Configurez le service MPLS L3VPN sur le routeur PE utilisant REST-API (IOS-XE)

Contenu

Introduction

Ce document explique l'utilisation du python programmant pour provision un MPLS L3VPN sur un routeur de périphérie de fournisseur de services (PE) utilisant le REPOS API. Cet exemple utilise des Routeurs de Cisco CSR1000v (IOS-XE) comme Routeurs de PE.

Contribué par : Anuradha Perera

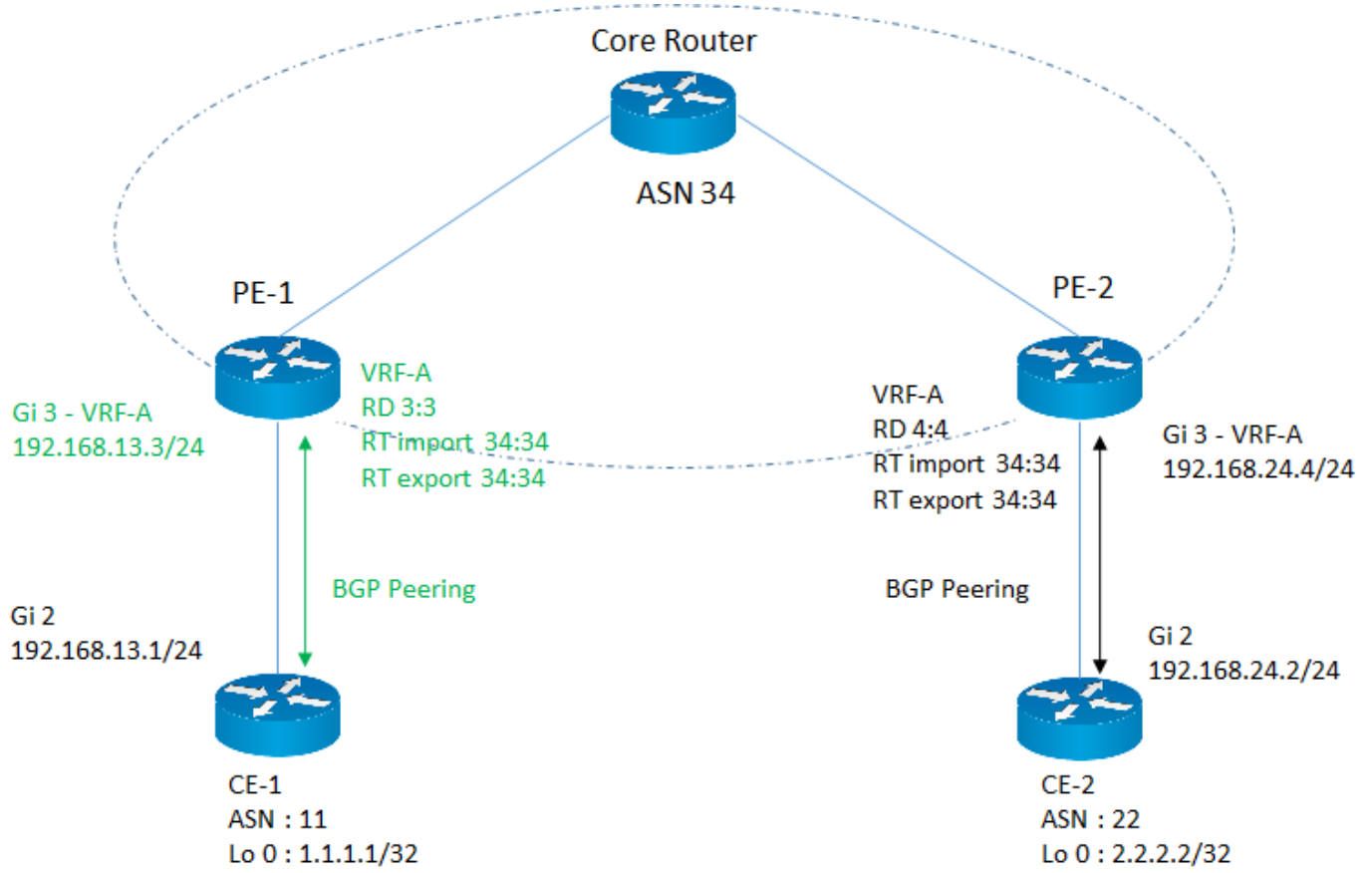
Édité par : Kumar Sridhar

Conditions préalables

- Accès de Gestion du REPOS API aux Routeurs CSR1000v (les pleae se rapportent aux références à la fin de ce document).
- Bibliothèque de python (version 2.x ou 3.x) et de python de « demandes » installée sur l'ordinateur utilisé pour configurer les Routeurs.
- La programmation de python de connaissance de base.

Configuration

Diagramme du réseau



Dans cet exemple le foyer est sur configurer les paramètres de service exigés MPLS L3VPN sur le routeur PE-1, qui sont mis en valeur dans la couleur rose.

