Orchestrierung einer BGP-Konfiguration mit NETCONF und netmiko

Die Übung hat die Erstellung eines Skripts zum Ziel, das auf den Devices der Lab-Umgebung eine BGP-Konfiguration orchestriert, mit der die Devices sich über iBGP mit ihren Loopback0-Adressen benachbarn. Da die Loopback0-Adressen per OSPF geroutet werden, sollten die Nachbarschaften auch funktionieren.
Dazu liest das Skript per NETCONF die Loopback0-Adressen aller Devices ein, erzeugt die erforderliche BGP-Konfiguration und pusht diese mit netmiko an die Devices. Um die Entwicklung des Codes zu erleichtern, stehen zwei Module getloopback.py und genaratebgp.py zur Verfügung, deren Code in dem zu erstellenden Skript via Import weiter verwertet wird.

1.) Auf den Netzelementen muss NETCONF aktiviert werden:

    NX-OS: feature netconf
    IOS-XE: netconf-yang
            aaa new-model
            aaa authentication login default local
            aaa authorization exec default local

    Verbinden Sie sich dazu per Telnet/SSH mit dem Device und geben Sie die Kommandos ein.

2.) Wechseln Sie auf der VM in den Ordner ~/CPRN:

    student@workplace:~$ cd CPRN
    student@workplace:~/CPRN$

    Hier sind zwei Python-Skripte hinterlegt, die in der Übung weiterverwertet werden sollen:
    - getloopback.py: Liest per NETCONF die Loopback0-Adresse eines Netzelements aus und liefert diese als Rückgabe (String).
    - generatebgp.py: Erzeugt eine BGP-Konfiguration und liefert diese als Rückgabe (Liste mit Strings)

    Öffnen Sie beide Skripte im Editor und analysieren Sie den Code. In beiden Skripten gibt es Funktionen nxos() und iosxe() für die beiden Device-Typen im Lab. Welche Argumente erwarten die Funktionen bei Aufruf?

3.) Testen sie getloopback.py in der Interactive Shell:

    student@workplace:~/CPRN$ python3
    Python 3.8.10 (default, Mar 15 2022, 12:22:08)
    [GCC 9.4.0] on linux
    Type "help", "copyright", "credits" or "license" for more information.
    >>> import getloopback
    >>> getloopback.nxos('10.0.0.100', 'student', 'L1nux_dc')
    10.255.255.100
    '10.255.255.100'

```
>>> getloopback.iosxe('10.0.0.101', 'student', 'L1nux_dc')
10.255.255.101
'10.255.255.101'
>>> getloopback.iosxe('10.0.0.102', 'student', 'L1nux_dc')
10.255.255.102
'10.255.255.102'
```

4.) Testen Sie generatebgp.py in der Interactive Shell:

```
>>> import generatebgp
>>> generatebgp.nxos(65000 , '10.255.255.100' , ['10.255.255.100' , '10.255.255.101',
'10.255.255.102'])
['router bgp 65000', 'router-id 10.255.255.100', 'neighbor 10.255.255.101 remote-as 65000',
'update-source loopback 0', 'address-family ipv4 unicast', 'next-hop-self', 'neighbor
10.255.255.102 remote-as 65000', 'update-source loopback 0', 'address-family ipv4 unicast',
'next-hop-self']
>>> generatebgp.iosxe(65000 , '10.255.255.101' , ['10.255.255.100' , '10.255.255.101',
'10.255.255.102'])
['router bgp 65000', 'bgp router-id 10.255.255.101', 'neighbor 10.255.255.100 remote-as
65000', 'neighbor 10.255.255.100 update-source loopback 0', 'neighbor 10.255.255.100 next-
hop-self', 'neighbor 10.255.255.102 remote-as 65000', 'neighbor 10.255.255.102 update-
source loopback 0', 'neighbor 10.255.255.102 next-hop-self']
>>> exit()
student@workplace:~/CPRN$
```

5.) Ihre Aufgabe besteht darin, ein Python-Skript pushbgp.py zu erstellen, dass für alle Devices der Lab-Umgebung eine BGP-Konfiguration erzeugt und per netmiko an die Devices sendet. (Sie finden die finale Version des Codes in pushbgp_final.py)
Das Skript soll die Module getloopback.py und generatebgp.py durch import weiter verwerten. Öffnen Sie eine neue Datei pushbgp.py in pluma.

Fügen Sie die erforderlichen import Statements ein:

```
import getloopback as GL
import generatebgp as GB
import netmiko
import time
```

Das Paket time wird benötigt, um das Skript warten zu lassen, bis BGP konvergiert ist.

6.) Erzeugen Sie globale Variablen für die Management-Adressen der IOS-XE und NX-OS Systeme:

```
ip_nx = ['10.0.0.100']
ip_iosxe = ['10.0.0.101', '10.0.0.102']
```

Iterieren Sie die Listen um mit getloopback die zugehörigen Loopback0-Adressen zu ermitteln und in einer neuen Liste anzusammeln. Für das IOS-XE sieht das z.B. so aus:

```
lp_iosxe = []
for ip in ip_iosxe:
   lp_iosxe.append(GL.iosxe(ip, 'student', 'L1nux_dc'))
```

Fügen Sie den Code für das NX-OS hinzu! Hier soll die Liste lp_nx gefüllt werden. Orientieren Sie sich an dem Code für das IOS-XE!

