Automatisation des commutateurs Catalyst 9000 à l'aide de scripts Python

Table des matières

Introduction

Ce document décrit comment étendre EEM avec des scripts Python pour automatiser la configuration et la collecte de données sur les commutateurs Catalyst 9000.

Conditions préalables

Exigences

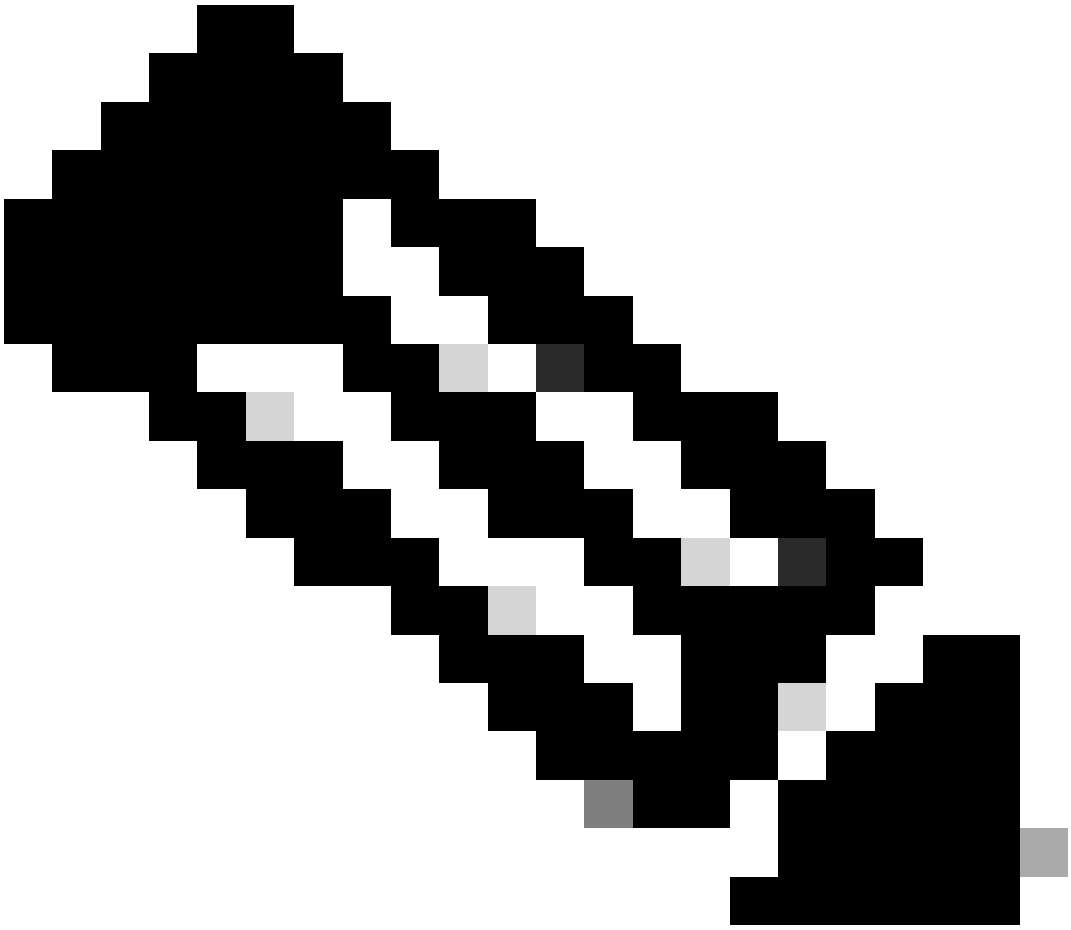
Cisco vous recommande de connaître et de vous familiariser avec les sujets suivants :

- Cisco IOS® et Cisco IOS® XE EEM
- Hébergement d'applications et shell invité
- Script Python
- Commandes Linux

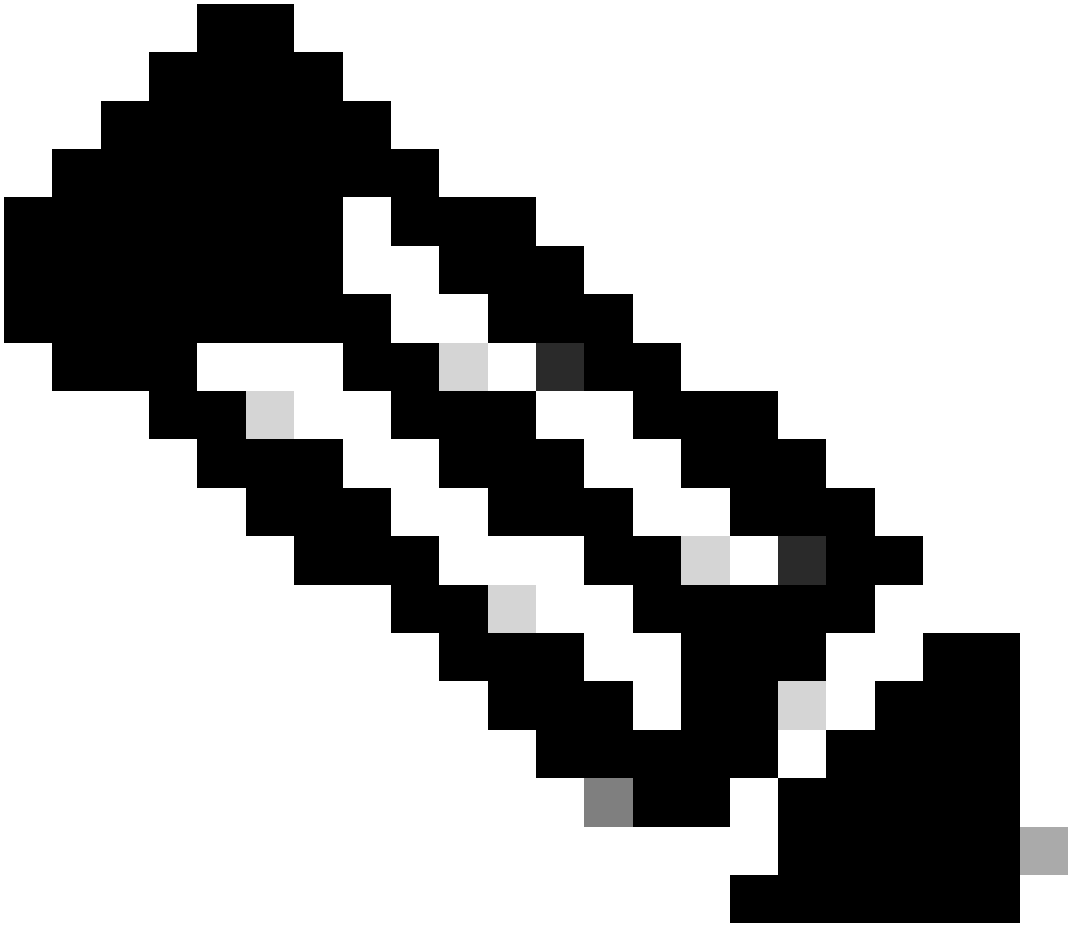
Composants utilisés

Les informations contenues dans ce document sont basées sur les versions de matériel et de logiciel suivantes :

