIDGE).

Die Auswahl der richtigen Methode und des Typs hängt von Ihrer Netzwerktopologie, Ihren Performance-Anforderungen und anderen spezifischen Bedürfnissen Ihres Netzwerks ab. Jede Methode und jeder Typ hat seine eigenen Vor- und Nachteile in Bezug auf Komplexität, Durchsatz und Flexibilität.

Projekt installation mit Ansiblescript und Tools für Floating ips

Installations Keepalived mit HAProxy Loadbalancer mit Floating IPs mittels Ansible script

Beschreibung:

Auf der vorherigen Seite haben wir gesehen wie wir den manuell installieren und wie das ganze funktioniert.
Und wie Services eingebunden werden.

Hier ein Ansible Beispiel wie ein Loadbalancer mit floating ips bei Hetzner aufgesetzt werden aknn.
Ziel ist ein Loadbalancer zu haben der selbst eine Floating ips zwischen beiden LBs hat und keine andere Aufgabe hat als wenn ein LB offline geht die floating ips auf den oder die weiteren LB auszurollen.
Wofür könnte man sowas brauchen. Zum Beispiel umd diese floating ips zu Tiunneln odewr immer einen VPN Server der auf den LBs laufen könnte erreichbar zu halten.
Für was auch immer. Der Loadbalancer in diesem Setup ist Dualstack fähig also ipv4 und ipv6

Hier geht es ja nur darum, einen LB zu haben der Floating ips von links nach recht schiebt und umgegehrt.
Aber wenn man noch Services anglegen will, das die dann umgeschwenkt werden, kann man in der services.yml auch noch festlegen.

Ein internes Netz zwischen den Loadbalancern für die Heartbeat funktion. (Hier wird ein broadcast gesendet, der über Öffentliche Netzte nicht funktioniert)

Verwendenung des ansible Scriptes

Aufbau des Verzeichnisses:

```
.
├── getids.sh
├── hetzner.yml
├── hosts.ini
├── installkeepalived.yml
├── installpython3.yml
├── installwireuardmesh.yml
├── install_yq_jq.sh
├── output.txt
├── README.md
├── server_table.txt
├── services.yml
├── table.txt
├── templates
│   ├── haproxy.cfg.j2
│   ├── keepalived.j2
│   ├── keepalived_state_change.j2
│   └── wireguard.conf.j2
├── testapache2erstellen
│ ├── hosts.ini
│   └── installapache2withindex.yml
└── update_hetzner.sh
```

getids.sh

ist ein Script mit der die hetzner.yml erweitert wird.
Einfach ausführen auf dem Service System.
Mit den jeweiligen Parametern wird die hetzner.yml erweitert.
Bei ipv4 wird immer die einzelne Adresse vergeben, bei ipv6 wird immer ein komplettes Subnetz vergeben.

Die Parameter:

```
Usage: ./getids.sh [OPTIONS]
Options:
  -h, --help               Show this help message
  -d, --debug              Show debug information (JSON output)
  -lfl LABEL, --loadbalancer-floating-label LABEL
                           Specify label for loadbalancer floating IPs
```

```
  -fl LABEL, --floating-label LABEL
                        Specify label for other floating IPs
```

mit -h wird die hilfe aufgerufen

mit -lfl oder --loadbalancer-floating-label wird festgelegt das diese ips für den Loadbalancer selbst verwendet werden sollen.
Sprich diese werden dann auch einer Netzwerkkarte hinzugefügt Welche das ist wird in der Hostvariable

```
interface_floating=eth0
```

festgelegt
Diese werden dann als erstes in die hetzner xml eingetragen. sollten zwei ipvs mit dem selben Label vorhanden sein. Entscheidet der Zufall darüber, welche genommen wird. Deshalb immer auf eindeutige Label bzw Label Kombinationen achten.
Label können via Komma separiert werden um mehre Labels anzugeben. Es müssen alle Labels erfüllt sein, damit sie genommen werden (und verknüpft)

Beispiel:

```
-lfl lb_ips,loadbalancer_oldenburg
```

 -fl LABEL, --floating-label LABEL, Definiert die Floating IPS die auf dem an den Loadblancer gepackt werden sollen, aber keiner Netzwerkkarte zugeordnet, da diese eventuell weitergetunnelt werden könnten, oder anderweitig verwendet werden sollen.
SIe sollen nur auf dem Loadbalancer zur Verfügung stehen.

Es wird vom sSript auch geprüft ob die IP schon als IP für den Loadbalancer vergeben wurde, um doppel Vergebung zu vermeiden.

