Configureer MPLS L3VPN-service op PE-router met REST-API (IOS-XE)

Inhoud

Inleiding

Dit document toont het gebruik van Python-programmering aan om een MPLS L3VPN op een PoE-router (Service Provider Edge) te provisioneren met behulp van REST API. In dit voorbeeld worden Cisco CSR1000v (IOS-XE) routers als PE-routers gebruikt.

Bijdrage van: Anuradha Perera

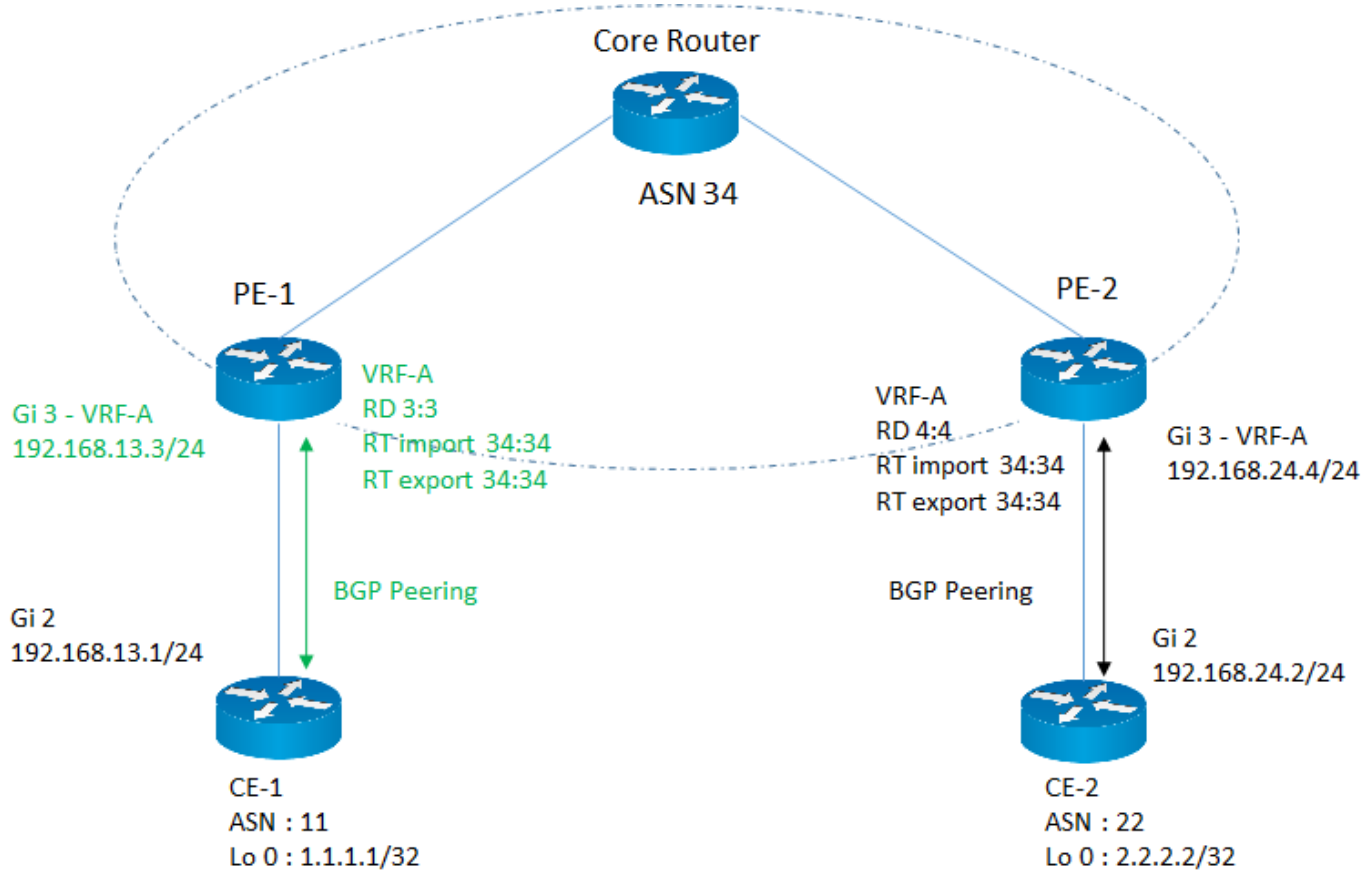
Bewerkt door: Kumar Sridhar

Voorwaarden

- REST API-beheertoegang tot CSR1000v-routers (raadpleeg de referenties aan het eind van dit document).
- Python (Versie 2.x of 3.x) en Python-bibliotheek "Verzoeken" geïnstalleerd op de computer die wordt gebruikt voor het configureren van de routers.
- Enkele basiskennis van Python-programmering.

Configuratie

Netwerkdiagram



In dit voorbeeld ligt de nadruk op het configureren van de vereiste MPLS L3VPN-serviceparameters op de PE-1 router, die in roze kleur worden gemarkeerd.