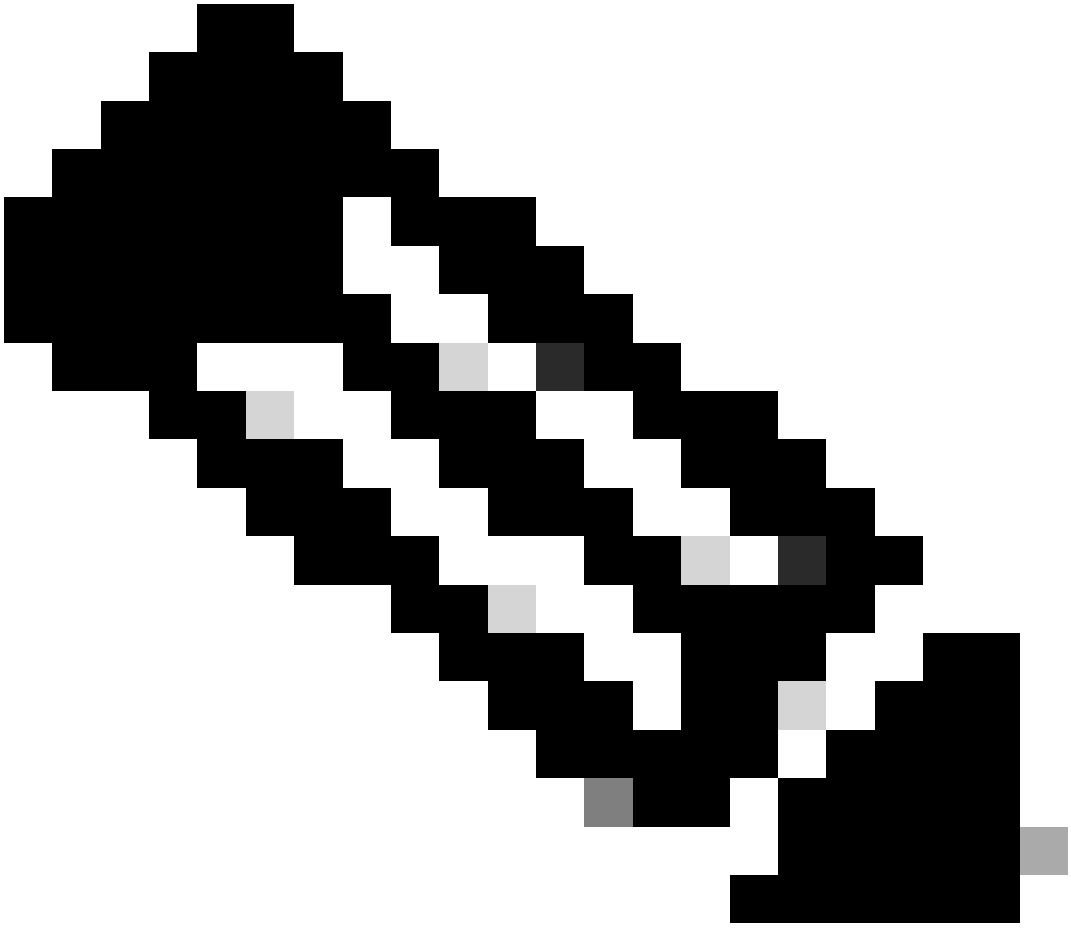
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1 et versions ultérieures



> Remarque : Consultez le guide de configuration approprié pour connaître les commandes utilisées afin d'activer ces fonctionnalités sur d'autres plateformes Cisco.



Remarque : Les commutateurs Catalyst 9200L ne prennent pas en charge Guest Shell.



Remarque : Ces scripts ne sont pas pris en charge par le centre d'assistance technique Cisco et sont fournis tels quels à des fins pédagogiques.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. Si votre réseau est en ligne, assurez-vous de bien comprendre l’incidence possible des commandes.

Conventions

Pour plus d'informations sur les conventions utilisées dans ce document, reportez-vous à Conventions relatives aux conseils techniques Cisco.

Informations générales

Shell invité et scripts Python

L'hébergement d'applications sur la gamme de commutateurs Cisco Catalyst 9000 présente des opportunités d'innovation pour les partenaires et les développeurs, car les périphériques réseau peuvent être fusionnés avec un environnement d'exécution d'applications.

Il prend en charge les applications conteneurisées, ce qui permet une isolation complète du système d'exploitation principal et du noyau Cisco IOS XE. Cette séparation garantit que les allocations de ressources pour les applications hébergées sont distinctes des fonctions principales de routage et de commutation.

L'infrastructure d'hébergement d'applications pour les périphériques Cisco IOS XE est appelée IOx (Cisco IOS + Linux), ce qui facilite l'hébergement d'applications et de services développés par Cisco, ses partenaires et des développeurs tiers sur des périphériques réseau, assurant une intégration transparente sur diverses plates-formes matérielles.

Guest Shell, un déploiement de conteneur spécialisé, illustre une application utile pour le déploiement du système.

Guest Shell propose un environnement Linux virtualisé conçu pour exécuter des applications Linux personnalisées, y compris Python, afin de permettre le contrôle et la gestion automatisés des périphériques Cisco. Le conteneur Guest Shell permet aux utilisateurs d'exécuter des scripts et des applications dans le système. Plus précisément, sur les plates-formes Intel x86, le conteneur Guest Shell est un conteneur Linux (LXC) avec un système de fichiers racine minimal CentOS 8.0. Dans Cisco IOS XE Amsterdam 17.3.1 et versions ultérieures, seul Python V3.6 est pris en charge. Des bibliothèques Python supplémentaires peuvent être installées pendant l'exécution à l'aide de l'utilitaire Yum dans CentOS 8.0. Les paquets Python peuvent également être installés ou mis à jour à l'aide de PIP (Pip Install Packages).

Guest Shell inclut une API (Application Programming Interface) Python, qui permet d'exécuter les commandes Cisco IOS XE à l'aide du module ILC Python. De cette façon, les scripts Python améliorent les capacités d'automatisation, fournissant aux ingénieurs réseau un outil polyvalent pour développer des scripts pour automatiser les tâches de configuration et de collecte de données. Bien que ces scripts puissent être exécutés manuellement via l'interface de ligne de commande, ils peuvent également être utilisés avec des scripts EEM pour répondre à des événements spécifiques, tels que des messages Syslog, des événements d'interface ou des exécutions de commandes. Pratiquement, tout événement pouvant déclencher un script EEM peut également être utilisé pour déclencher un script Python, ce qui augmente le potentiel d'automatisation des commutateurs Cisco Catalyst 9000.

Lorsque Guest Shell est installé, un répertoire de partage d'invité est automatiquement créé dans le système de fichiers flash. Il s'agit du système de fichiers accessible à partir des scripts Python et de Guest Shell. Pour garantir une synchronisation correcte lors de l'utilisation de l'empilage, conservez ce dossier sous 50 Mo.

Avantages de l'utilisation de Python avec des scripts EEM

- Python étend les capacités d'automatisation des scripts EEM en permettant à la logique complexe (comme les expressions régulières, les boucles et les correspondances) d'être gérée dans le script Python. Cette fonctionnalité permet de créer des scripts EEM plus

puissants.

- En tant que langage de programmation bien connu, Python abaisse la barrière d'entrée pour les ingénieurs réseau qui souhaitent automatiser les périphériques Cisco IOS XE. En outre, il offre la facilité de maintenance et la lisibilité.
- Python fournit également des fonctionnalités de gestion des erreurs ainsi qu'une bibliothèque standard puissante.

