REST-API(IOS-XE)を使用したPEルータでのMPLS L3VPNサービスの設定

内容

概要

このドキュメントでは、Pythonプログラミングを使用して、REST APIを使用してサービスプロバイダーエッジ(PE)ルータ上にMPLS L3VPNをプロビジョニングする方法について説明します。この例では、PEルータとしてCisco CSR1000v(IOS-XE)ルータを使用しています。

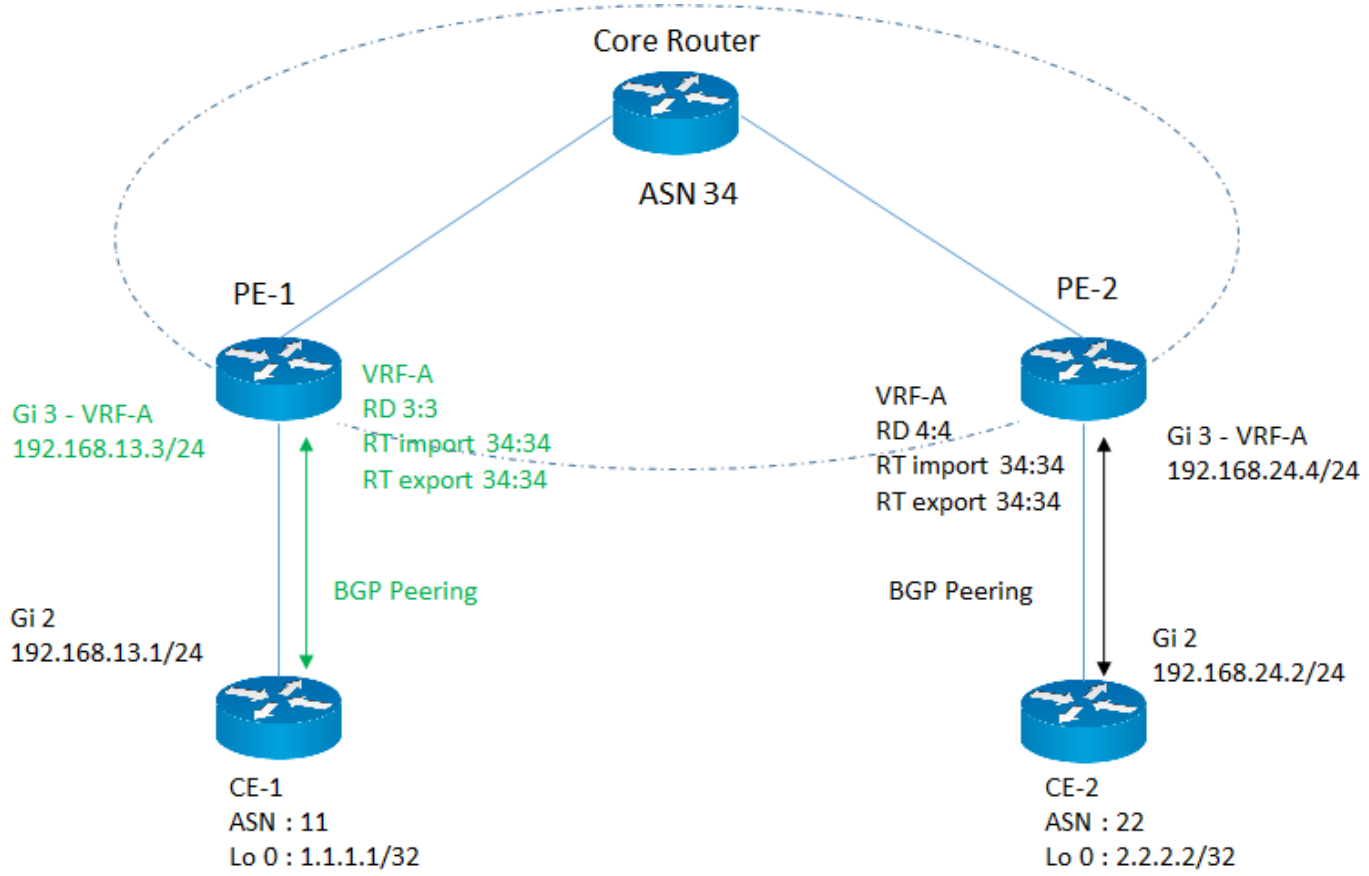
Anuradha Perera

前提条件

- CSR1000vルータへのREST API管理アクセス（このドキュメントの最後にある「参考資料」を参照）。
- ルータの設定に使用するコンピュータにインストールされたPython（バージョン2.xまたは3.x）および「要求」Pythonライブラリ。
- Pythonプログラミングに関する基本的な知識