Beispiel:

```
-fl loadbalancer_oldenburg
```

hetzner.yml

Diese Datei enthält einmal den API-KEY und die URL für für Hetzner.
Sollte diese Datei nicht existieren. Einfach anlegen und diese Sektionen einfügen.
Damit ist auch das Grundgerüst gleich erklärt
Einmal ein Global Bereich für die Variablen und der floating IP-Bereich in dem die Floating IPs gespeichert werden.

```
global:
  hetzner_api_key: zHE7y9vFJQfWEpSuSUhRynLRScO5yM2OLA85NRAswnf23bFDZkugj7LORwVQq5fp
  hetzner_api_url: https://api.hetzner.cloud/v1
floating_ips:
```

Wenn das getids.sh script ausgeführt wurde, dann sieht die hetzner.yml als Beispiel so aus:

```
floating_ips:
- name: "lbs_ipv6"
  hetzner_id: xxxxx
  ipv6: "2xxxxxxxxxx::/64"

- name: "lbs_ipv4"
  hetzner_id: 61xxxx
  ipv4: "78.46.xxx.xxx"

- name: "ipv4_free04"
  hetzner_id: 16xxxx
  ipv4: "116.202.1xxx.xxx"

- name: "ipv4_free03"
  hetzner_id: 2xxxx
  ipv4: "116.202.xxx.xxx"

- name: "ipv6_free04"
  hetzner_id: 4xxxxx
  ipv6: "xxxxxxxxxxx::/64"
```

Das Ansible script wertet die ersten beiden floating IPs als Eigene IPs für sich selbst.
Das getids.sh script baue die hetzner.yml so das der erste wert eine ipv4 und der zweiter eine ipv6 aus der liste ist die dem Label entsprechen.

In der Hetzner.yml wird der Name der ip bei hetzner gespeichert
die id und der typ steht vor der ip entweder ipv4 oder ipv6.

alle anderen floating ips werden dem Loadbalancer nicht zugewiesen (also an ein interface gebunden) sondern werden nur auf der Machine verfügbar gemacht, so das Sie an ein Interface gebunden werden könnten.
Nun können diese Floating IPs flexibel verwendet werden. Entweder werden sie an Interfaces gebunden oder weiter getunnelt.
Je nach dem was man möchte.

host.ini

Sie enthält die Loadbalancer auf die das Script angewendet werden soll

In Ihr ist eine Gruppe für die dann gloabale variablen gelten.
Denn die müssen bei allen Loadblancern gleich sein und bei jedem host muss noch die hetzner_serverid angegben werden. Damit dann beim failover die ips an den richtigen server gepackt werden.
desweiteren muss noch das interface für die flaoting ip angeben werden auf den der Loablancer listen soll.
und ein weiteres interface mit einem privaten netzwerk über das das heratbeat laufen soll.
In dieser Beispiel host.ini sind es 3 Loadbalancer instancen. Minimum müssen zwei angeben sein.

Es gibt unter global variables der Pfad zur keepalived_state_change.sh
Des weiteren gibt es eine virtual_router id, die muss auch bei allen gleich sein, denn die beschreibt die Gruppe, diese ID wird per Broadcast gesendet.
Der Paramater **balancer_passphrase=** ist veraltet und wird nicht mehr unterstützt, da konnte noch ein Password vergeben werden. Deshalb sollten diese Verbindungen entweder durch einen Tunnel (Dafür der Wiregaurd, wenn man möchte) oder eine dedizierte private Verbindung in einem VLAN oder wie bei Hetzner ein lokales Netz.
Der Name ist frei vergebbar für die Datei und auch der Name, denn die Datei wird vom Ansible script erstellt.

```
[loadbalancer]
host1 hetzner_id=1020301 interface_floating=eth0 interface_heartbeat=eth1 ansible_ssh_user=root
host2 hetzner_id=1020302 interface_floating=eth0 interface_heartbeat=eth1 ansible_ssh_user=root
host3 hetzner_id=1020303 interface_floating=eth0 interface_heartbeat=eth1 ansible_ssh_user=root


[loadbalancer:vars]
virtual_router_id=50
script_path=/root/keepalived_state_change.sh
```

installkeepalived.yml

Dieses script rollt die Loadbalancer aus.
es benötigt keine weiteren paramter als die host.ini

```
ansible-playbook -i host.ini installkeepalived.yml
```

Und schon rennt er los.

installpython3.yml

Dieses script installiert die python3 library uf den hosts
Es ist ein ansible script.

```
ansible-playbook -i host.ini installpython3.yml
```

Muss nur einmal gemacht werden

installwireuardmesh.yml

Wenn der/die Loadbalancer als Wiregaurd Meshserver dienen soll, dieses Playbook ausführen.
wenn in den Hostvariablen diese Variable angeben ist und true ist, wird dieses palybook darauf angewandt
Es wir ein reines internes VPN-Netz gebaut. Und somit nur routen über das lokale subnetz gebaut.
Siehe variable wireguard_subnet.
Diese kann dazu verwendet werden um das keepalived heartbeat über einen Tunnel laufen zu lassen, wenn kein lokales Netzwerk zwischen den Loadbalancern möglich ist. Dann wird in der hostvariable für das Loadbalancer Netz der name des Wirguardadapters angegeben. In unserem Fall wirelb siehe weiter unten gloabale Wireguard Variablen.

```
enable_wireguard=true
```

