REST-API(IOS-XE)를 사용하여 PE 라우터에서 MPLS L3VPN 서비스 구성

목차

소개

이 문서에서는 REST API를 사용하여 PE(Service Provider Edge) 라우터에서 MPLS L3VPN을 프로비저닝하기 위해 Python 프로그래밍을 사용하는 방법을 설명합니다. 이 예에서는 Cisco CSR1000v(IOS-XE) 라우터를 PE 라우터로 사용합니다.

: Anuradha Perera

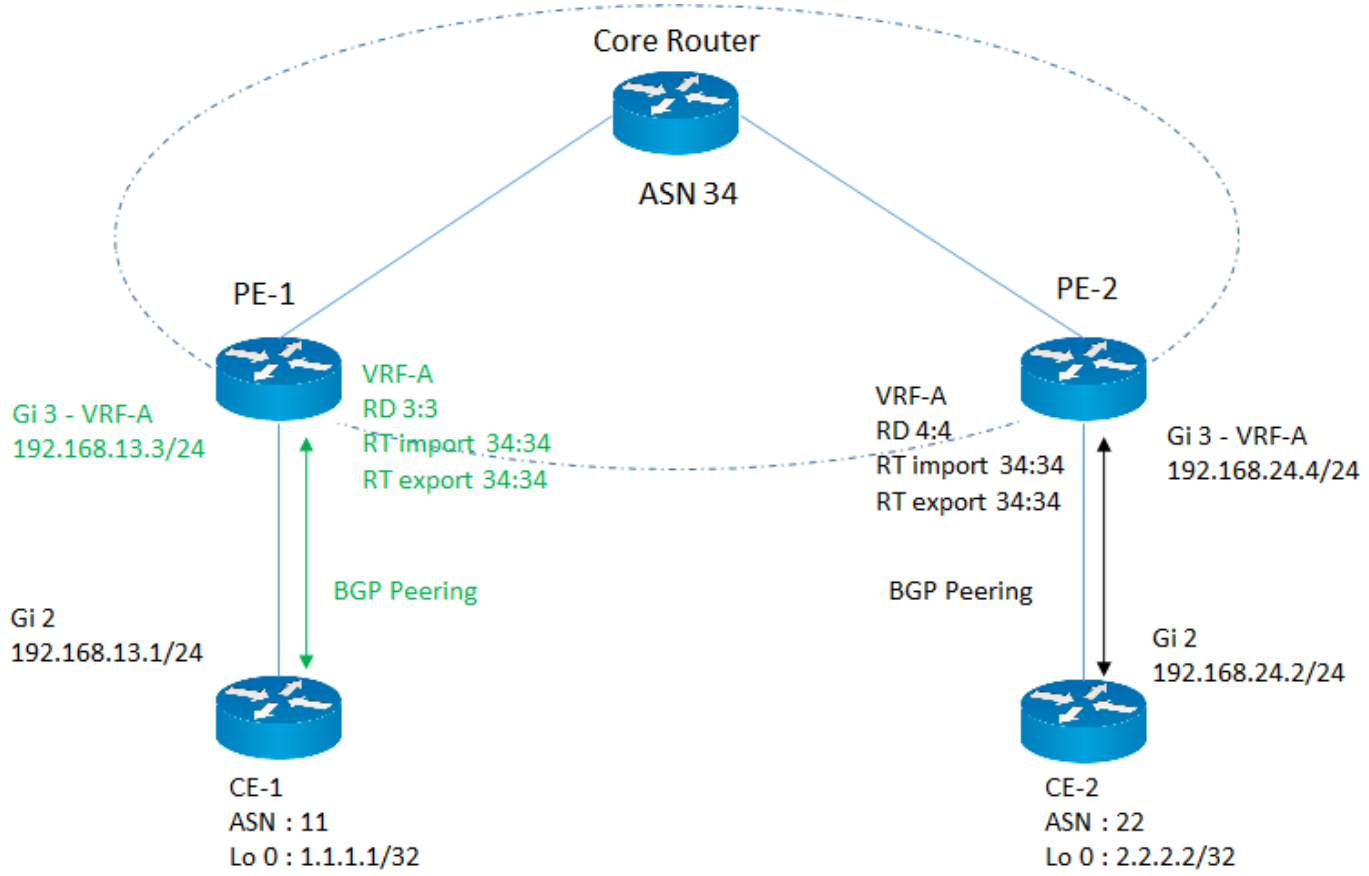
: Kumar Sridhar

사전 요구 사항

- CSR1000v 라우터에 대한 REST API 관리 액세스(이 문서 끝의 참조).
- Python(버전 2.x 또는 3.x) 및 "Requests" 라우터를 구성하는 데 사용되는 컴퓨터에 설치된 Python 라이브러리입니다.
- Python 프로그래밍에 대한 기본적인 지식.

