Configuración del servicio MPLS L3VPN en el router PE mediante REST-API (IOS-XE)

Contenido

Introducción
Prerequisites
_
Configuración
Diagrama de la red
Procedimiento de Configuración
1. Recuperar ID de token
2. Crear VRF
3. Mover interfaz a un VRF
4. Asignar dirección IP a la interfaz
5. Cree bgp que reconozca VRF
6. Defina el vecino BGP bajo la familia de direcciones VRF
Referencias
Acrónimos utilizados:

Introducción

Este documento demuestra el uso de la programación Python para aprovisionar una MPLS L3VPN en un router de extremo del proveedor de servicios (PE) usando la API REST. Este ejemplo utiliza routers Cisco CSR1000v (IOS-XE) como routers PE.

Gentileza de: Anuradha Perera

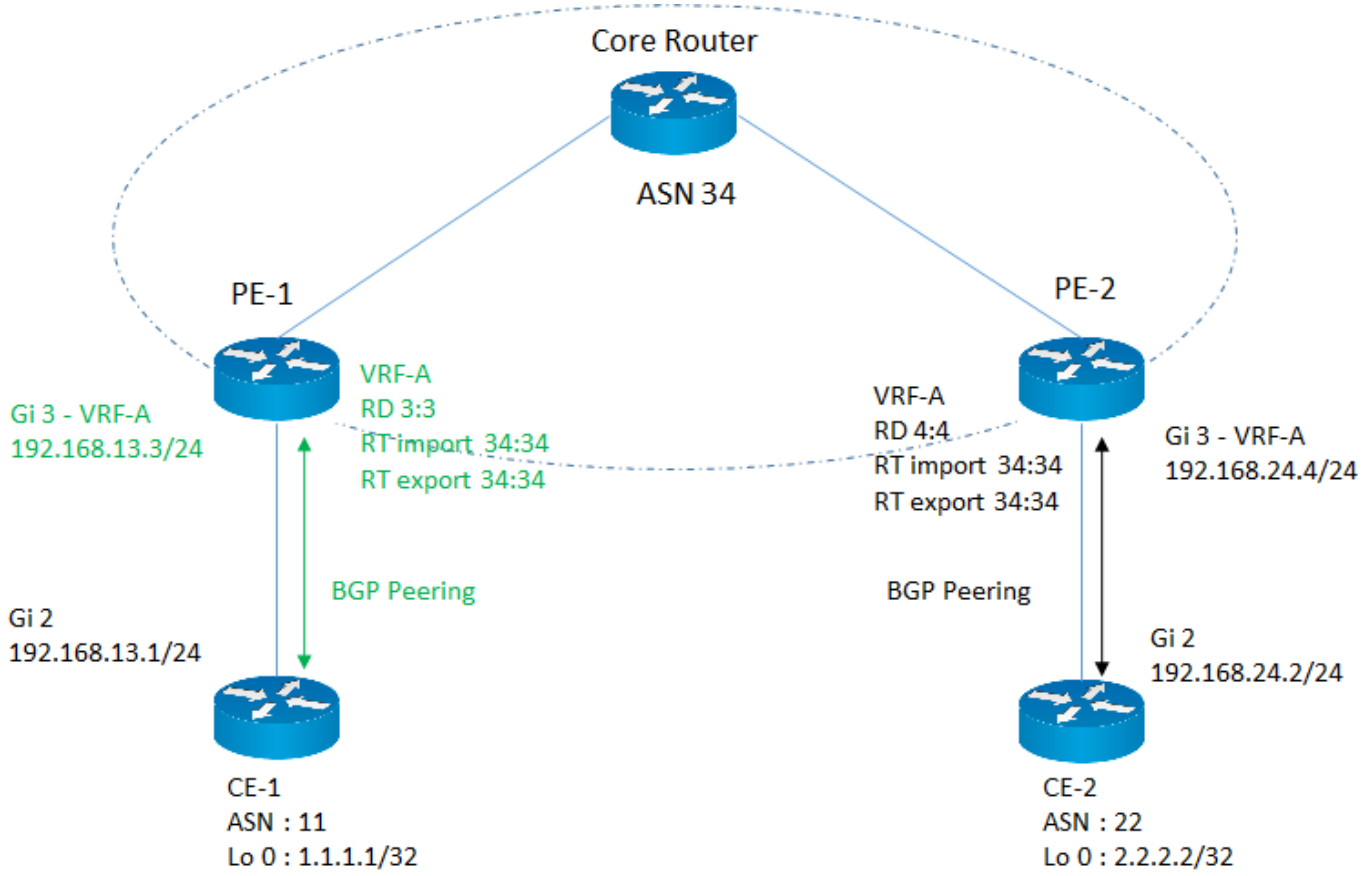
Editado por: Kumar Sridhar

Prerequisites

- Acceso de administración de API REST a routers CSR1000v (consulte las referencias al final de este documento).
- Biblioteca de Python (versión 2.x o 3.x) y "Solicitudes" de Python instalada en el equipo utilizado para configurar los routers.
- Algunos conocimientos básicos sobre programación Python.