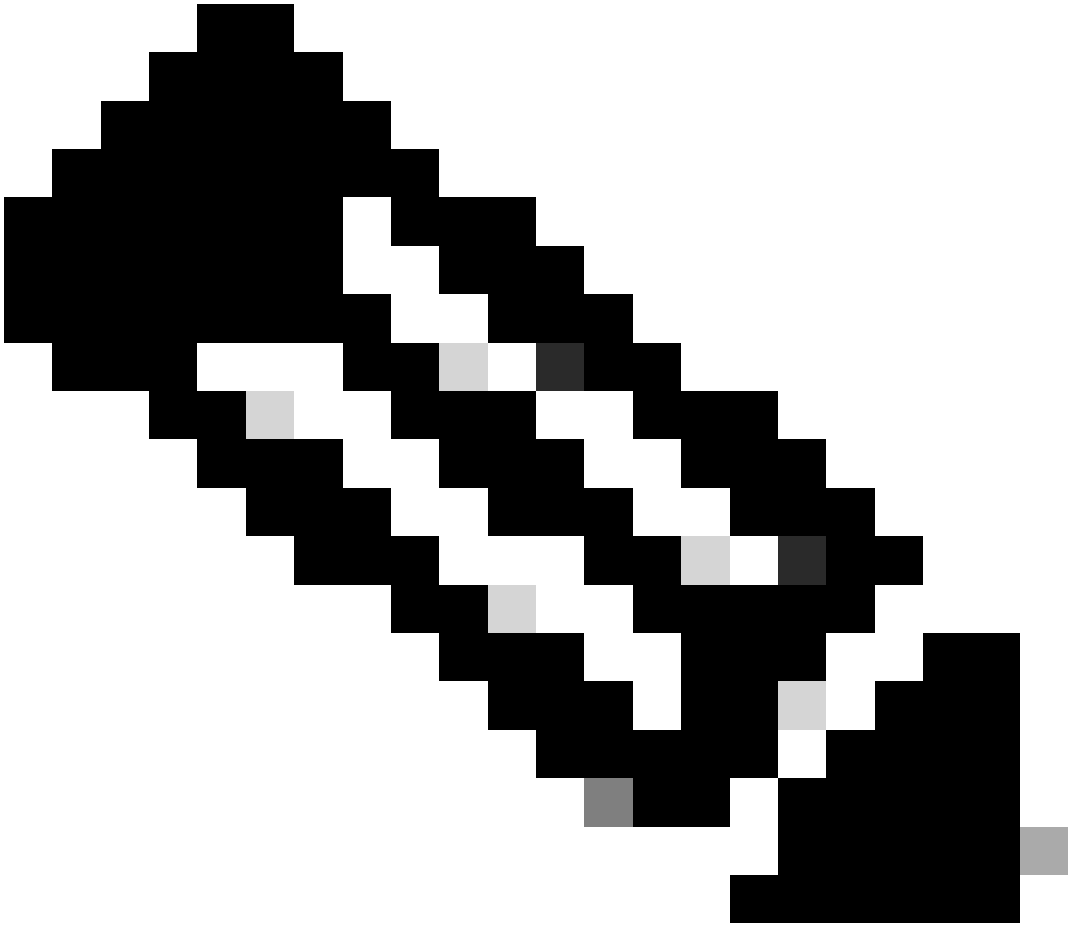
Considérations sur l'utilisation de Python avec des scripts EEM

- Guest Shell n'est pas activé par défaut, il doit donc être activé avant que les scripts Python puissent être exécutés.
- Les scripts Python ne peuvent pas être créés directement dans la CLI ; ils doivent d'abord être développés dans un environnement de développement, puis copiés dans la mémoire flash du commutateur.

SELinux dans Cisco IOS XE

Depuis Cisco IOS XE 17.8.1, la prise en charge de Security-Enhanced Linux (SELinux) a été introduite pour appliquer la sécurité avec des politiques qui régissent la manière dont les processus, les utilisateurs et les fichiers interagissent les uns avec les autres. La stratégie SELinux définit les actions et les ressources auxquelles un processus ou un utilisateur peut accéder. Une violation peut se produire lorsqu'un utilisateur ou un processus tente d'effectuer une action non autorisée par la stratégie, par exemple, l'accès à une ressource ou l'exécution d'une commande. SELinux peut fonctionner dans 2 modes différents :

1. Mode permissif : SELinux n'applique aucune stratégie. Cependant, il enregistre toujours les violations comme si elles avaient été refusées.
2. Application : SELinux applique activement les politiques de sécurité sur le système. Si une action enfreint la stratégie SELinux, elle est refusée et consignée.



Remarque : Le mode par défaut a été défini sur permissive lors de son introduction dans Cisco IOS XE 17.8.1. Cependant, à partir de la version 17.14.1, SELinux est activé en mode application.

Lors de l'utilisation de Guest Shell, l'accès à certaines ressources peut être refusé lors de l'utilisation du mode application. Si, lors d'une tentative d'exécution d'une action à l'aide de Guest Shell ou d'un script Python, une erreur d'autorisation est refusée, semblable à ce journal :

```
*Jan 21 13:22:01: %SELINUX-1-VIOLATION: Chassis 1 R0/0: audispd: type=AVC msg=audit(1738074795.448:198)
```

Pour vérifier si un script est refusé par SELinux, utilisez la commande`show platform software audit summary`pour vérifier si les nombres de refus augmentent. En outre,`show platform software audit all`affiche un journal des actions bloquées par SELinux. Access Vector Cache (AVC) est le mécanisme utilisé pour enregistrer les décisions de contrôle d'accès dans SELinux. Par conséquent, lorsque vous

utilisez cette commande, recherchez les journaux qui commencent par type=AVC.

Si un script est refusé et bloqué, SELinux peut être défini en mode permissif à l'aide de la commande`set platform software selinux permissive`. Cette modification n'est pas stockée dans la configuration en cours ou de démarrage. Par conséquent, après un rechargement, le mode revient à l'application. Par conséquent, chaque fois que le commutateur se recharge, cette modification doit être réappliquée. La modification peut être vérifiée à l'aide de`show platform software selinux`.

Configurer

Pour automatiser les tâches sur le commutateur à l'aide de scripts EEM et Python, procédez comme suit :

1. Activez l'interpréteur de commandes invité.
2. Copiez le script Python dans le répertoire /flash/guest-share/. Vous pouvez utiliser n'importe quel mécanisme de copie disponible dans Cisco IOS XE, tel que SCP, FTP ou le gestionnaire de fichiers dans l'interface utilisateur Web. Une fois que le script Python est dans la mémoire flash, vous pouvez l'exécuter en utilisant la commande`guestshell run python3 /flash/guest-share/cat9k_script.py`.
3. Configurez un script EEM qui exécute le script Python. Cette configuration permet de déclencher le script Python à l'aide de l'un des multiples détecteurs d'événements fournis par les scripts EEM, tels qu'un message syslog, un modèle CLI et un planificateur Cron.

L'étape 1 est expliquée dans cette section. La section suivante fournit des exemples de mise en oeuvre des étapes 2 et 3.

