Configurar o serviço MPLS L3VPN no roteador de PE usando REST-API (IOS-XE)

Índice

Introdução

Este documento demonstra o uso do pitão que programa para provision um MPLS L3VPN em um roteador da borda do provedor de serviços (PE) que usa o RESTO API. Este exemplo usa o Roteadores de Cisco CSR1000v (IOS-XE) como roteadores de PE.

Contribuído por: Anuradha Perera

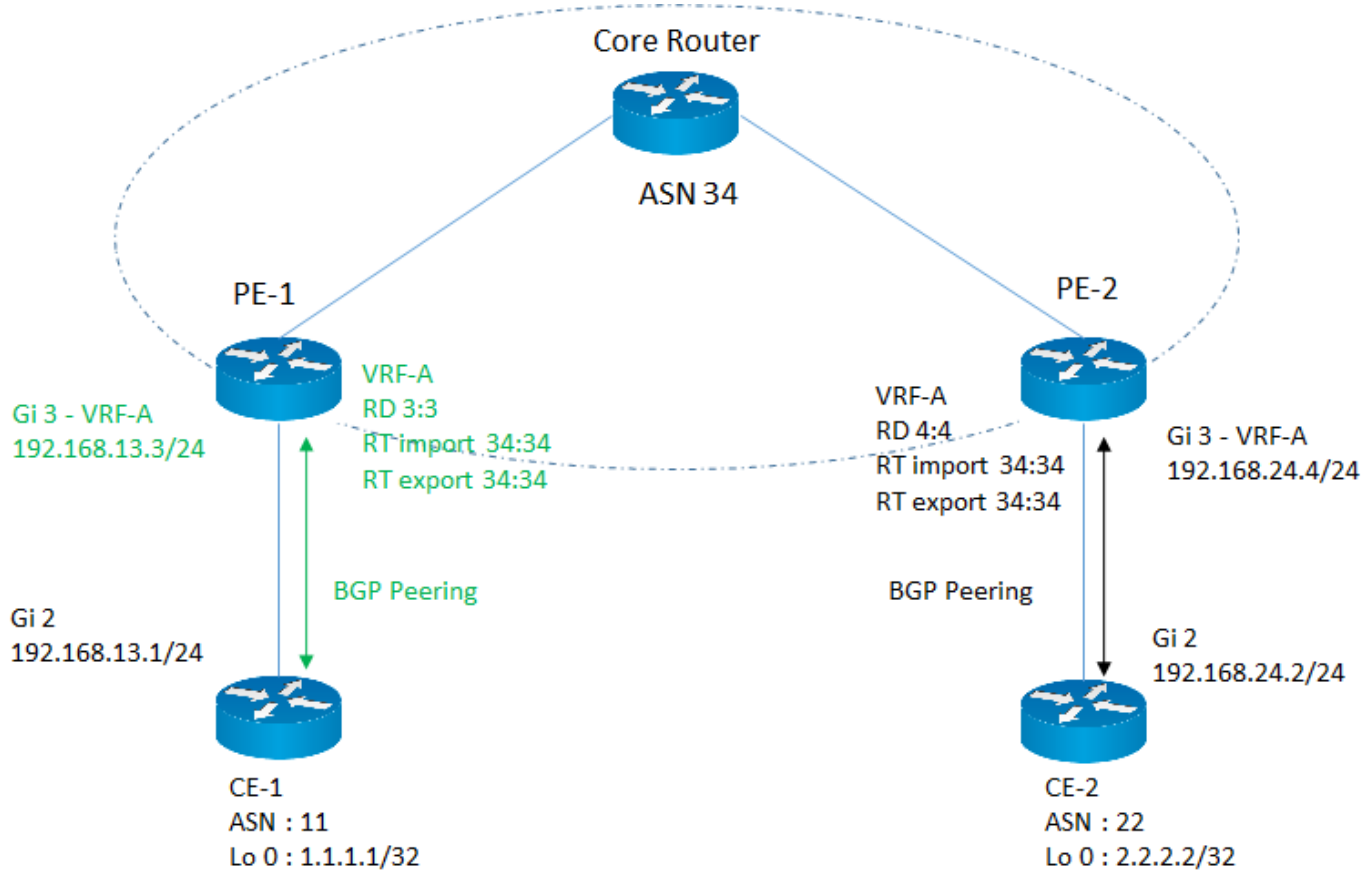
Editado por: Kumar Sridhar

Pré-requisitos

- Acesso de gerenciamento do RESTO API ao Roteadores CSR1000v (os pleae referem as referências na extremidade deste documento).
- Biblioteca do pitão (versão 2.x ou 3.x) e do pitão dos “pedidos” instalada no computador usado configurando o Roteadores.
- Alguma programação do pitão do conhecimento básico.