Configuratieprocedure

De configuratietaak is verdeeld in een aantal subtaken en elke subtaak wordt geïmplementeerd onder een door de gebruiker gedefinieerde functie. Zo kunnen functies worden hergebruikt wanneer dat nodig is.

Alle functies gebruiken de bibliotheek "Verzoeken" om toegang te krijgen tot REST API's op de router en dataformaat is JSON. In HTTP verzoeken "verify" wordt parameter ingesteld op "False" om het valideren van SSL certificaat te negeren.

1. Uploadtoken-id ophalen

Alvorens met om het even welke configuratie op een router te werk te gaan moet u een geldige token-id hebben die uit de router wordt verkregen. Deze functie initieert HTTP-verzoek om een token-id te authenticeren en te verkrijgen, zodat deze andere API's kan oproepen met deze token. Het antwoord op dit verzoek omvat een token-id.

```
#---------------------------------

def getToken (ip, poort, gebruikersnaam, wachtwoord):

aanvragen voor invoer

importbasis64

 url = "https://" + ip + "+ poort + "/api/v1/auth/token-services"

 headers = {

   "content-type": "application/json",

   "autorisatie": "Basic" + base64.b64encode((gebruikersnaam + ":" + wachtwoord).encode("UTF-8")).decode("ascii"),
```

```
    'cache-control': "zonder cache"
  }
  antwoord = request.request("POST", url, headers=headers, verify=False )
indien response.status_code == 200:
  return response.json()['token-id']
anders:
  terugkeren "mislukt"
#---------------------------------
```

2. VRF maken

Deze functie maakt de VRF op de PE-router met de vereiste routeonderscheidingsteken (RD) en import/export-routedoelen (RT)

```
#---------------------------------
def creatiefVRF (ip, poort, tokenID, vrfName, RD, importRT, exportRT):
aanvragen voor invoer
  url = "https://" + ip + ":"+ poort + "/api/v1/vrf"
  headers = {
    "content-type": "application/json",
    "X-auth-token": tokenID,
    'cache-control': "zonder cache"
  }
  gegevens = {
    "naam": vrfName,
    "rd": RD,
    "route-doelstelling": [
        {
            "actie" : "invoer",
```

```
        "Gemeenschap": importRT
      },
      {
        "actie" : "uitvoer",
        "gemeenschap": exportRT
      }
   ]
}
 antwoord = request.request("POST", url, headers=headers, json=data, verify=False )
indien response.status_code == 201:
   terugkeren "succesvol"
anders:
   terugkeren "mislukt"
#----------------------------------
```

3. Interface naar een VRF verplaatsen

Deze functie verplaatst een gegeven interface naar een VRF.

```
#----------------------------------
def addInterfacetoVRF (ip, poort, tokenID, vrfName, interfaceName, RD, importRT, exportRT):
aanvragen voor invoer
 url = "https://" + ip + ":"+ poort + "/api/v1/vrf/" + vrfName
 headers = {
   "content-type": "application/json",
   'X-auth-token': tokenID,
   'cache-control': "zonder cache"
   }
 gegevens = {
```

```
  "rd": RD,
  "doorsturen": [ interfaceName ],
  "route-doelstelling": [
     {
        "actie" : "invoer",
        "Gemeenschap": importRT
     },
     {
        "actie" : "uitvoer",
        "gemeenschap": exportRT
     }
  ]
}
 antwoord = request.request("PUT", url, headers=headers, json=data, verify=False )


indien response.status_code == 204:
  terugkeren "succesvol"
anders:
  terugkeren "mislukt"
#---------------------------------
```

4. IP-adres aan interface toewijzen

Deze functie wijst IP-adres toe aan de interface.

```
#---------------------------------
def. assignedInterfaceIP (ip, poort, token-id, interfaceName, interfaceIP, interfaceSubnet):
aanvragen voor invoer
 url = "https://" + ip + ":"+ poort + "/api/v1/interfaces/" + interfaceName
 headers = {
```

```
  "content-type": "application/json",
  "X-auth-token": tokenID,
  'cache-control': "zonder cache"
  }
 gegevens = {
  "type": "Ethernet",
  "if-name": interfaceName;
  'ip-adres': interfaceIP,
  'subnetmasker': interfaceSubnet
  }
 antwoord = request.request("PUT", url, headers=headers, json=data, verify=False )
indien response.status_code == 204:
  retourneren "geslaagd"
anders:
  retourneren "mislukt"
#---------------------------------
```

5. VRF-bewuste bgp maken

Hierdoor wordt VRF-adresfamilie ipv4 ingeschakeld.

```
#---------------------------------
def. creeerVrfBGP (ip, poort, tokenID, vrfName, ASN):
aanvragen voor invoer
 url = "https://" + ip + ":"+ poort + "/api/v1/vrf/" + vrfName + "/routing-svc/bgp"
 headers = {
  "content-type": "application/json",
  "X-auth-token": tokenID,
  'cache-control': "zonder cache"
  }
```

```
 gegevens = {
   'routing-protocol-id': ASN
   }
 antwoord = request.request("POST", url, headers=headers, json=data, verify=False )
indien response.status_code == 201:
   terugkeren "succesvol"
anders:
   terugkeren "mislukt"
#----------------------------------
```