Activer le shell invité avec une adresse IP statique

Procédez comme suit pour activer le shell invité :

1. Activer IOx.

<#root>

```
Switch#conf t
Switch(config)#iox
Switch(config)#
*Feb 17 18:13:24.440: %UICFGEXP-6-SERVER_NOTIFIED_START: Switch 1 R0/0: psd: Server iox has been r
*Feb 17 18:13:28.797: %IOX-3-IOX_RESTARTABITLITY: Switch 1 R0/0: run_ioxn_caf: Stack is in N+1 mod

*Feb 17 18:13:36.069: %IM-6-IOX_ENABLEMENT: Switch 1 R0/0: ioxman: IOX is ready.
```

2. Configurez le réseau d'hébergement d'applications pour l'interpréteur de commandes invité. Cet exemple utilise l'interface AppGigabitEthernet pour fournir un accès réseau ; Cependant, l'interface de gestion (Gi0/0) peut également être utilisée.

```
Switch(config)#int appgig1/0/1
Switch(config-if)#switchport mode trunk
Switch(config-if)#switchport trunk allowed vlan 20

Switch(config)#app-hosting appid guestshell
Switch(config-app-hosting)#app-vnic appGigabitEthernet trunk
Switch(config-config-app-hosting-trunk)#vlan 20 guest-interface 0
Switch(config-config-app-hosting-vlan-access-ip)#guest-ipaddress 10.20.1.2 netmask 255.255.255.0
Switch(config-config-app-hosting-vlan-access-ip)#exit
Switch(config-config-app-hosting-trunk)#exit
Switch(config-app-hosting)#app-default-gateway 10.20.1.1 guest-interface 0
Switch(config-app-hosting)#name-server0 10.31.104.74
Switch(config-app-hosting)#end
```

3. Activez l'interpréteur de commandes invité.

<#root>

```
Switch#guestshell enable
Interface will be selected if configured in app-hosting
Please wait for completion
guestshell installed successfully
Current state is: DEPLOYED
guestshell activated successfully
Current state is: ACTIVATED
guestshell started successfully
Current state is: RUNNING
```

Guestshell enabled successfully

4. Validez l'interpréteur de commandes invité. Cet exemple valide l'accessibilité avec la passerelle par défaut ainsi qu'avec cisco.com. En outre, validez que Python 3 peut être exécuté à partir de Guest Shell.

<#root>

```
! Validate that the Guest Shell is running.
Switch#
```

show app-hosting list

```
App id                                   State
---------------------------------------------------------
```

guestshell

RUNNING

```
Switch#
```

guestshell run bash

```
[guestshell@guestshell ~]$
```

! Validate that the IP address of the Guest Shell is configured correctly.

```
[guestshell@guestshell ~]$
```

sudo ifconfig

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

inet 10.20.1.2 netmask 255.255.255.0

```
  broadcast 10.20.1.255
        inet6 fe80::5054:ddff:fe61:24c7  prefixlen 64  scopeid 0x20
        ether 52:54:dd:61:24:c7  txqueuelen 1000  (Ethernet)
        RX packets 23  bytes 1524 (1.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9  bytes 726 (726.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 177  bytes 34754 (33.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 177  bytes 34754 (33.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

! Validate reachability to the default gateway and ensure that DNS is resolving correctly.

```
[guestshell@guestshell ~]$
```

ping 10.20.1.1

```
PING 10.20.1.1 (10.20.1.1) 56(84) bytes of data.
64 bytes from 10.20.1.1: icmp_seq=2 ttl=254 time=0.537 ms
64 bytes from 10.20.1.1: icmp_seq=3 ttl=254 time=0.537 ms
64 bytes from 10.20.1.1: icmp_seq=4 ttl=254 time=0.532 ms
64 bytes from 10.20.1.1: icmp_seq=5 ttl=254 time=0.574 ms
64 bytes from 10.20.1.1: icmp_seq=6 ttl=254 time=0.590 ms
^C
--- 10.20.1.1 ping statistics ---
6 packets transmitted, 5 received, 16.6667% packet loss, time 5129ms
rtt min/avg/max/mdev = 0.532/0.554/0.590/0.023 ms

[guestshell@guestshell ~]$
```

ping cisco.com

```
PING cisco.com (X.X.X.X) 56(84) bytes of data.
64 bytes from www1.cisco.com (X.X.X.X): icmp_seq=1 ttl=237 time=125 ms
64 bytes from www1.cisco.com (X.X.X.X): icmp_seq=2 ttl=237 time=125 ms
^C
--- cisco.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 124.937/125.141/125.345/0.204 ms
```

! Validate the Python version.

```
[guestshell@guestshell ~]$
```

python3 --version

Python 3.6.8

```
! Run Python commands within the Guest Shell.
[guestshell@guestshell ~]$

python3

Python 3.6.8 (default, Dec 22 2020, 19:04:08)
[GCC 8.4.1 20200928 (Red Hat 8.4.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>

print("Cisco")
Cisco

>>> exit()
[guestshell@guestshell ~]$

[guestshell@guestshell ~]$ exit
exit

Switch#
```

Activer le shell invité avec une adresse IP DHCP

Généralement, l'interpréteur de commandes invité est configuré avec une adresse IP statique, car le conteneur de l'interpréteur de commandes invité ne dispose pas du service client DHCP par défaut. S'il est nécessaire que l'interpréteur de commandes invité demande une adresse IP de manière dynamique, le service client DHCP doit être installé. Procédez comme suit :

1. Suivez les étapes pour activer l'interpréteur de commandes invité avec une adresse IP statique. Cependant, cette fois, n'attribuez pas l'adresse IP dans la configuration d'hébergement d'applications au cours de l'étape 2. Utilisez plutôt cette configuration :

```
Switch(config)#int appgig1/0/1
Switch(config-if)#switchport mode trunk
Switch(config-if)#switchport trunk allowed vlan 20

Switch(config)#app-hosting appid guestshell
Switch(config-app-hosting)#app-vnic appGigabitEthernet trunk
Switch(config-config-app-hosting-trunk)#vlan 20 guest-interface 0
Switch(config-app-hosting)#end
```

2. Le client DHCP peut être installé à l'aide de l'utilitaire Yum avec la commande`sudo yum install dhcp-client`. Cependant, les référentiels de CentOS Stream 8 ont été désactivés. Pour résoudre ce problème, les packages client DHCP peuvent être téléchargés et installés manuellement. Sur un PC, téléchargez ces packages depuis le coffre-fort CentOS Stream 8 et créez-les dans un fichier tar.

    - bind-export-libs-9.11.36-13.el8.x86_64.rpm
    - dhcp-client-4.3.6-50.el8.x86_64.rpm

- dhcp-common-4.3.6-50.el8.noarch.rpm
- dhcp-libs-4.3.6-50.el8.x86_64.rpm

```
[cisco@CISCO-PC guestshell-packages] % tar -cf dhcp-client.tar bind-export-libs-9.11.36-13.el8.x86
```

3. Copiez le fichier dansdhcp-client.tarle répertoire /flash/guest-share/ du commutateur.
4. Entrez une session bash Guest Shell et exécutez les commandes Linux pour installer le client DHCP et demander une adresse IP.

<#root>

```
513E.D.02-C9300X-12Y-A-17#

guestshell run bash


[guestshell@guestshell ~]$

sudo ifconfig

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536

<--- no eth0 interface

        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 149  bytes 32462 (31.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 149  bytes 32462 (31.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