Procédure de configuration

La tâche de configuration est divisée dedans à un certain nombre de sous tâches et chaque sous tâche est mise en application sous une fonction définie par l'utilisateur. De cette façon des fonctions peuvent être réutilisées en cas de besoin.

Toute l'utilisation de fonctions « invite » la bibliothèque pour accéder au REPOS API sur le routeur et le format des données est JSON. Dans des demandes de HTTP « vérifiez » le paramètre est placé à « faux » pour ignorer valider le certificat ssl.

1. Récupérez le jeton-id

Avant de commencer avec n'importe quelle configuration sur un routeur vous devez avoir un jeton-id valide obtenu du routeur. Cette fonction initie la demande de HTTP d'authentifier et obtenir un jeton-id de sorte qu'elle puisse appeler d'autres API utilisant ce jeton. La réponse de cette demande inclut un jeton-id.

```
#~#----------------------------------

le def getToken (IP, port, nom d'utilisateur, mot de passe) :

demandes d'importation

importation base64

 URL = « https:// » + IP + » : « + port + "/api/v1/auth/token-services"

 en-têtes = {

   « type de contenu » : « application/json »,

   « autorisation » : « De base » + base64.b64encode((username + « :  » + mot de passe) .encode
```

```
('UTF-8')).decode (« ASCII '),
   « contrôle du cache » : « NO--cache »
 }
 réponse = requests.request (« POST », URL, headers=headers, verify= faux)
si == 200 response.status_code :
 renvoyez response.json () [le « jeton-id »]
autrement :
 retournez « manqué »
#~#----------------------------------
```

2. Créez le VRF

Cette fonction créera le VRF sur le routeur PE avec le moteur de distinction de route exigé (RD) et les cibles d'artère d'importation/exportation (la droite)

```
#~#----------------------------------
createVRF de def (IP, port, tokenID, vrfname, RD, importRT, exportRT) :
demandes d'importation
 URL = « https:// » + IP + » : « + port + "/api/v1/vrf"
 en-têtes = {
   « type de contenu » : « application/json »,
   « X-auth-jeton » : tokenID,
   « contrôle du cache » : « NO--cache »
 }
 données = {
   « nom » : vrfname,
   « rd » : RD,
   « route-target » : [
      {
```

```
        « action » : « importation »,
        la « communauté » : importRT
    },
    {
        « action » : « exportation »,
        la « communauté » : exportRT
    }
  ]
}
 réponse = requests.request (« POST », URL, headers=headers, json=data, verify= faux)
si == 201 response.status_code :
  renvoyez « réussi »
autrement :
  retournez « manqué »
#~#----------------------------------
```

3. Entrez l'interface dans un VRF

Cette fonction entrera une interface donnée dans un VRF.

```
#~#----------------------------------
addInterfacetoVRF de def (IP, port, tokenID, vrfname, interfaceName, RD, importRT, exportRT) :
demandes d'importation
 URL = « https:// » + IP + » : « + port + "/api/v1/vrf/" + vrfname
 en-têtes = {
  « type de contenu » : « application/json »,
  « X-auth-jeton » : tokenID,
  « contrôle du cache » : « NO--cache »
  }
```

```
 données = {
   « rd » : RD,
   « expédiant » : [interfaceName],
   « route-target » : [
      {
         « action » : « importation »,
         la « communauté » : importRT
      },
      {
         « action » : « exportation »,
         la « communauté » : exportRT
      }
   ]
}
 réponse = requests.request (« MIS », URL, headers=headers, json=data, verify= faux)


si == 204 response.status_code :
   renvoyez « réussi »
autrement :
   retournez « manqué »
#~#----------------------------------
```

4. Assignez l'adresse IP pour relier

Cette fonction assignera l'IP address à l'interface.

```
#~#----------------------------------
assignInterfaceIP de def (IP, port, tokenID, interfaceName, interfaceIP, interfaceSubnet) :
demandes d'importation
 URL = « https:// » + IP + » : « + port + "/api/v1/interfaces/" + interfaceName
```

```
 en-têtes = {
   « type de contenu » : « application/json »,
   « X-auth-jeton » : tokenID,
   « contrôle du cache » : « NO--cache »
   }
 données = {
   « type » : « Ethernets »,
   « si-nom » : interfaceName,
   « IP address » : interfaceIP,
   « subnet mask » : interfaceSubnet
   }
 réponse = requests.request (« MIS », URL, headers=headers, json=data, verify= faux)
si == 204 response.status_code :
   renvoyez « réussi »
autrement :
   retournez « manqué »
#~#----------------------------------
```

