Configure el servicio MPLS L3VPN en el router PE que usa REST-API (IOS-XE)

Contenido

Introducción
prerrequisitos

_
Configuración
Diagrama de la red
Procedimiento de Configuración
1. Extraiga la token-identificación
2. Cree el VRF
3. Trasládese la interfaz a un VRF
4. Asigne la dirección IP para interconectar
5. Cree el BGP enterado VRF
6. Defina al vecino BGP bajo la familia del direccionamiento VRF
Referencias
Siglas usadas:

Introducción

Este documento demuestra el uso de Python que programa para provision un MPLS L3VPN en un router del borde del proveedor de servicio (PE) que usa el RESTO API. Este ejemplo utiliza al Routers de Cisco CSR1000v (IOS-XE) como Routers PE.

Contribuido por: Anuradha Perera

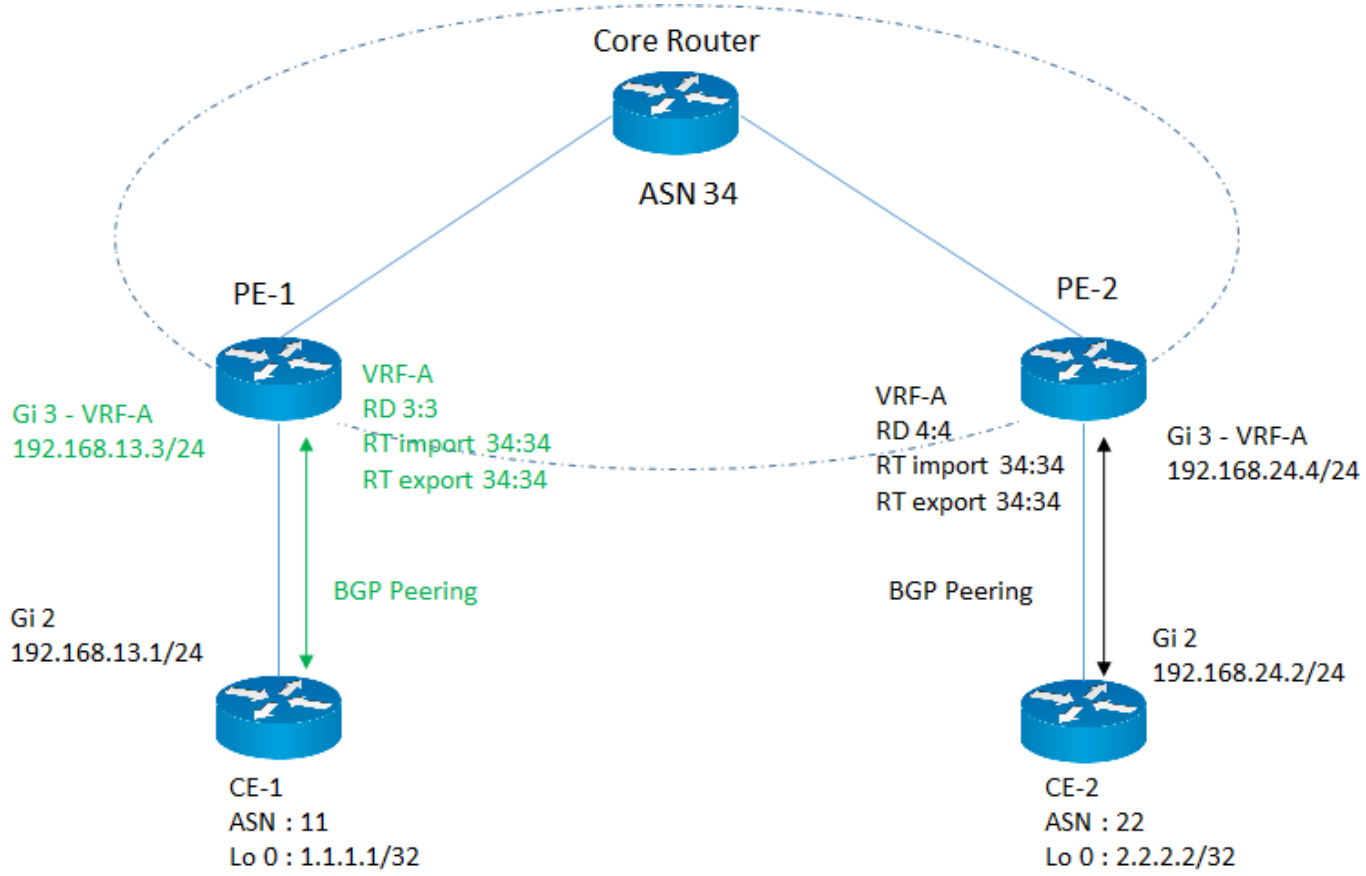
Editado por: Kumar Sridhar

Prerequisites

- Acceso de administración del RESTO API al Routers CSR1000v (los pleae refieren a las referencias en el extremo de este documento).
- Python (versión 2.x o 3.x) y biblioteca de Python de las “peticiones” instalada en el ordenador usado para configurar al Routers.
- Una cierta programación de Python del conocimiento básico.