! Unpack the packages needed for the DHCP client service.
[guestshell@guestshell ~]$

tar -xf /flash/guest-share/dhcp-client.tar

tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.metadata:kMDItemWhereFro
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.macl'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.metadata:kMDItemWhereFro
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.macl'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.metadata:kMDItemWhereFro
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.macl'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.quarantine'
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.metadata:kMDItemWhereFro
tar: Ignoring unknown extended header keyword 'LIBARCHIVE.xattr.com.apple.macl'

[guestshell@guestshell ~]$

ls

bind-export-libs-9.11.36-13.el8.x86_64.rpm  dhcp-common-4.3.6-50.el8.noarch.rpm
dhcp-client-4.3.6-50.el8.x86_64.rpm         dhcp-libs-4.3.6-50.el8.x86_64.rpm
```

! Install the packages using DNF.
[guestshell@guestshell ~]$

sudo dnf -y --disablerepo=* localinstall *.rpm

```
Warning: failed loading '/etc/yum.repos.d/CentOS-Base.repo', skipping.
Dependencies resolved.
================================================================================
 Package                 Architecture    Version                     Repository             Siz
================================================================================
Installing:
 bind-export-libs        x86_64          32:9.11.36-13.el8           @commandline          1.1
 dhcp-client             x86_64          12:4.3.6-50.el8             @commandline          319
 dhcp-common             noarch          12:4.3.6-50.el8             @commandline          208
 dhcp-libs               x86_64          12:4.3.6-50.el8             @commandline          148

Transaction Summary
================================================================================
Install  4 Packages

Total size: 1.8 M
Installed size: 3.9 M
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/
  Installing       : dhcp-libs-12:4.3.6-50.el8.x86_64                       1/
  Installing       : dhcp-common-12:4.3.6-50.el8.noarch                     2/
  Installing       : bind-export-libs-32:9.11.36-13.el8.x86_64              3/
  Running scriptlet: bind-export-libs-32:9.11.36-13.el8.x86_64              3/
  Installing       : dhcp-client-12:4.3.6-50.el8.x86_64                     4/
  Running scriptlet: dhcp-client-12:4.3.6-50.el8.x86_64                     4/
  Verifying        : bind-export-libs-32:9.11.36-13.el8.x86_64              1/
  Verifying        : dhcp-client-12:4.3.6-50.el8.x86_64                     2/
  Verifying        : dhcp-common-12:4.3.6-50.el8.noarch                     3/
  Verifying        : dhcp-libs-12:4.3.6-50.el8.x86_64                       4/

Installed:
  bind-export-libs-32:9.11.36-13.el8.x86_64          dhcp-client-12:4.3.6-50.el8.x86_64
  dhcp-common-12:4.3.6-50.el8.noarch                 dhcp-libs-12:4.3.6-50.el8.x86_64

Complete!
```

! Request a DHCP IP address for eth0.
[guestshell@guestshell ~]$

sudo dhclient eth0

! Validate the leased IP address.
[guestshell@guestshell ~]$

sudo ifconfig eth0

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
```

inet 10.1.1.12 netmask 255.255.255.0

```
  broadcast 10.1.1.255
```

```
        inet6 fe80::5054:ddff:fe85:a0d5  prefixlen 64  scopeid 0x20
        ether 52:54:dd:85:a0:d5  txqueuelen 1000  (Ethernet)
        RX packets 7  bytes 1000 (1000.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 1354 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[guestshell@guestshell ~]$

exit

exit

! You can validate the leased IP address from Cisco IOS XE too.
513E.D.02-C9300X-12Y-A-17#

guestshell run sudo ifconfig eth0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500


inet 10.1.1.12  netmask 255.255.255.0

  broadcast 10.1.1.255
        inet6 fe80::5054:ddff:fe85:a0d5  prefixlen 64  scopeid 0x20
        ether 52:54:dd:85:a0:d5  txqueuelen 1000  (Ethernet)
        RX packets 28  bytes 2344 (2.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13  bytes 1494 (1.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Scénarios :

Cas d'utilisation 1 : Enregistrement automatique des modifications de configuration sur un serveur SCP

Dans certaines situations, il est préférable d'enregistrer automatiquement les configurations de commutateur sur un serveur chaque fois que la commande est utilisée`write memory`. Cette pratique permet de conserver un enregistrement des modifications et de restaurer la configuration si nécessaire. Lors du choix d'un serveur, les protocoles TFTP et SCP peuvent être utilisés, mais un serveur SCP offre une couche de sécurité supplémentaire.

La fonction d'archivage de Cisco IOS fournit cette fonctionnalité. Cependant, un inconvénient important est que les identifiants SCP ne peuvent pas être masqués dans la configuration ; le chemin du serveur est affiché en texte clair dans les configurations en cours et de démarrage.

```
Switch#show running-config | section archive
archive
 path scp://cisco:Cisco!123@10.31.121.224/
 write-memory
```

En utilisant Guest Shell et Python, il est possible d'obtenir la même fonctionnalité tout en gardant les informations d'identification cachées. Pour ce faire, les variables d'environnement du shell invité sont exploitées pour stocker les identifiants de connexion SCP réels. Par conséquent, les informations d'identification du serveur SCP ne sont pas visibles dans la configuration en cours.

Dans cette approche, la configuration en cours n'affiche que le script EEM, tandis que les scripts Python construisent la commande avec les informations d'identification et la transmettentcopy running-config scp:au script EEM à exécuter.

Pour cet exemple, procédez comme suit :

1. Copiez le script Python dans le répertoire /flash/guest-share. Ce script lit les variables d'environnement SCP_USER et SCP_PASSWORD, et retourne lacopy startup-config scp:commande, afin que le script EEM puisse l'exécuter. Le script requiert l'adresse IP du serveur SCP comme argument. En outre, le script conserve un journal de chaque exécution de la commandewrite memorydans un fichier persistant situé à l'adresse /flash/guest-share/TAC-write-memory-log.txt. Voici le script Python :

```
import sys
import os
import cli
from datetime import datetime

# Get SCP server from the command-line argument (first argument passed)
scp_server = sys.argv[1]  # Expects the SCP server address as the first argument

# Configure CLI to suppress file prompts (quiet mode)
cli.configure("file prompt quiet")

# Get the current date and time
current_time = datetime.now()

# Format the current time for human-readable output and to use in filenames
formatted_time = current_time.strftime("%Y-%m-%d %H:%M:%S %Z")  # e.g., 2025-03-13 14:30:00 UTC
file_name_time = current_time.strftime("%Y-%m-%d_%H_%M_%S")  # e.g., 2025-03-13_14_30_00

# Retrieve SCP user and password from environment variables securely
scp_user = os.getenv('SCP_USER')  # SCP username from environment
scp_password = os.getenv('SCP_PASSWORD')  # SCP password from environment

# Ensure the credentials are set in the environment, raise error if missing
if not scp_user or not scp_password:
    raise ValueError("SCP user or password not found in environment variables!")