설정

네트워크 다이어그램



이 예에서는 분홍색으로 강조 표시된 PE-1 라우터에서 필수 MPLS L3VPN 서비스 매개변수를 구성하는 데 중점을 둡니다.

컨피그레이션 절차

컨피그레이션 작업은 여러 하위 작업으로 나뉘며 각 하위 작업은 사용자 정의 기능으로 구현됩니다. 이러한 방식으로 기능은 필요할 때 재사용될 수 있다.

모든 기능은 라우터의 REST API에 액세스하기 위해 "요청" 라이브러리를 사용하며 데이터 형식은 JSON입니다. HTTP 요청에서 "verify" 매개변수는 SSL 인증서 검증을 무시하도록 "False"로 설정됩니다.

1. token-id 검색

라우터에서 컨피그레이션을 진행하기 전에 라우터에서 얻은 유효한 token-id가 있어야 합니다. 이 함수는 토큰 ID를 인증하고 얻기 위해 HTTP 요청을 시작하여 이 토큰을 사용하여 다른 API를 호출할 수 있습니다. 이 요청의 응답에는 token-id가 포함됩니다.

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

def getToken(ip, port, username, password):

가져오기 요청

가져오기 기본64

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

 헤더 = {

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

   '권한 부여': "기본 " + base64.b64encode((사용자 이름 + ":" + 비밀번호).encode('UTF-8')).decode('ascii'),
```

```
    'cache-control': "no-cache"
}
 response = requests.request("POST", url, headers=headers, verify=False )
response.status_code == 200인 경우:
 return response.json()['token-id']
기타:
 반환 "실패"
#---------------------------------
```

2. VRF 생성

이 기능은 필수 RD(Route Distinguisher) 및 RT(Route Target) 가져오기/내보내기를 통해 PE 라우터에 VRF를 생성합니다.

```
#---------------------------------
def createVRF(ip, port, tokenID, vrfName, RD, importRT, exportRT):
가져오기 요청
 url = "https://" + ip + ":"+ 포트 + "/api/v1/vrf"
 헤더 = {
   'content-type': "application/json",
   'X-auth-token': 토큰 ID,
   'cache-control': "no-cache"
}
 데이터 = {
   '이름': vrfName,
   'rd': RD,
   'route-target' : [
      {
         '작업' : "가져오기",
```

```
        '커뮤니티': importRT
    },
    {
        '작업' : "내보내기",
        '커뮤니티': exportRT
    }
  ]
}
 response = requests.request("POST", url, headers=headers, json=data, verify=False )
response.status_code == 201인 경우:
  반환 "성공"
기타:
  반환 "실패"
#----------------------------------
```

3 . 인터페이스를 VRF로 이동

이 기능은 지정된 인터페이스를 VRF로 이동합니다.

```
#----------------------------------
def addInterfacetoVRF(ip, port, tokenID, vrfName, interfaceName, RD, importRT, exportRT):
가져오기 요청
 url = "https://" + ip + ":"+ 포트 + "/api/v1/vrf/" + vrf 이름
 헤더 = {
  'content-type': "application/json",
  'X-auth-token': 토큰 ID,
  'cache-control': "no-cache"
  }
 데이터 = {
```

```
  'rd': RD,
  '전달': [ interfaceName ],
  'route-target' : [
     {
        'action' : "import",
        '커뮤니티': importRT
     },
     {
        'action' : "export",
        '커뮤니티': exportRT
     }
  ]
}
응답 = requests.request("PUT", url, headers=headers, json=data, verify=False )

response.status_code == 204인 경우:
  반환 "성공"
기타:
  반환 "실패"
#---------------------------------
```

4. 인터페이스에 IP 주소 할당

이 기능은 인터페이스에 ip 주소를 할당합니다.

```
#---------------------------------
def assignInterfaceIP(ip, port, tokenID, interfaceName, interfaceIP, interfaceSubnet):
가져오기 요청
 url = "https://" + ip + ":"+ 포트 + "/api/v1/interfaces/" + interfaceName
 헤더 = {
```

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

'X-auth-token': 토큰 ID,

'cache-control': "no-cache"

}

데이터 = {

'유형': "이더넷",

'if-name': 인터페이스 이름,

'ip-address': interfaceIP,

'subnet-mask': interfaceSubnet

}

응답 = requests.request("PUT", url, headers=headers, json=data, verify=False)

response.status_code == 204인 경우:

"성공" 반환

기타:

"실패" 반환

#----------

5. VRF 인식 bgp 생성

그러면 VRF 주소군 ipv4가 활성화됩니다.

#----------

def createVrfBGP(ip, port, tokenID, vrfName, ASN):

가져오기 요청

url = "https://" + ip + ":"+ 포트 + "/api/v1/vrf/" + vrfName + "/routing-svc/bgp"

헤더 = {

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

'X-auth-token': 토큰 ID,

'cache-control': "no-cache"

}

```
 데이터 = {
   'routing-protocol-id': ASN
   }
 response = requests.request("POST", url, headers=headers, json=data, verify=False )
response.status_code == 201인 경우:
   반환 "성공"
기타:
   반환 "실패"
#----------------------------------
```

6. VRF 주소군 아래에 BGP 인접 디바이스 정의

이 기능은 VRF 주소군 IPV4 아래에서 BGP 인접 디바이스를 정의합니다.