5. Créez le BGP averti de VRF

Ceci activera l'ipv4 de famille d'adresse de VRF.

```
#~#----------------------------------
createVrfBGP de def (IP, port, tokenID, vrfname, ASN) :
demandes d'importation
 URL = « https:// » + IP + » : « + port + "/api/v1/vrf/" + vrfname + « /routing-svc/bgp »
 en-têtes = {
   « type de contenu » : « application/json »,
   « X-auth-jeton » : tokenID,
   « contrôle du cache » : « NO--cache »
```

```
  }
 données = {
  « routage-Protocol-id » : ASN
  }
 réponse = requests.request (« POST », URL, headers=headers, json=data, verify= faux)
si == 201 response.status_code :
  renvoyez « réussi »
autrement :
  retournez « manqué »
#~#----------------------------------
```

6. Définissez le voisin BGP sous la famille d'adresse de VRF

Cette fonction définira le voisin BGP sous l'IPV4 de famille d'adresse de VRF.

```
#~#----------------------------------
defineVrfBGPNeighbour de def (IP, port, tokenID, vrfname, ASN, neighbourIP, remoteAS) :
demandes d'importation
 URL = « https:// » + IP + » : « + port + "/api/v1/vrf/" + vrfname + « /routing-svc/bgp/ « + ASN + » /neighbors »
 en-têtes = {
  « type de contenu » : « application/json »,
  « X-auth-jeton » : tokenID,
  « contrôle du cache » : « NO--cache »
  }
 données = {
  « routage-Protocol-id » : ASN,
  « adresse » : neighbourIP,
  « remote-as » : remoteAS
  }
```

```
 réponse = requests.request (« POST », URL, headers=headers, json=data, verify= faux)

 si == 201 response.status_code :

   renvoyez « réussi »

autrement :

   retournez « manqué »

#~#----------------------------------
```

Description et valeurs des paramètres d'entrée

```
IP = "10.0.0.1" # IP address du routeur

port = "55443" # port du REPOS API sur le routeur

nom d'utilisateur = « Cisco » # nom d'utilisateur à ouvrir une session. Ceci devrait être configuré avec le niveau de privilège 15.

le mot de passe = « Cisco » # mot de passe a associé avec le nom d'utilisateur

le returned> de tokenID = de <value # l'ID symbolique a obtenu du routeur que l'utilisation getToken la fonction

vrfname = « VRF-A » # nom du VRF

RD = "3:3" # moteur de distinction de route pour le VRF

importRT = "34:34" # cible d'artère d'importation

exportRT = "34:34" # cible d'artère d'exportation

interfaceName = "GigabitEthernet3" # nom du Customer Edge (CE) faisant face à l'interface

interfaceIP = "192.168.13.3" # adresse IP de CE faisant face à l'interface

interfaceSubnet = "255.255.255.0" # sous-réseau de CE faisant face à l'interface

ASN = "34" # numéro de système autonome BGP de routeur PE

neighbourIP = "192.168.13.1" # IP scrutant BGP de routeur CE

remoteAS = "11" # numéro de système autonome de routeur CE
```

Dans toutes les fonctions ci-dessus, des API dédiés ont été nécessités chaque setp de configuration. L'exemple ci-dessous explique comment passer IOS-XE CLI, généralement dans le corps de l'appel du REPOS API. Ceci peut être utilisé comme contournement pour automatiser si l'API particulier n'est pas disponible. Dans les fonctions ci-dessus « type de contenu » est placé à la « application/au json », mais dans l'exemple ci-dessous, le « type de contenu » est « de textoter réglé/brute » car il analyse l'entrée standard CLI.

Cet exemple définit la description d'interface pour l'interface GigabitEthernet3. La configuration peut être personnalisée en changeant le paramètre de « cliInput ».

```
#~#---------------------------------

passCLIInput de def (IP, port, tokenID) :

demandes d'importation

 URL = « https:// » + IP + » : « + port + "/api/v1/global/running-config"

 en-têtes = {

   « type de contenu » : « texte/brute »,

  « X-auth-jeton » : tokenID,

   « contrôle du cache » : « NO--cache »

   }

 line1 = « interfaces GigabitEthernet 3"

 line2 = « client de description faisant face à l'interface »

 cliInput = line1 + « \ r \ n » + line2

 réponse = requests.request (« MIS », URL, headers=headers, data=cliInput, verify= faux)

 print(response.text)

si == 204 response.status_code :

   renvoyez « réussi »

autrement :

   retournez « manqué »

#~#---------------------------------
```

Références

- Guide de configuration de logiciel du router de services en nuage de gamme CSR 1000v de Cisco

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

- Guide de référence de Gestion du REPOS API de Cisco IOS XE

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

Acronymes utilisés :

MPLS - Commutation par étiquette multi de Protocol

L3 - Couche 3

VPN - Réseau privé virtuel

VRF - Expédition de route virtuelle

BGP - Protocole BGP

REPOS - Transfert figurative d'état

API - Interface de programmation

JSON - Notation d'objet de JAVASCRIPT

HTTP - Texte hyper Transfer Protocol