Erzeugen Sie dann eine List mit allen Loopback0-Adressen:

```
lp_all = lp_nx + lp_iosxe
```

Ergänzen Sie ein print-Statement zur Ausgabe der Listenwerke:

```
print(lp_nx , lp_iosxe, lp_all)
```

7.) Iterieren Sie die Listen mit den Loopback0-Adressen lp_nx bzw. lp_iosxe, um
- mit generatebgp die BGP-Konfiguration mit ASN 65000 zu erzeugen
- mit netmiko die erzeugt BGP-Konfiguration an das Device mit der zugehörigen Management-Adresse zu pushen
- mit netmiko den Nachbar-Zustand prüfen

Für das NX-OS sieht das z.B. wie folgt aus:

```
for loop in lp_nx:
   bgp_nx = GB.nxos('65000' , loop , lp_all)
   session_nx = netmiko.ConnectHandler(ip = ip_nx[lp_nx.index(loop)] ,
                        username = 'student' ,
                        password = 'L1nux_dc' ,
                        device_type = "cisco_nxos")
   session_nx.send_config_set(config_commands = 'feature bgp')
   session_nx.send_config_set(config_commands = bgp_nx)
   time.sleep(3)
   print(session_nx.send_command('show bgp ipv4 unicast summary'))
   session_nx.disconnect()
```

Wozu wir die Built-In Funktion index() in der dritten Zeile verwendet? Wozu könnte time.sleep(3) nützlich sein?

Entwickeln sie eine weitere for-Loop für die IOS-XE Systeme, in dem sie die Loop für die NX-OS Systeme als Vorlage nutzen (feature bgp ist im IOS-XE nicht erforderlich). Die Variabel device-type hat hier den Wert “cisco-ios“.

Speichern Sie abschließend das Skript ab.

8.) Rufen Sie das Skript auf der Linux Shell auf und studieren Sie seinen Output. U.U. müssen Sie das Skript ein weiteres mal aufrufen, um funktionstüchtige BGP-Nachbarschaften zu sehen (Das Skript ist idempotent, sodass mehrfache Anwendung die Konfiguration der Systeme

nicht mehr ändert):

```
student@workplace:~/CPRN$ python3 pushbgp.py
10.255.255.100
10.255.255.101
10.255.255.102
['10.255.255.100'] ['10.255.255.101', '10.255.255.102']
['10.255.255.100', '10.255.255.101', '10.255.255.102']
BGP summary information for VRF default, address family IPv4 Unicast
BGP router identifier 10.255.255.100, local AS number 65000
BGP table version is 3, IPv4 Unicast config peers 2, capable peers 2
0 network entries and 0 paths using 0 bytes of memory
BGP attribute entries [0/0], BGP AS path entries [0/0]
BGP community entries [0/0], BGP clusterlist entries [0/0]

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.255.255.101  4 65000       2       4        3    0    0 00:00:18 0
10.255.255.102  4 65000       2       4        3    0    0 00:00:19 0

BGP router identifier 10.255.255.101, local AS number 65000
BGP table version is 1, main routing table version 1

Neighbor        V          AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.255.255.100  4       65000       4       2        1    0    0 00:00:22 0
10.255.255.102  4       65000       2       2        1    0    0 00:00:18 0

BGP router identifier 10.255.255.102, local AS number 65000
BGP table version is 1, main routing table version 1

Neighbor        V          AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.255.255.100  4       65000       4       2        1    0    0 00:00:28 0
10.255.255.101  4       65000       2       2        1    0    0 00:00:22 0
```

Verifizieren Sie über eine SSH/Telnet Session mit den Devices, dass die BGP-Konfiguration vorhanden ist und funktioniert:

```
N9Kv# show running-config bgp

!Command: show running-config bgp
!Running configuration last done at: Tue May 10 06:20:36 2022
!Time: Tue May 10 06:24:06 2022

version 10.2(2) Bios:version
feature bgp

router bgp 65000
```

```
  router-id 10.255.255.100
  neighbor 10.255.255.101
    remote-as 65000
    update-source loopback0
    address-family ipv4 unicast
      next-hop-self
  neighbor 10.255.255.102
    remote-as 65000
    update-source loopback0
    address-family ipv4 unicast
      next-hop-self

N9Kv# show bgp ipv4 unicast summary
BGP summary information for VRF default, address family IPv4 Unicast
BGP router identifier 10.255.255.100, local AS number 65000
BGP table version is 3, IPv4 Unicast config peers 2, capable peers 2
0 network entries and 0 paths using 0 bytes of memory
BGP attribute entries [0/0], BGP AS path entries [0/0]
BGP community entries [0/0], BGP clusterlist entries [0/0]

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.255.255.101  4 65000       7       8        3    0    0 00:04:10 0
10.255.255.102  4 65000       7       8        3    0    0 00:04:11 0

CSR-1#show running-config | section bgp
router bgp 65000
 bgp router-id 10.255.255.101
 bgp log-neighbor-changes
 neighbor 10.255.255.100 remote-as 65000
 neighbor 10.255.255.100 update-source Loopback0
 neighbor 10.255.255.100 next-hop-self
 neighbor 10.255.255.102 remote-as 65000
 neighbor 10.255.255.102 update-source Loopback0
 neighbor 10.255.255.102 next-hop-self


CSR-1#show bgp ipv4 unicast summary
BGP router identifier 10.255.255.101, local AS number 65000
BGP table version is 1, main routing table version 1

Neighbor        V           AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.255.255.100  4        65000       9       9        1    0    0 00:05:44        0
10.255.255.102  4        65000       9       9        1    0    0 00:05:40        0
```