Desweiteren werden folgende globale Variablen angewandt.
Die host.ini damit dann erweitern in der section [loadbalancer:vars]

```
wireguard_subnet=10.15.2.0/24
wireguard_interface_name=wirelb
wireguard_port=51820
wireguard_conf_dir=/etc/wireguard
```

install_yq_jq.sh

Dieses script installiert die Abhängihketen von yq und jq um json files zu parsen.
Diese json files kommen von der hetzner API.
Einfach ausführen mit root rechten auf dem Service System.
Dies sind abhängigkeiten die für getids.sh benötigt werden.

Muss nur einmal gemacht werden

output.txt

Enthält wenn die getids.sh mit dem parameter debug aufgerufen wird die komplette hetzner.json stream.
Dient zum debuggen.

server_table.txt

Diese enthält die Liste der Server mit den ids.
Diese werden erstellt wenn getids.sh ohne Paramter aufgerufen wird.
Gleichzeitg wird diese Tabelle auch in der Console nochmals ausgegben

services.yml

Über diese Datei werden, die Dienste (server festgelet die Überwacht werden sollen. Webserver smtp server etc.
Diese wird benötigt wenn er nicht nur als Loadbalancer für die IPS sondern auch für services dienen soll.

Aufbau:

```
services:
  - name: "nextcloud"
    floating_ipv4: "198.51.xxx.xxx"
    floating_ipv6: "xxxxxxxx"
    frontend_port: 443
    frontend_protocol: "TCP"
    backend_servers:
      - name: "nextcloud01"
        ipv4: "192.0.xxx.xxx"
        ipv6: "2001:xxxxxx"
        backend_port: 443
        backend_protocol: "TCP"
      - name: "nextcloud02"
        ipv4: "192.0.xxx.xxx"
        ipv6: "2001:xxxx.xxxx"
        backend_port: 443
        backend_protocol: "TCP"
```

Es können beliebig viele Services mit backend server angelegt werden.
Eigentlich selbst erkärend.

-name des dienstes
darunter die beiden floating ips
darunter der port und protokoll auf LB Seite ankommend

dann die backend server als liste
wieder name ipv4 und ipv6 des server port und Protokoll
und dannd er nächste server.
Dann könnet mann den den nächsten Dienst uim gleichen format anlegen.

Möchte man gar keine weiteren Dienste lässt man in der yml Datei nur services: stehen

```
services:
```

table.txt

Diese enthält die Liste der floating ips und den Servern mit den ids.
Diese werden erstellt wenn getids.sh ohne Paramter aufgerufen wird.
Gleichzeitg wird diese Tabelle auch in der Console nochmals ausgegben

templates

Im Template Verzeichnis sind die Template Dateien

```
├── templates
│   ├── haproxy.cfg.j2
│   ├── keepalived.j2
│   ├── keepalived_state_change.j2
│   └── wireguard.conf.j2
```

haproxy.cfg.j2 ist das template das die haproxy config aus der service.xml baut
Die keepalived.j2 dient zur erstellung der keepalived config
Die keepalived_state_change.j2 dient zur erstellungen des scriptes, in diesem Falle für Hetzner zum ip wchsel beim Loadbalancer wechsel bzw. failover. DIese script Dateien werden für das ansible Playbook benötigt.
wireguard.conf.j2 erstellt die wireguard konfiguration. Diese wird vom wireguardplaybook verwendet

Test Apache2

Der Testapache dient dazu schnell Apache server auszurollen die eine http webseite bereitstellen mit Hostname und ip Adresse in der index.html. Dient zum testen des keepalived mit dem haproxy. Dann kann in der service.xml diese hosts angeben werden.

```
└── testapache2erstellen
    ├── hosts.ini
    └── installapache2withindex.yml
```

hosts.ini für die server zum installieren
installapache2withindex.yml erstellt die index.html

update_hetzner.sh

Diese datei ist nur ein hilfe um nicht immer wieder den gleichen Befehl eintippen zu müssen wenn die Tags feststehen

Inhalt

```
#!/bin/bash
./getids.sh -lfl lb_ips,loadbalancer_oldenburg -fl loadbalancer_oldenburg
```

Ausführbar machen

```
chmod +x update_hetzner.sh
```

Abschluss

Damit wäre die installation unseres Ansible Script Ordner eingerichtet, alle abhängigkeiten uaf unserem Service PC installiert und die Dateien erklärt.
Wichtig ist das auf den LB Servern vorher noch das installpython3.yml ansible playbook ausgeführt wird um auch Sicher zu gehen das python3 zur verfügung steht auf den Servern.