# Construct the SCP command to copy the file, avoiding exposure of sensitive data in print
# WARNING: The password should not be shared openly in logs or outputs.
print(f"copy startup-config scp://{scp_user}:{scp_password}@{scp_server}/config-backup-{file_name_

# Save the event in flash memory (log the write operation)
directory = '/flash/guest-share'  # Directory path where log will be saved
file_name = os.path.join(directory, 'TAC-write-memory-log.txt')  # Full path to log file

# Prepare the log entry with the formatted timestamp
line = f'{formatted_time}: Write memory operation.\n'
```

```
# Open the log file in append mode to add the new log entry
with open(file_name, 'a') as file:
    file.write(line)  # Append the log entry to the file
```

Dans cet exemple, le script Python est copié sur le commutateur à l'aide d'un serveur TFTP :

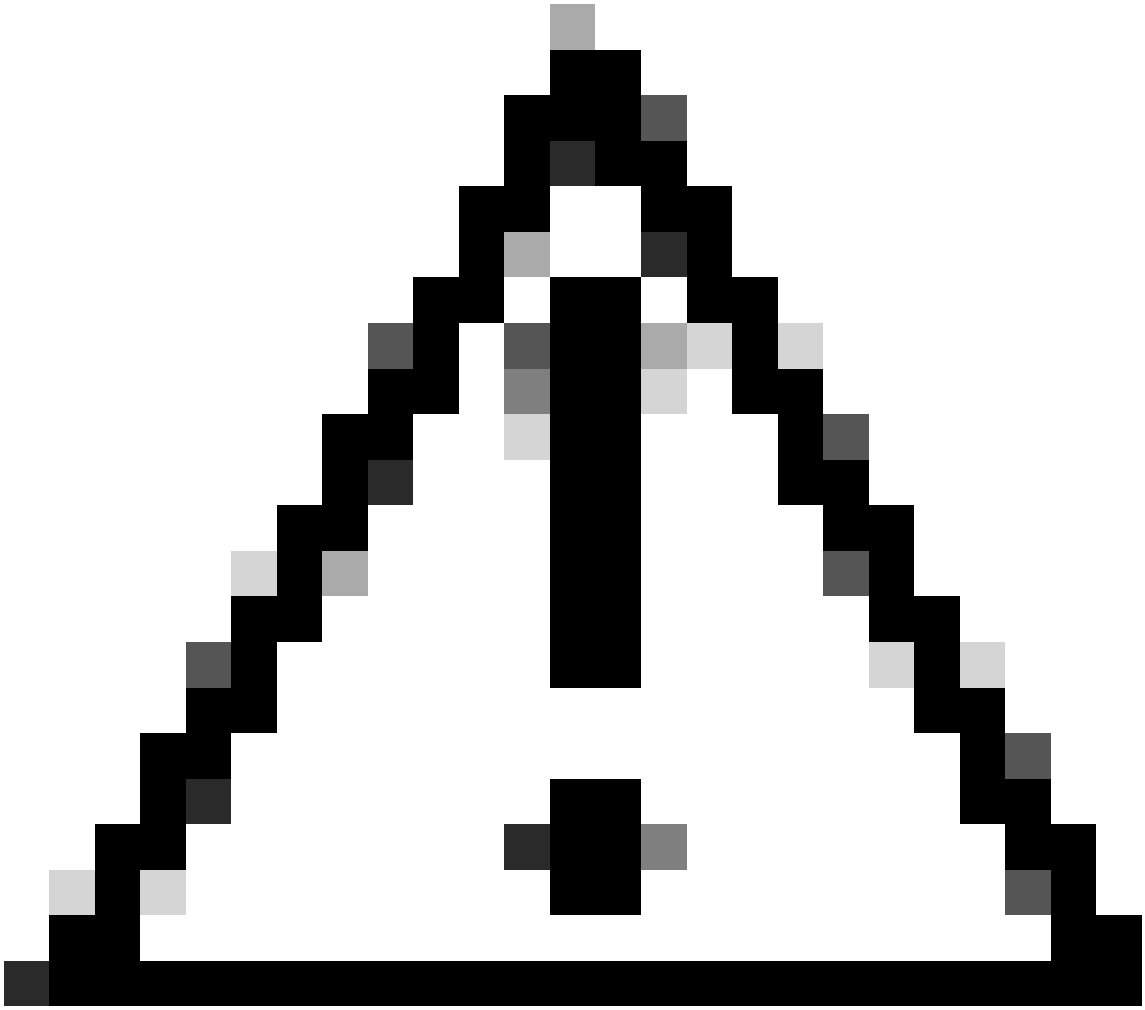
```
<#root>

Switch#

copy tftp://10.207.204.10/cat9k_scp_command.py flash:/guest-share/cat9k_scp_command.py


Accessing tftp://10.207.204.10/cat9k_scp_command.py...
Loading cat9k_scp_command.py from 10.207.204.10 (via GigabitEthernet0/0): !
[OK - 917 bytes]


917 bytes copied in 0.017 secs (53941 bytes/sec)
```

2. Installez le script EEM. Ce script appelle le script Python. qui renvoie la commande`copy startup-config scp:`nécessaire pour enregistrer la configuration sur le serveur SCP. Le script EEM exécute ensuite la commande retournée par le script Python.

```
event manager applet Python-config-backup authorization bypass
 event cli pattern "^write|write memory|copy running-config startup-config" sync no skip no maxrur
 action 0000 syslog msg "Config save detected, TAC EEM-python started."
 action 0005 cli command "enable"
 action 0015 cli command "guestshell run python3 /bootflash/guest-share/cat9k_scp_command.py 10.31
 action 0020 regexp "(^.*)\n" "$_cli_result" match command
 action 0025 cli command "$command"
 action 0030 syslog msg "TAC EEM-python script finished with result: $_cli_result"
```

3. Définissez les variables d'environnement Guest Shell en les insérant dans le fichier`~/.bashrc`. Cela garantit que les variables d'environnement sont persistantes à chaque ouverture d'un interpréteur de commandes invité, même après le rechargement du commutateur. Ajoutez ces deux lignes :

```
export SCP_USER="cisco"
export SCP_PASSWORD="Cisco!123"
```



Mise en garde : Les informations d'identification utilisées dans cet exemple sont uniquement à des fins éducatives. Ils ne sont pas destinés à être utilisés dans des environnements de production. Les utilisateurs doivent remplacer ces informations d'identification par leurs propres informations d'identification sécurisées et spécifiques à l'environnement.

C'est le processus pour ajouter ces variables au fichier~/.bashrc:

<#root>

```
! 1. Enter a Guest Shell bash session.
Switch#
```

guestshell run bash

```
! 2. Locate the ~/.bashrc file.
[guestshell@guestshell ~]$
```

ls ~/.bashrc

```
/home/guestshell/.bashrc

! 3. Add the SCP_USER and SCP_PASSWORD environment variables at the end of the ~/.bashrc file.
[guestshell@guestshell ~]$

echo 'export SCP_USER="cisco"' >> ~/.bashrc


[guestshell@guestshell ~]$

echo 'export SCP_PASSWORD="Cisco!123"' >> ~/.bashrc



! 4. To validate these 2 new lines were added correctly, display the content of the ~/.bashrc file
[guestshell@guestshell ~]$

cat ~/.bashrc

# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
[guestshell@guestshell ~]$ echo 'export SCP_USER="cisco"' >> ~/.bashrc
[guestshell@guestshell ~]$ echo 'export SCP_PASSWORD="Cisco!123"' >> ~/.bashrc
[guestshell@guestshell ~]$ cat ~/.bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific environment
if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
then
    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
fi
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

export SCP_USER="cisco"
export SCP_PASSWORD="Cisco!123"

! 5. Reload the ~/.bashrc file in the current session.
[guestshell@guestshell ~]$
```

```
source ~/.bashrc


! 6. Validate that the environment variables are added, then exit the Guest Shell session.
[guestshell@guestshell ~]$

printenv | grep SCP
SCP_USER=cisco
SCP_PASSWORD=Cisco!123

[guestshell@guestshell ~]$ exit

Switch#
```

4. Exécutez le script Python manuellement pour valider qu'il retourne la bonnecopycommande et consigne l'opération dans le fichier persistantTAC-write-memory-log.txt.