Configuración

Diagrama de la red



En este foco del ejemplo está en configurar los parámetros de servicio requeridos MPLS L3VPN en el router PE-1, que se resaltan en el color rosado.

Procedimiento de Configuración

La tarea de configuración se divide adentro a varias subtareas y cada subtarea se implementa bajo función definida por el usario. De esta manera las funciones se pueden reutilizar cuando sea necesario.

Todo el uso de las funciones “solicita” la biblioteca para acceder el BASAR API en el router y el formato de datos es JSON. En los pedidos de HTTP “verifique” el parámetro se fija a “falso” para ignorar validar el certificado SSL.

1. Extraiga la token-identificación

Antes de proceder con cualquier configuración en un router usted necesita tener una token-identificación válida obtenida del router. Esta función inicia el pedido de HTTP de autenticar y de obtener una token-identificación de modo que pueda invocar otros API usando este token. La respuesta de esta petición incluye una token-identificación.

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

el def getToken (IP, puerto, nombre de usuario, contraseña):

peticiones de la importación

base64 de la importación

 URL = “https://” + IP +”: “+ puerto el + "/api/v1/auth/token-services"

 encabezados = {

  “tipo de contenido”: “aplicación/json”,

  “autorización”: “Básico” + base64.b64encode((username + “: ” + contraseña) .encode ('UTF-
```

```
8')).decode ("ASCII "),
   "caché-control": "ninguno-caché"
 }
 respuesta = requests.request ("POSTE", URL, headers=headers, verify= falsos)
si == 200 response.status_code:
 vuelva el ["token-id"] response.json ()
:
 vuelva "falló"
#----------------------------------
```

2. Cree el VRF

Esta función creará el VRF en el router PE con el Route Distinguisher requerido (RD) y lo importará/las blancos de la ruta de exportación (el RT)

```
#----------------------------------
createVRF del def (IP, puerto, tokenID, vrfName, RD, importRT, exportRT):
peticiones de la importación
 URL = "https://" + IP +": "+ puerto el + "/api/v1/vrf"
 encabezados = {
   "tipo de contenido": "aplicación/json",
   "X-auth-token": tokenID,
   "caché-control": "ninguno-caché"
 }
 datos = {
   "nombre": vrfName,
   "rd": RD,
   "ruta-blanco": [
      {
```

```
            “acción”: “importación”,
            “comunidad”: importRT
        },
        {
            “acción”: “exportación”,
            “comunidad”: exportRT
        }
    ]
}
 respuesta = requests.request (“POSTE”, URL, headers=headers, json=data, verify= falsos)
si == 201 response.status_code:
   vuelva “acertado”
:
   vuelva “falló”
#---------------------------------
```

3. Trasládese la interfaz a un VRF

Esta función se trasladará una interfaz dada a un VRF.

```
#---------------------------------
addInterfacetoVRF del def (IP, puerto, tokenID, vrfName, interfaceName, RD, importRT,
exportRT):
peticiones de la importación
 URL = “https://” + IP +”: “+ puerto el + "/api/v1/vrf/" + vrfName
 encabezados = {
   “tipo de contenido”: “aplicación/json”,
   “X-auth-token”: tokenID,
   “caché-control”: “ninguno-caché”
```

```
  }
 datos = {
  “rd”: RD,
  “remitiendo”: [interfaceName],
  “ruta-blanco”: [
    {
      “acción”: “importación”,
      “comunidad”: importRT
    },
    {
      “acción”: “exportación”,
      “comunidad”: exportRT
    }
  ]
 }
 respuesta = requests.request (“PUESTO”, URL, headers=headers, json=data, verify= falsos)

si == 204 response.status_code:
  vuelva “acertado”
:
  vuelva “falló”
#----------------------------------
```

4. Asigne la dirección IP para interconectar

Esta función asignará el IP Address a la interfaz.

```
#----------------------------------
assignInterfaceIP del def (IP, puerto, tokenID, interfaceName, interfaceIP, interfaceSubnet):
peticiones de la importación
```

```
 URL = “https://” + IP +”: “+ puerto el + "/api/v1/interfaces/" + interfaceName
 encabezados = {
   “tipo de contenido”: “aplicación/json”,
   “X-auth-token”: tokenID,
   “caché-control”: “ninguno-caché”
   }
 datos = {
   “tipo”: “Ethernetes”,
   “si-nombre”: interfaceName,
   “IP-address": interfaceIP,
   “subnet mask”: interfaceSubnet
   }
 respuesta = requests.request (“PUESTO”, URL, headers=headers, json=data, verify= falsos)
si == 204 response.status_code:
   vuelva “acertado”
:
   vuelva “falló”
#---------------------------------
```

5. Cree el BGP enterado VRF

Esto habilitará a la familia ipv4 del direccionamiento VRF.