Configuração

Diagrama de Rede



Neste foco do exemplo está em configurar os parâmetros de serviço exigidos MPLS L3VPN no roteador PE-1, que são destacados na cor cor-de-rosa.

Procedimento de configuração

As tarefas de configuração são divididas dentro a um número de secundário-tarefas e cada secundário-tarefa é executada sob uma função definida pelo utilizador. Desta maneira as funções podem ser reutilizadas se necessário.

Todo o uso das funções “pede” a biblioteca para alcançar o RESTO API no roteador e o formato de dados é JSON. Nos pedidos do HTTP “verifique que” o parâmetro está ajustado a “falso” para ignorar a validação do certificado SSL.

1. Recupere a token-identificação

Antes de continuar com toda a configuração em um roteador você precisa de ter uma token-identificação válida obtida do roteador. Esta função inicia o pedido do HTTP autenticar e obter uma token-identificação de modo que possa invocar outros API usando este token. A resposta deste pedido inclui uma token-identificação.

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

o def getToken (IP, porta, username, senha):

pedidos da importação

importação base64

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

 encabeçamentos = {

  “tipo de conteúdo”: “aplicativo/json”,

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

```
8')).decode (“ascii '),
  “esconderijo-controle”: “nenhum-esconderijo”
 }
 resposta = requests.request (“CARGO”, URL, headers=headers, verify= falsos)
se == 200 response.status_code:
 retorne o ["token-id"] response.json ()
mais:
 retorne “falhou”
#----------------------------------
```

2. Crie o VRF

Esta função criará o VRF no roteador de PE com o distinguidor de rota exigido (RD) e importá-lo-á/alvos da rota de exportação (o RT)

```
#----------------------------------
createVRF do def (IP, porta, tokenID, vrfName, RD, importRT, exportRT):
pedidos da importação
 URL = “https://” + IP +”: “+ porta + "/api/v1/vrf"
 encabeçamentos = {
  “tipo de conteúdo”: “aplicativo/json”,
  “X-AUTH-token”: tokenID,
  “esconderijo-controle”: “nenhum-esconderijo”
 }
 dados = {
  “nome”: vrfName,
  “rd”: RD,
  “rota-alvo”: [
    {
```

```
            “ação”: “importação”,
            a “comunidade”: importRT
        },
        {
            “ação”: “exportação”,
            a “comunidade”: exportRT
        }
    ]
}
 resposta = requests.request (“CARGO”, URL, headers=headers, json=data, verify= falsos)
se == 201 response.status_code:
   retorne “bem sucedido”
mais:
   retorne “falhou”
#----------------------------------
```

3. Mova a relação em um VRF

Esta função moverá uma dada interface em um VRF.

```
#----------------------------------
addInterfacetoVRF do def (IP, porta, tokenID, vrfName, interfaceName, RD, importRT, exportRT):
pedidos da importação
 URL = “https://” + IP +”: “+ porta + "/api/v1/vrf/" + vrfName
 encabeçamentos = {
   “tipo de conteúdo”: “aplicativo/json”,
   “X-AUTH-token”: tokenID,
   “esconderijo-controle”: “nenhum-esconderijo”
   }
```

```
 dados = {
   “rd”: RD,
   “enviando”: [interfaceName],
   “rota-alvo”: [
      {
         “ação”: “importação”,
         a “comunidade”: importRT
      },
      {
         “ação”: “exportação”,
         a “comunidade”: exportRT
      }
   ]
 }
 resposta = requests.request (“POSTO”, URL, headers=headers, json=data, verify= falsos)


se == 204 response.status_code:
   retorne “bem sucedido”
mais:
   retorne “falhou”
#---------------------------------
```

4. Atribua o endereço IP de Um ou Mais Servidores Cisco ICM NT para conectar

Esta função atribuirá o endereço IP de Um ou Mais Servidores Cisco ICM NT à relação.

```
#---------------------------------
assignInterfaceIP do def (IP, porta, tokenID, interfaceName, interfaceIP, interfaceSubnet):
pedidos da importação
 URL = “https://” + IP +”: “+ porta + "/api/v1/interfaces/" + interfaceName
```

```
 encabeçamentos = {
   “tipo de conteúdo”: “aplicativo/json”,
   “X-AUTH-token”: tokenID,
   “esconderijo-controle”: “nenhum-esconderijo”
   }
 dados = {
   “tipo”: “Ethernet”,
   “se-nome”: interfaceName,
   “IP-address": interfaceIP,
   “subnet mask”: interfaceSubnet
   }
 resposta = requests.request (“POSTO”, URL, headers=headers, json=data, verify= falsos)
se == 204 response.status_code:
   retorne “bem sucedido”
mais:
   retorne “falhou”
#---------------------------------
```