```
<#root>

Switch#

guestshell run python3 /flash/guest-share/cat9k_scp_command.py 10.31.121.224
copy startup-config scp://cisco:Cisco!123@10.31.121.224/config-backup-2025-01-25_18_35_18.txt


Switch#

dir flash:guest-share/

Directory of flash:guest-share/

286725  -rw-              368  Jan 25 2025 18:35:18 +00:00

TAC-write-memory-log.txt

286726  -rw-              903  Jan 25 2025 18:34:45 +00:00  cat9k_scp_command.py
286723  -rw-              144  Jan 25 2025 15:07:07 +00:00  TAC-shutdown-log.txt
286722  -rw-              977  Jan 25 2025 14:50:56 +00:00  cat9k_noshut.py

11353194496 bytes total (3751542784 bytes free)

Switch#

more flash:/guest-share/TAC-write-memory-log.txt
2025-01-25 18:35:18 : Write memory operation.
```

5. Testez le script EEM. Ce script EEM envoie également un syslog avec le résultat de l'opération de copie, qu'elle soit réussie ou échouée. Voici un exemple d'exécution réussie :

```
<#root>

Switch#

write memory
```

```
Building configuration...
[OK]
Switch#
*Jan 25 19:23:22.189: %HA_EM-6-LOG: Python-config-backup: Config save detected, TAC EEM-python sta
*Jan 25 19:23:42.885: %HA_EM-6-LOG: Python-config-backup:

TAC EEM-python script finished with result:
Writing config-backup-2025-01-25_19_23_26.txt !
8746 bytes copied in 15.175 secs (576 bytes/sec)

Switch#


Switch#

more flash:guest-share/TAC-write-memory-log.txt
2025-01-25 19:23:26 : Write memory operation.
```

Pour tester un transfert ayant échoué, le serveur SCP est arrêté. Voici le résultat de cette exécution ratée :

```
<#root>

Switch#

write

Building configuration...
[OK]
Switch#
*Jan 25 19:25:31.439: %HA_EM-6-LOG: Python-config-backup: Config save detected, TAC EEM-python sta
*Jan 25 19:26:06.934: %HA_EM-6-LOG: Python-config-backup:

TAC EEM-python script finished with result:
Writing config-backup-2025-01-25_19_25_36.txt % Connection timed out; remote host not responding

%Error writing scp://*:*@10.31.121.224/config-backup-2025-01-25_19_25_36.txt (Undefined error)

Switch#
Switch#

Switch#
Switch#

more flash:guest-share/TAC-write-memory-log.txt

2025-01-25 19:23:26 : Write memory operation.

2025-01-25 19:25:36 : Write memory operation.
```

Cas d'utilisation 2 : Surveillance des incréments dans les modifications de topologie STP

Cet exemple est utile pour surveiller les problèmes liés à l'instabilité du Spanning Tree et identifier l'interface qui reçoit les notifications de modification de topologie (TCN). Le script EEM est exécuté

périodiquement à un intervalle de temps spécifié et appelle un script Python qui exécute une commande show et vérifie si les TCN ont augmenté.

La création de ce script en utilisant uniquement des commandes EEM nécessiterait l'utilisation de boucles for et de plusieurs correspondances regex, ce qui serait fastidieux. Par conséquent, cet exemple montre comment le script EEM délègue cette logique complexe à Python.

Pour cet exemple, procédez comme suit :

1. Copiez le script Python dans le répertoire /flash/guest-share/. Ce script effectue les tâches suivantes :

    1. Il exécute la commande`show spanning-tree detail`et analyse le résultat pour enregistrer les informations TCN pour chaque VLAN dans un dictionnaire.

    2. Il compare les informations TCN analysées avec les données du fichier JSON de l'exécution du script précédent. Pour chaque VLAN, si les TCN ont augmenté, un message syslog est envoyé avec des informations similaires à cet exemple :

```
*Jan 31 18:57:37.852: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY
```

    3. Il enregistre les informations TCN actuelles dans un fichier JSON à comparer lors de la prochaine exécution. Voici le script Python :

```
import os
import json
import cli
import re
from datetime import datetime


def main():
    # Get TCNs by running the CLI command to show spanning tree details
    tcns = cli.cli("show spanning-tree detail")

    # Parse the output into a dictionary of VLAN details
    parsed_tcns = parse_stp_detail(tcns)

    # Path to the JSON file where VLAN TCN data will be stored
    file_path = '/flash/guest-share/tcns.json'

    # Initialize an empty dictionary to hold stored TCN data
    stored_tcn = {}

    # Check if the file exists and process it if it does
    if os.path.exists(file_path):
        try:
            # Open the JSON file and parse its contents into stored_tcn
            with open(file_path, 'r') as f:
                stored_tcn = json.load(f)
```

```
                result = compare_tcn_sets(stored_tcn, parsed_tcns)

                # Check each VLAN in the result and log changes if TCN increased
                for vlan_id, vlan_data in result.items():
                    if vlan_data['tcn_increased']:
                        log_message = (
                            f"TCNs increased in VLAN {vlan_id} "
                            f"from {vlan_data['old_tcn']} to {vlan_data['new_tcn']}. "
                            f"Last TCN seen on {vlan_data['source_interface']}."
                        )
                        # Send log message using CLI
                        cli.cli(f"send log facility GUESTSHELL severity 5 mnemonics PYTHON_S

            except json.JSONDecodeError:
                print("Error: The file contains invalid JSON.")
            except Exception as e:
                print(f"An error occurred: {e}")

        # Write the current TCN data to the JSON file for future comparison
        with open(file_path, 'w') as f:
            json.dump(parsed_tcns, f, indent=4)


    def parse_stp_detail(cli_output: str):
        """
        Parses the output of "show spanning-tree detail" into a dictionary of VLANs and their TC

        Args:
            cli_output (str): The raw output from the "show spanning-tree detail" command.

        Returns:
            dict: A dictionary where the keys are VLAN IDs and the values contain TCN details.
        """
        vlan_info = {}

        # Regular expressions to match various parts of the "show spanning-tree detail" output
        vlan_pattern = re.compile(r'^\s*(VLAN|MST)(\d+)\s*', re.MULTILINE)
        tcn_pattern = re.compile(r'^\s*Number of topology changes (\d+)\s*', re.MULTILINE)
        last_tcn_pattern = re.compile(r'last change occurred (\d+:\d+:\d+) ago\s*', re.MULTILINE
        last_tcn_days_pattern = re.compile(r'last change occurred (\d+d\d+h) ago\s*', re.MULTILI
        tcn_interface_pattern = re.compile(r'from ([a-zA-Z]+[\d+\/]+\d+)\s*', re.MULTILINE)

        # Find all VLAN blocks in the output
        vlan_blocks = vlan_pattern.split(cli_output)[1:]
        vlan_blocks = [item for item in vlan_blocks if item not in ["VLAN", "MST"]]

        for i in range(0, len(vlan_blocks), 2):
            vlan_id = vlan_blocks[i].strip()