```
#----------------------------------
def defineVrfBGPNeigbor(ip, port, tokenID, vrfName, ASN, neighbourIP, remoteAS):
가져오기 요청
 url = "https://" + ip + ":"+ 포트 + "/api/v1/vrf/" + vrfName + "/routing-svc/bgp/" + ASN +"/neighbors"
 헤더 = {
   'content-type': "application/json",
   'X-auth-token': 토큰 ID,
   'cache-control': "no-cache"
   }
 데이터 = {
   'routing-protocol-id': ASN,
   '주소': neighbourIP,
   'remote-as': remoteAS
   }
 response = requests.request("POST", url, headers=headers, json=data, verify=False )
 response.status_code == 201인 경우:
```

```
  반환 "성공"

기타:

  반환 "실패"

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

입력 매개 변수의 설명 및 값

```
ip = "10.0.0.1 "                          라우터의 ip 주소 #

포트 = "55443"                            라우터의 REST API 포트 수

사용자 이름 = "cisco"                      # 로그인할 사용자 이름. 권한 레벨 15로 구성해야 합니다
.

암호 = "cisco"                          # 사용자 이름과 연결된 비밀번호

tokenID = <값 반환됨> # getToken 함수를 사용하여 라우터에서 얻은 토큰 ID

vrfName = "VRF-A" # VRF 이름

RD = "3:3" # VRF용 경로 구별자

importRT = "34:34" # 경로 대상 가져오기

exportRT = "34:34" # 경로 대상 내보내기

interfaceName = CE(customer edge) 연결 인터페이스의 "GigabitEthernet3" # 이름

interfaceIP = "192.168.13.3" # CE 연결 인터페이스의 IP 주소

interfaceSubnet = "255.255.255.0" # CE 대면 인터페이스의 서브넷

ASN = "34" # BGP AS PE 라우터 수

neighbourIP = "192.168.13.1" # CE 라우터의 BGP 피어링 IP

remoteAS = "11" # CE 라우터 수
```

위의 모든 기능에서 각 컨피그레이션 단계에 대해 전용 API가 호출되었습니다. 아래 예는 일반적으로 REST API 호출의 본문에서 IOS-XE CLI를 전달하는 방법을 보여줍니다. 이는 특정 API를 사용할 수 없는 경우 자동화하는 해결 방법으로 사용할 수 있습니다. 위 함수에서는 'content-type'이 'application/json'으로 설정되었으나, 아래 예에서는 'content-type'이 표준 CLI 입력을 구문 분석 중이므로 'text/plain'으로 설정됩니다.

이 예에서는 인터페이스 GigabitEthernet3에 대한 인터페이스 설명을 정의합니다. 컨피그레이션은 "cliInput" 매개변수를 변경하여 사용자 지정할 수 있습니다.

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

def passCLInput(ip, 포트, tokenID):
```

가져오기 요청

```
url = "https://" + ip + ":"+ 포트 + "/api/v1/global/running-config"
헤더 = {
  'content-type': "text/plain",
 'X-auth-token': 토큰 ID,
  'cache-control': "no-cache"
  }
line1 = "인터페이스 기가비트 이더넷 3"
line2 = "고객 응대 인터페이스 설명"
cliInput = line1 + "\r\n" + line2
response = requests.request("PUT", url, headers=headers, data=cliInput, verify=False )
print(response.text)
```

response.status_code == 204인 경우:

  반환 "성공"

기타:

  반환 "실패"

#---------------------------------

참조

- Cisco CSR 1000v Series Cloud Services Router 소프트웨어 컨피그레이션 가이드

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

- Cisco IOS XE REST API 관리 참조 설명서

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

사용된 약어:

MPLS - 다중 프로토콜 레이블 스위칭

L3 - 레이어 3

VPN - 가상 사설망

VRF - 가상 경로 전달

BGP - 보더 게이트웨이 프로토콜

REST - REST(Representational State Transfer)

API - Application Program Interface

JSON - Java 스크립트 객체 표기법

HTTP - 하이퍼 텍스트 전송 프로토콜

이 번역에 관하여

Cisco는 전 세계 사용자에게 다양한 언어로 지원 콘텐츠를 제공하기 위해 기계 번역 기술과 수작업 번역을 병행하여 이 문서를 번역했습니다. 아무리 품질이 높은 기계 번역이라도 전문 번역가의 번역 결과물만큼 정확하지는 않습니다. Cisco Systems, Inc.는 이 같은 번역에 대해 어떠한 책임도 지지 않으며 항상 원본 영문 문서(링크 제공됨)를 참조할 것을 권장합니다.