コンフィギュレーション

ネットワーク図



この例では、ピンク色で強調表示されているPE-1ルータで必要なMPLS L3VPNサービスパラメータを設定することに焦点を当てています。

構成手順

設定タスクは複数のサブタスクに分割され、各サブタスクはユーザ定義関数の下に実装されます。これにより、必要に応じて機能を再利用できます。

すべての関数は「リクエスト」ライブラリを使用してルータ上のREST APIにアクセスし、データ形式はJSONです。 HTTP要求では、SSL証明書の検証を無視するために「verify」パラメータが「False」に設定されます。

1.トークンIDの取得

ルータの設定に進む前に、ルータから取得した有効なトークンIDが必要です。この関数は、HTTP要求を開始して認証を行い、トークンIDを取得します。これにより、このトークンを使用して他のAPIを呼び出すことができます。この要求の応答にはトークンIDが含まれます。

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

def getToken（ip、ポート、ユーザ名、パスワード）:

輸入要求

import base64

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

 ヘッダー= {

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

   'authorization': "Basic " + base64.b64encode((username + ":" + password).encode('UTF-8')).decode('ascii'),
```

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

それ以外の場合：

戻る "failed"

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

2. VRFの作成

この機能は、必要なルート識別子(RD)とインポート/エクスポートルートターゲット(RT)を持つPEルータ上にVRFを作成します

```
#---------------------------------
def createVRF(ip、port、tokenID、vrfName、RD、importRT、exportRT):
```

輸入要求

```
 url = "https://" + ip + ":"+ port + "/api/v1/vrf"
```

ヘッダー= {

```
   'content-type': "application/json",
   'X-auth-token':tokenID、
   'cache-control': "no-cache"
}
```

データ= {

```
   'name':vrfName、
    「rd」:RD、
   'route-target': [
      {
          'action':"import",
```

```
        'community' : importRT
    },
    {
        'action':"export",
        'community' : exportRT
    }
  ]
}
 response = requests.request("POST", url, headers=headers, json=data, verify=False )
response.status_code == 201:
  戻る "成功"
それ以外の場合：
  戻る "failed"
#----------------------------------
```

を選択します。 インターフェイスをVRFに移動する

この機能は、特定のインターフェイスをVRFに移動します。

```
#----------------------------------
def addInterfacetoVRF(ip、port、tokenID、vrfName、interfaceName、RD、importRT、
exportRT):
輸入要求
 url = "https://" + ip + ":"+ port + "/api/v1/vrf/" + vrfName
 ヘッダー= {
  'content-type': "application/json",
  'X-auth-token': tokenID,
  'cache-control': "no-cache"
  }
```

```
データ= {
  「rd」:RD、
  'forwarding': [ interfaceName ],
  'route-target': [
    {
      'action' : "import",
      'community' : importRT
    },
    {
      'action' : "export",
      'community' : exportRT
    }
  ]
}
 response = requests.request("PUT", url, headers=headers, json=data, verify=False )


response.status_code == 204:
  戻る "成功"
それ以外の場合：
  戻る "failed"
#----------------------------------
```

4.インターフェイスへのIPアドレスの割り当て

この機能は、インターフェイスにIPアドレスを割り当てます。

```
#----------------------------------
def assignInterfaceIP(ip、port、tokenID、interfaceName、interfaceIP、interfaceSubnet):
輸入要求
 url = "https://" + ip + ":"+ port + "/api/v1/interfaces/" + interfaceName
```

ヘッダー= {

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

'X-auth-token':tokenID、

'cache-control': "no-cache"

}

データ= {

'type': "ethernet",

'if-name':interfaceName,

「ip-address」:interfaceIP、

「サブネットマスク」:interfaceSubnet

}

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

response.status_code == 204:

「successful」を返します

それ以外の場合：

return "failed"

#----------

5. VRF対応BGPの作成

これにより、VRFアドレスファミリipv4が有効になります。

#----------

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

輸入要求

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

ヘッダー= {

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

'X-auth-token':tokenID、

'cache-control': "no-cache"