5. Crie o BGP ciente VRF

Isto permitirá o IPv4 da família do endereço VRF.

```
#---------------------------------
createVrfBGP do def (IP, porta, tokenID, vrfName, ASN):
pedidos da importação
 URL = “https://” + IP +”: “+ porta + "/api/v1/vrf/" + vrfName + “/routing-svc/bgp”
 encabeçamentos = {
   “tipo de conteúdo”: “aplicativo/json”,
   “X-AUTH-token”: tokenID,
   “esconderijo-controle”: “nenhum-esconderijo”
```

```
  }
 dados = {
  “roteamento-protocolo-identificação”: ASN
  }
 resposta = requests.request (“CARGO”, URL, headers=headers, json=data, verify= falsos)
se == 201 response.status_code:
  retorne “bem sucedido”
mais:
  retorne “falhou”
#---------------------------------
```

6. Defina o vizinho de BGP sob a família do endereço VRF

Esta função definirá o vizinho de BGP sob a família IPV4 do endereço VRF.

```
#---------------------------------
defineVrfBGPNeighbour do def (IP, porta, tokenID, vrfName, ASN, neighbourIP, remoteAS):
pedidos da importação
 URL = “https://” + IP +”: “+ porta + "/api/v1/vrf/" + vrfName + “/routing-svc/bgp/ “+ ASN +” /neighbors”
 encabeçamentos = {
  “tipo de conteúdo”: “aplicativo/json”,
  “X-AUTH-token”: tokenID,
  “esconderijo-controle”: “nenhum-esconderijo”
  }
 dados = {
  “roteamento-protocolo-identificação”: ASN,
  “endereços: neighbourIP,
  “remoto-como”: remoteAS
  }
```

```
 resposta = requests.request (“CARGO”, URL, headers=headers, json=data, verify= falsos)

 se == 201 response.status_code:

   retorne “bem sucedido”

mais:

   retorne “falhou”

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

Descrição e valores dos parâmetros de entrada

```
IP = "10.0.0.1" # endereço IP de Um ou Mais Servidores Cisco ICM NT do roteador

porta = "55443" # porta do RESTO API no roteador

username = “Cisco” # username a entrar. Isto deve ser configurado com nível de privilégio 15.

senha = “Cisco” # senha associada com o username

o returned> do tokenID = do <value # o token ID obtido da utilização do roteador getToken a função

vrfName = “VRF-A” # nome do VRF

RD = "3:3" # distinguidor de rota para o VRF

importRT = "34:34" # alvo da rota da importação

exportRT = "34:34" # alvo da rota de exportação

interfaceName = "GigabitEthernet3" # nome do edge de cliente (CE) que enfrenta a relação

interfaceIP = "192.168.13.3" # endereço IP de Um ou Mais Servidores Cisco ICM NT do CE que enfrenta a relação

interfaceSubnet = "255.255.255.0" # sub-rede do CE que enfrenta a relação

ASN = "34" # BGP COMO o número de roteador de PE

neighbourIP = "192.168.13.1" # IP espreitando BGP do CE Router

remoteAS = "11" # COMO o número de CE Router
```

Em todas as funções acima, os API dedicados foram chamados para cada setp da configuração. O exemplo abaixo demonstra como passar IOS-XE CLI, geralmente, no corpo do atendimento do RESTO API. Isto pode ser usado como uma ação alternativa para automatizar se o API particular não está disponível. Nas funções acima “tipo de conteúdo” é ajustado ao “aplicativo/json”, mas no exemplo abaixo, o “tipo de conteúdo” é ajustado “para text/liso” enquanto está analisando gramaticalmente a entrada padrão CLI.

Este exemplo define a descrição da relação para a relação GigabitEthernet3. A configuração pode ser personalizada mudando o parâmetro do “cliInput”.

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

passCLIInput do def (IP, porta, tokenID):

pedidos da importação

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

 encabeçamentos = {

  “tipo de conteúdo”: “texto/liso”,

 “X-AUTH-token”: tokenID,

  “esconderijo-controle”: “nenhum-esconderijo”

  }

 line1 = da “gigabitethernet 3" relação

 line2 = da “cliente descrição que enfrenta a relação”

 cliInput = line1 + “\ r \ n” + line2

 resposta = requests.request (“POSTO”, URL, headers=headers, data=cliInput, verify= falsos)

 print(response.text)

se == 204 response.status_code:

  retorne “bem sucedido”

mais:

  retorne “falhou”

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

Referências

- A nuvem da série de Cisco CSR 1000v presta serviços de manutenção ao manual de configuração do software do roteador

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

- Guia de referência de gerenciamento do RESTO API do Cisco IOS XE

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

Acrônimos usados:

MPLS - Multi Protocol Label Switching

L3 - Camada 3

VPN - Virtual Private Network

VRF - Transmissão da rota virtual

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

RESTO - Transferência representacional do estado

API - Application Program Interface

JSON - Notação do objeto do script de Java

HTTP - Protocolo hyper text transfer