6. Definieer BGP-buur onder VRF-adresfamilie

Deze functie definieert BGP-buur onder VRF-adresfamilie IPV4.

```
#----------------------------------
defdefinedVrfBGPeighbor (ip, poort, tokenID, vrfName, ASN, buurtIP, remoteAS):
aanvragen voor invoer
 url = "https://" + ip + ":"+ poort + "/api/v1/vrf/" + vrfName + "/routing-svc/bgp/" + ASN +"/buren"
 headers = {
   "content-type": "application/json",
   "X-auth-token": tokenID,
   'cache-control': "zonder cache"
   }
 gegevens = {
   "Routing-Protocol-ID": ASN;
   "adres": buurIP,
   'afstandsbediening': afstandsbediening
   }
 antwoord = request.request("POST", url, headers=headers, json=data, verify=False )
 indien response.status_code == 201:
```

```
    terugkeren "succesvol"
anders:
    terugkeren "mislukt"
#---------------------------------
```

Beschrijving en waarden van invoerparameters

```
ip = "10.0.0.1"                          # ip adres van de router
poort = "55443"                          # REST API-poort op router
gebruikersnaam = "cisco"                 # gebruikersnaam om in te loggen.Dit moet worden geconfigureerd met prioriteitsniveau 15.
wachtwoord = "cisco"                     # wachtwoord gekoppeld aan gebruikersnaam
tokenID = <waarde return> # Token ID verkregen uit router met behulp van de getToken functie
vrfName = "VRF-A" # naam van de VRF
RD = "3:3" # routeonderscheidingsteken voor VRF
importRT = "34:34" # Routedoel importeren
exportRT = "34:34" # exportroutedoel
interfaceName = "Gigabit Ethernet3" # naam van de Customer Edge (CE) facing-interface
interfaceIP = "192.168.13.3" # IP-adres van CE-interface
interfaceSubnet = "255.255.255.0" # subnett van CE-interface
ASN = "34" # BGP AS-nummer van PE-router
NeighbourIP = "192.168.13.1" # BGP peering IP van CE-router
remoteAS = "11" # AS aantal CE-router
```

In alle bovengenoemde functies zijn speciale API's voor elke configuratie opgeroepen. Het voorbeeld hieronder toont aan hoe IOS-XE CLI in het algemeen, in het lichaam van REST API vraag over te gaan. Dit kan als tijdelijke oplossing worden gebruikt om te automatiseren als bepaalde API niet beschikbaar is. In de bovenstaande functies is 'content-type' ingesteld op 'application/json', maar in het onderstaande voorbeeld 'content-type' is ingesteld op 'text/plain' als het standaard CLI input parseert.

Dit voorbeeld definieert een interfacebeschrijving voor de interface Gigabit Ethernet3. De configuratie kan worden aangepast door de parameter "cliInput" te wijzigen.

```
#---------------------------------
def passCLIIinvoer (ip, poort, tokenID):
```

```
aanvragen voor invoer
 url = "https://" + ip +":"+ poort + "/api/v1/global/running-config"
 headers = {
   "content-type": "tekst/effen",
  "X-auth-token": tokenID,
   'cache-control': "zonder cache"
   }
 line1 = "Interface Gigabit Ethernet 3"
 line2 = "Beschrijving Customer Facing Interface"
 cliInput = line1 + "\r\n" + line2
 antwoord = request.request("PUT", url, headers=headers, data=cliInput, verify=False )
 afdrukken (response.text)
indien response.status_code == 204:
   terugkeren "Succesvol"
anders:
   terugkeren "mislukt"
#---------------------------------
```

Referenties

- Software voor Cisco CRS-1000v Series softwaregids voor cloudservices router

https://www.cisco.com/c/en/us/td/docs/routers/csr1000/software/configuration/b_CSR1000v_Configuration_Guide /b_CSR1000v_Configuration_Guide_chapter_01101.html

- Naslaghandleiding voor Cisco IOS XE REST API-beheer

https://www.cisco.com/c/en/us/td/docs/routers/csr1000/software/restapi/restapi.html

Gebruikte afkortingen:

MPLS - Multi Protocol Label Switching

L3 - Layer 3

VPN - Virtual Private Network

VRF - virtueel routedoorsturen

BGP-protocol voor grensgateway

REST - representatieve staatsoverdracht

API - Application Program Interface

JSON - JavaScript Object Notation

HTTP - Hyper-Text Transfer Protocol

Over deze vertaling

Cisco heeft dit document vertaald via een combinatie van machine- en menselijke technologie om onze gebruikers wereldwijd ondersteuningscontent te bieden in hun eigen taal. Houd er rekening mee dat zelfs de beste machinevertaling niet net zo nauwkeurig is als die van een professionele vertaler. Cisco Systems, Inc. is niet aansprakelijk voor de nauwkeurigheid van deze vertalingen en raadt aan altijd het oorspronkelijke Engelstalige document (link) te raadplegen.