```
  }
 データ= {
  'routing-protocol-id':ASN
  }
 response = requests.request("POST", url, headers=headers, json=data, verify=False )
response.status_code == 201:
  戻る "成功"
それ以外の場合 :
  戻る "failed"
#----------------------------------
```

6. VRFアドレスファミリでBGPネイバーを定義する

この機能は、VRFアドレスファミリIPV4でBGPネイバーを定義します。

```
#----------------------------------
def defineVrfBGPNeighbor(ip、port、tokenID、vrfName、ASN、neighbourIP、remoteAS):
輸入要求
 url = "https://" + ip + ":"+ port + "/api/v1/vrf/" + vrfName + "/routing-svc/bgp/" + ASN +"/neighbors"
 ヘッダー= {
  'content-type': "application/json",
  'X-auth-token':tokenID、
  'cache-control': "no-cache"
  }
 データ= {
   「routing-protocol-id」:ASN、
   「address」:neighbourIP、
   「remote-as」:remoteAS
  }
 response = requests.request("POST", url, headers=headers, json=data, verify=False )
```

response.status_code == 201:

戻る "成功"

それ以外の場合：

戻る "failed"

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

入力パラメータの説明と値

ip = "10.0.0.1" # ip address of the router

ポート= "55443" #ルータのREST APIポート

ユーザ名= "cisco" #ログインするユーザ名。これは特権レベル15で設定する必要があります。

パスワード= "cisco" #ユーザ名に関連付けられたパスワード

tokenID = <返された値> # getToken関数を使用してルータから取得したトークンID

vrfName = "VRF-A" # VRFの名前

RD = "3:3" # VRFのルート識別子

importRT = "34:34" # Import Route Target

exportRT = "34:34" # export Route Target

interfaceName = "GigabitEthernet3" #カスタマーエッジ(CE)側インターフェイスの名前

interfaceIP = "192.168.13.3" # CE側インターフェイスのIPアドレス

interfaceSubnet = "255.255.255.0" # CE側インターフェイスのサブネット

ASN = "34" # BGP AS番号（PEルータの数）

neighbourIP = "192.168.13.1" # CEルータのBGPピアリングIP

remoteAS = "11" # CEルータのAS番号

上記のすべての関数では、各設定ステップに対して専用のAPIが呼び出されました。次の例は、IOS-XE CLIを渡す方法を示しています。一般的には、REST API呼び出しの本文で渡します。これは、特定のAPIが使用できない場合に自動化する回避策として使用できます。上の関数では'content-type'は'application/json'に設定されていますが、下の例では'content-type'は標準のCLI入力を解析しているため、'text/plain'に設定されています。

この例では、インターフェイスGigabitEthernet3のインターフェイスの説明を定義します。 設定は、「cliInput」パラメータを変更することでカスタマイズできます。

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

```
def passCLIInput(ip、port、tokenID):
輸入要求
 url = "https://" + ip + ":" + port + "/api/v1/global/running-config"
 ヘッダー= {
   'content-type': "text/plain",
  'X-auth-token':tokenID、
   'cache-control': "no-cache"
   }
 line1 = "Interface GigabitEthernet 3"
 line2 = "description Customer Facing Interface"
 cliInput = line1 + "\r\n" + line2
 response = requests.request("PUT", url, headers=headers, data=cliInput, verify=False )
 print(response.text)
response.status_code == 204:
   戻る 「成功」
それ以外の場合 :
   戻る "failed"
#---------------------------------
```

参考資料

- 『Cisco CSR 1000v Series Cloud Services Router Software Configuration Guide』

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 Management Reference Guide』

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

使用する略語:

MPLS:Multi Protocol Label Switching

L3 – レイヤ3

VPN:Virtual Private Network

VRF:Virtual Route Forwarding（仮想ルート転送）

BGP（ボーダーゲートウェイプロトコル）

REST:Representational State Transfer(Representational State Transfer)

API:Application Program Interface

JSON - Javaスクリプトオブジェクト表記

HTTP:Hyper Text Transfer Protocol（ハイパーテキスト転送プロトコル）

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。