Configuración

Diagrama de la red



En este ejemplo, el enfoque está en configurar los parámetros de servicio MPLS L3VPN requeridos en el router PE-1, que están resaltados en color rosa.

Procedimiento de Configuración

La tarea de configuración se divide en varias subtareas y cada subtarea se implementa bajo una función definida por el usuario. De este modo, las funciones se pueden reutilizar cuando sea necesario.

Todas las funciones utilizan la biblioteca de solicitudes para acceder a las API REST en el router y el formato de datos es JSON. En solicitudes HTTP, el parámetro "verify" se establece en "False" para omitir la validación del certificado SSL.

1. Recuperar ID de token

Antes de continuar con cualquier configuración en un router, debe tener un token-id válido obtenido del router. Esta función inicia una solicitud HTTP para autenticar y obtener un token-id de modo que pueda invocar otras API mediante este token. La respuesta de esta solicitud incluye un token-id.

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

def getToken (ip, puerto, nombre de usuario, contraseña):

solicitudes de importación

import base64

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

 encabezados = {

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

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

```
    'cache-control': "no-cache"
  }
  response = requests.request("POST", url, encabezados=encabezados, verify=False )
si response.status_code == 200:
  return response.json()['token-id']
de lo contrario:
  volver "fallado"
#----------------------------------
```

2. Crear VRF

Esta función creará el VRF en el router PE con el discriminador de rutas (RD) requerido y los destinos de rutas de importación/exportación (RT)

```
#----------------------------------
def createVRF (ip, port, tokenID, vrfName, RD, importRT, exportRT):
solicitudes de importación
  url = "https://" + ip + ":"+ puerto + "/api/v1/vrf"
  encabezados = {
    'content-type': "application/json",
    'X-auth-token': tokenID,
    'cache-control': "no-cache"
  }
  datos = {
    'name': vrfName,
    'rd': RD,
    'route-target': [
        {
            'action' : "import",
```

```
            'community' : importRT
        },
        {
            'action' : "export",
            'community' : exportRT
        }
    ]
}
 response = requests.request("POST", url, encabezados=encabezados, json=data, verify=False )
if response.status_code == 201:
    volver "satisfactorio"
de lo contrario:
    volver "fallado"
#----------------------------------
```

3. Mover interfaz a un VRF

Esta función moverá una interfaz determinada a un VRF.

```
#----------------------------------
def addInterfacetoVRF (ip, puerto, tokenID, vrfName, interfaceName, RD, importRT, exportRT):
solicitudes de importación
 url = "https://" + ip + ":"+ puerto + "/api/v1/vrf/" + vrfName
 encabezados = {
   'content-type': "application/json",
   'X-auth-token': tokenID,
   'cache-control': "no-cache"
   }
 datos = {
```

```
  'rd': RD,
  'reenvío': [ nombreDeInterfaz ],
  'route-target': [
     {
        'action' : "import",
        'community' : importRT
     },
     {
        'action' : "export",
        'community' : exportRT
     }
  ]
 }
 response = requests.request("PUT", url, encabezados=encabezados, json=datos, verify=False )


if response.status_code == 204:
   volver "satisfactorio"
de lo contrario:
   volver "fallado"
#---------------------------------
```

4. Asignar dirección IP a la interfaz

Esta función asignará la dirección IP a la interfaz.

```
#---------------------------------
def assignInterfaceIP (ip, port, tokenID, interfaceName, interfaceIP, interfaceSubnet):
solicitudes de importación
 url = "https://" + ip + ":"+ port + "/api/v1/interfaces/" + interfaceName
 encabezados = {
```

```
  'content-type': "application/json",
  'X-auth-token': tokenID,
  'cache-control': "no-cache"
  }
 datos = {
  'type': "ethernet",
  'if-name': interfaceName,
  'ip-address': interfaceIP,
  'subnet-mask': interfaceSubnet
  }
 response = requests.request("PUT", url, encabezados=encabezados, json=datos, verify=False )
if response.status_code == 204:
  devolver "correcto"
de lo contrario:
  return "failed"
#---------------------------------
```