            # Match the relevant patterns for TCN and related details
            tcn_match = tcn_pattern.search(vlan_blocks[i + 1])
            last_tcn_match = last_tcn_pattern.search(vlan_blocks[i + 1])
            last_tcn_days_match = last_tcn_days_pattern.search(vlan_blocks[i + 1])
            tcn_interface_match = tcn_interface_pattern.search(vlan_blocks[i + 1])

            # Parse the TCN details and add to the dictionary
            if last_tcn_match:
                tcn = int(tcn_match.group(1))
                last_tcn = last_tcn_match.group(1)
                source_interface = tcn_interface_match.group(1) if tcn_interface_match else None
                vlan_info[vlan_id] = {
                    "id_int": int(vlan_id),
```

```
                "tcn": tcn,
                "last_tcn": last_tcn,
                "source_interface": source_interface,
                "tcn_in_last_day": True
            }
        elif last_tcn_days_match:
            tcn = int(tcn_match.group(1))
            last_tcn = last_tcn_days_match.group(1)
            source_interface = tcn_interface_match.group(1) if tcn_interface_match else None
            vlan_info[vlan_id] = {
                "id_int": int(vlan_id),
                "tcn": tcn,
                "last_tcn": last_tcn,
                "source_interface": source_interface,
                "tcn_in_last_day": False
            }

    return vlan_info


def compare_tcn_sets(set1, set2):
    """
    Compares two sets of VLAN TCN data to determine if TCN values have increased.

    Args:
        set1 (dict): The first set of VLAN TCN data.
        set2 (dict): The second set of VLAN TCN data.

    Returns:
        dict: A dictionary indicating whether the TCN has increased for each VLAN.
    """
    tcn_changes = {}

    # Compare TCN values for VLANs that exist in both sets
    for vlan_id, vlan_data_1 in set1.items():
        if vlan_id in set2:
            vlan_data_2 = set2[vlan_id]
            tcn_increased = vlan_data_2['tcn'] > vlan_data_1['tcn']
            tcn_changes[vlan_id] = {
                'tcn_increased': tcn_increased,
                'old_tcn': vlan_data_1['tcn'],
                'new_tcn': vlan_data_2['tcn'],
                'source_interface': vlan_data_2['source_interface']
            }
        else:
            tcn_changes[vlan_id] = {
                'tcn_increased': None,  # No comparison if VLAN is not in set2
                'old_tcn': vlan_data_1['tcn'],
                'new_tcn': None
            }

    # Check for VLANs in set2 that are not in set1
    for vlan_id, vlan_data_2 in set2.items():
        if vlan_id not in set1:
            tcn_changes[vlan_id] = {
                'tcn_increased': None,  # No comparison if VLAN is not in set1
                'old_tcn': None,
                'new_tcn': vlan_data_2['tcn']
            }

    return tcn_changes
```

```
if __name__ == "__main__":
    main()
```

Dans cet exemple, le script Python est copié sur le commutateur à l'aide d'un serveur TFTP :

```
<#root>

Switch#

copy tftp://10.207.204.10/cat9k_tcn.py flash:/guest-share/cat9k_tcn.py

Accessing tftp://10.207.204.10/cat9k_tcn.py...
Loading cat9k_tcn.py from 10.207.204.10 (via GigabitEthernet0/0): !
[OK - 5739 bytes]


5739 bytes copied in 0.023 secs (249522 bytes/sec)
```

2. Installez le script EEM. Dans cet exemple, la seule tâche du script EEM est d'exécuter le script Python, qui envoie un message de journalisation si un incrément TCN est détecté. Dans cet exemple, le script EEM s'exécute toutes les 5 minutes.

```
event manager applet tcn_monitor authorization bypass
 event timer watchdog time 300
 action 0000 syslog msg "TAC EEM-python script started."
 action 0005 cli command "enable"
 action 0015 cli command "guestshell run python3 /bootflash/guest-share/cat9k_tcn.py"
 action 0020 syslog msg "TAC EEM-python script finished."
```

3. Pour valider la fonctionnalité du script, vous pouvez soit exécuter le script Python manuellement ou attendre cinq minutes que le script EEM l'appelle. Dans les deux cas, un syslog est envoyé uniquement si les TCN ont augmenté pour un VLAN.

```
<#root>

Switch#

more flash:/guest-share/tcns.json


{
"0001": {
"id_int": 1,
"tcn": 20,
"last_tcn": "00:01:18",
"source_interface": "TwentyFiveGigE1/0/5",
"tcn_in_last_day": true
```

```
},
"0010": {
"id_int": 10,
"tcn": 2,
"last_tcn": "00:00:22",
"source_interface": "TwentyFiveGigE1/0/1",
"tcn_in_last_day": true
},
"0020": {
"id_int": 20,
"tcn": 2,
"last_tcn": "00:01:07",
"source_interface": "TwentyFiveGigE1/0/2",
"tcn_in_last_day": true
},
"0030": {
"id_int": 30,
"tcn": 1,
"last_tcn": "00:01:18",
"source_interface": "TwentyFiveGigE1/0/3",
"tcn_in_last_day": true
}
}

Switch#
guestshell run python3 /flash/guest-share/cat9k_tcn.py

Switch#
*Feb 17 21:34:45.846: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY): TCN
Switch#
```

4. Testez le script EEM en attendant qu'il s'exécute toutes les cinq minutes. Si les TCN ont augmenté pour un VLAN, un message syslog est envoyé. Dans cet exemple particulier, il est noté que les TCN augmentent constamment sur le VLAN 30, et l'interface recevant ces TCN constants est Twe1/0/3.

```
<#root>

*Feb 17 21:56:23.563: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script started.
*Feb 17 21:56:26.039: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY):
TCNs increased in VLAN 0030 from 3 to 5. Last TCN seen on TwentyFiveGigE1/0/3.
*Feb 17 21:56:26.585: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script finished.
*Feb 17 22:01:23.563: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script started.
*Feb 17 22:01:26.687: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script finished.
```

```
*Feb 17 22:06:23.564: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script started.
*Feb 17 22:06:26.200: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY):

TCNs increased in VLAN 0030 from 5 to 9. Last TCN seen on TwentyFiveGigE1/0/3.

*Feb 17 22:06:26.787: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script finished.
*Feb 17 22:11:23.564: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script started.
*Feb 17 22:11:26.079: %GUESTSHELL-5-PYTHON_SCRIPT: Message from tty73(user id: shxUnknownTTY):

TCNs increased in VLAN 0030 from 9 to 12. Last TCN seen on TwentyFiveGigE1/0/3.

*Feb 17 22:11:26.686: %HA_EM-6-LOG: tcn_monitor: TAC EEM-python script finished.
```

Informations connexes

- Guide de configuration de la programmabilité, Cisco IOS XE Cupertino 17.9.x (Chapitre : Shell invité)
- Comprendre les meilleures pratiques et les scripts utiles pour EEM
- Livre blanc sur l'hébergement d'applications sur les commutateurs Cisco Catalyst 9000

À propos de cette traduction
Cisco a traduit ce document en traduction automatisée vérifiée par une personne dans le cadre d'un service mondial permettant à nos utilisateurs d'obtenir le contenu d'assistance dans leur propre langue.

Il convient cependant de noter que même la meilleure traduction automatisée ne sera pas aussi précise que celle fournie par un traducteur professionnel.