```
#---------------------------------
createVrfBGP del def (IP, puerto, tokenID, vrfName, ASN):
peticiones de la importación
 URL = “https://” + IP +”: “+ puerto el + "/api/v1/vrf/" + vrfName + “/routing-svc/bgp”
 encabezados = {
   “tipo de contenido”: “aplicación/json”,
   “X-auth-token”: tokenID,
```

```
  “caché-control”: “ninguno-caché”
  }
 datos = {
  “encaminamiento-protocolo-identificación”: ASN
  }
 respuesta = requests.request (“POSTE”, URL, headers=headers, json=data, verify= falsos)
si == 201 response.status_code:
  vuelva “acertado”
:
  vuelva “falló”
#----------------------------------
```

6. Defina al vecino BGP bajo la familia del direccionamiento VRF

Esta función definirá al vecino BGP bajo el IPV4 de la familia del direccionamiento VRF.

```
#----------------------------------
defineVrfBGPNeighbour del def (IP, puerto, tokenID, vrfName, ASN, neighbourIP, remoteAS):
peticiones de la importación
 URL = “https://” + IP +”: “+ puerto el + "/api/v1/vrf/" + vrfName + “/routing-svc/bgp/ “+ ASN +” /neighbors”
 encabezados = {
  “tipo de contenido”: “aplicación/json”,
  “X-auth-token”: tokenID,
  “caché-control”: “ninguno-caché”
  }
 datos = {
  “encaminamiento-protocolo-identificación”: ASN,
  “direccionamiento”: neighbourIP,
  “telecontrol-como”: remoteAS
```

```
  }
 respuesta = requests.request (“POSTE”, URL, headers=headers, json=data, verify= falsos)
 si == 201 response.status_code:
   vuelva “acertado”
:
   vuelva “falló”
#---------------------------------
```

Descripción y valores de los parámetros de entrada

```
IP el = "10.0.0.1" # IP Address del router
puerto el = "55443" # puerto del RESTO API en el router
nombre de usuario = “Cisco” # nombre de usuario a iniciar sesión. Esto se debe configurar con el nivel de privilegio 15.
contraseña = “Cisco” # contraseña asociada al nombre de usuario
el returned> del tokenID = del <value # el token ID obtenido de usar del router getToken la función
vrfName = “VRF-A” # nombre del VRF
RD el = "3:3" # Route Distinguisher para el VRF
importRT el = "34:34" # blanco de la ruta de la importación
exportRT el = "34:34" # blanco de la ruta de exportación
interfaceName el = "GigabitEthernet3" # nombre de la frontera del cliente (CE) que hace frente a la interfaz
interfaceIP el = "192.168.13.3" # dirección IP del CE que hace frente a la interfaz
interfaceSubnet el = "255.255.255.0" # subred del CE que hace frente a la interfaz
ASN el = "34" # BGP COMO número de router PE
neighbourIP el = "192.168.13.1" # IP de mirada BGP del router CE
remoteAS el = "11" # COMO número de router CE
```

En todas las funciones antedichas, los API dedicados fueron pedidos cada setp de la configuración. El ejemplo abajo demuestra cómo pasar IOS-XE CLI,

generalmente en el cuerpo de la llamada del RESTO API. Esto se puede utilizar como solución alternativa para automatizar si el API determinado no está disponible. En las funciones antedichas “tipo de contenido” se fija a la “aplicación/al json”, pero en el ejemplo abajo, fijan al “tipo de contenido” “para mandar un SMS/llano” mientras que está analizando la entrada estándar CLI.

Este ejemplo define la descripción de la interfaz para la interfaz GigabitEthernet3. La configuración puede ser personalizada cambiando el parámetro del “cliInput”.

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

passCLIInput del def (IP, puerto, tokenID):

peticiones de la importación

 URL = “https://” + IP +”: “+ puerto el + "/api/v1/global/running-config"

 encabezados = {

  “tipo de contenido”: “texto/llano”,

 “X-auth-token”: tokenID,

  “caché-control”: “ninguno-caché”

  }

 line1 = “gigabitethernet el 3" de la interfaz

 line2 = “cliente de la descripción que hace frente a la interfaz”

 cliInput = line1 + “\ r \ n” + line2

 respuesta = requests.request (“PUESTO”, URL, headers=headers, data=cliInput, verify= falsos)

 print(response.text)

si == 204 response.status_code:

  vuelva “acertado”

:

  vuelva “falló”

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

Referencias

- La nube de la serie de Cisco CSR 1000v mantiene la guía de configuración del software del 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

- Guía de referencia de administración del RESTO API del Cisco IOS XE

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

Siglas usadas:

MPLS - Multi Protocol Label Switching

L3 - Capa 3

VPN - Virtual Private Network

VRF - Expedición de la ruta virtual

BGP - Protocolo Protocolo de la puerta de enlace marginal (BGP)

RESTO - Transferencia representativa del estado

API - Interfaz de programación de aplicaciones

JSON - Notación del objeto de la secuencia de comandos Java

HTTP - Protocolo hyper text transfer