5. Cree bgp que reconozca VRF

Esto habilitará la familia de direcciones VRF ipv4.

```
#---------------------------------
def createVrfBGP (ip, port, tokenID, vrfName, ASN):
solicitudes de importación
 url = "https://" + ip + ":"+ puerto + "/api/v1/vrf/" + vrfName + "/routing-svc/bgp"
 encabezados = {
  'content-type': "application/json",
  'X-auth-token': tokenID,
  'cache-control': "no-cache"
  }
```

```
 datos = {
   'routing-protocol-id': ASN
   }
 response = requests.request("POST", url, encabezados=encabezados, json=data, verify=False )
if response.status_code == 201:
   volver "satisfactorio"
de lo contrario:
   volver "fallado"
#---------------------------------
```

6. Defina el vecino BGP bajo la familia de direcciones VRF

Esta función definirá al vecino BGP bajo la familia de direcciones VRF IPV4.

```
#---------------------------------
def defineVrfBGPNeighbor (ip, puerto, tokenID, vrfName, ASN, neighborIP, remoteAS):
solicitudes de importación
 url = "https://" + ip + ":"+ puerto + "/api/v1/vrf/" + vrfName + "/routing-svc/bgp/" + ASN
+"/neighbors"
 encabezados = {
   'content-type': "application/json",
   'X-auth-token': tokenID,
   'cache-control': "no-cache"
   }
 datos = {
   'routing-protocol-id': ASN,
   'address': neighborIP,
   'remote-as': remoteAS
   }
 response = requests.request("POST", url, encabezados=encabezados, json=data, verify=False )
```

```
 if response.status_code == 201:
   volver "satisfactorio"
de lo contrario:
   volver "fallado"
#---------------------------------
```

Descripción y valores de los parámetros de entrada

```
ip = "10.0.0.1"                          # dirección ip del router
port = "55443"                           # Puerto API REST en router
username = "Cisco"                       # username to login.This debe configurarse con el nivel de privilegio 15.
password = "Cisco"                       # contraseña asociada con nombre de usuario
tokenID = <valor devuelto> # ID de token obtenido del router mediante la función getToken
vrfName = "VRF-A" # nombre del VRF
RD = "3:3" # Distinguidor de ruta para VRF
importRT = "34:34" # Objetivo de ruta de importación
exportRT = "34:34" # export Route Target
interfaceName = "GigabitEthernet3" # nombre de la interfaz de cara al cliente (CE)
interfaceIP = "192.168.13.3" # IP address of CE facing interface
interfaceSubnet = "255.255.255.0" # subnet of CE facing interface
ASN = "34" # número de AS BGP del router PE
neighborIP = "192.168.13.1" # BGP peering IP of CE router
remoteAS = "11" # AS number of CE router
```

En todas las funciones anteriores, se llamaron API dedicadas para cada paso de la configuración. El siguiente ejemplo muestra cómo pasar IOS-XE CLI, en general, en el cuerpo de la llamada API REST. Esto se puede utilizar como una solución alternativa para automatizar si una API en particular no está disponible. En las funciones anteriores, 'content-type' se establece en 'application/json', pero en el ejemplo siguiente, 'content-type' se establece en 'text/plain' mientras analiza la entrada CLI estándar.

Este ejemplo define la descripción de la interfaz GigabitEthernet3. La configuración se puede personalizar cambiando el parámetro "cliInput".

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

```
def passCLIInput (ip, port, tokenID):
solicitudes de importación
 url = "https://" + ip + ":"+ puerto + "/api/v1/global/running-config"
 encabezados = {
   'content-type': "text/plain",
  'X-auth-token': tokenID,
   'cache-control': "no-cache"
   }
 line1 = "Interface GigabitEthernet 3"
 line2 = "description Customer Facing Interface"
 cliInput = línea1 + "\r\n" + línea2
 response = requests.request("PUT", url, encabezados=encabezados, data=cliInput, verify=False )
 print(response.text)
if response.status_code == 204:
   volver "satisfactorio"
de lo contrario:
   volver "fallado"
#---------------------------------
```

Referencias

- Guía de configuración de software del router de servicios en la nube Cisco CSR serie 1000v

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

- Guía de referencia de administración de API REST de Cisco IOS XE

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

Acrónimos utilizados:

MPLS - Conmutación de etiquetas de protocolos múltiples

L3 - Capa 3

VPN - Red privada virtual

VRF - Reenvío de ruta virtual

BGP: protocolo de gateway fronterizo

REST: transferencia de estado representacional

API: interfaz del programa de aplicaciones

JSON: notación de objetos de Java Script

HTTP: protocolo de transferencia de hipertexto

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).