使用Python指令碼自動運行Catalyst 9000交換機

目錄

簡介

本文說明如何使用Python指令碼擴展EEM，以在Catalyst 9000交換機上自動執行配置和資料收集。

必要條件

需求

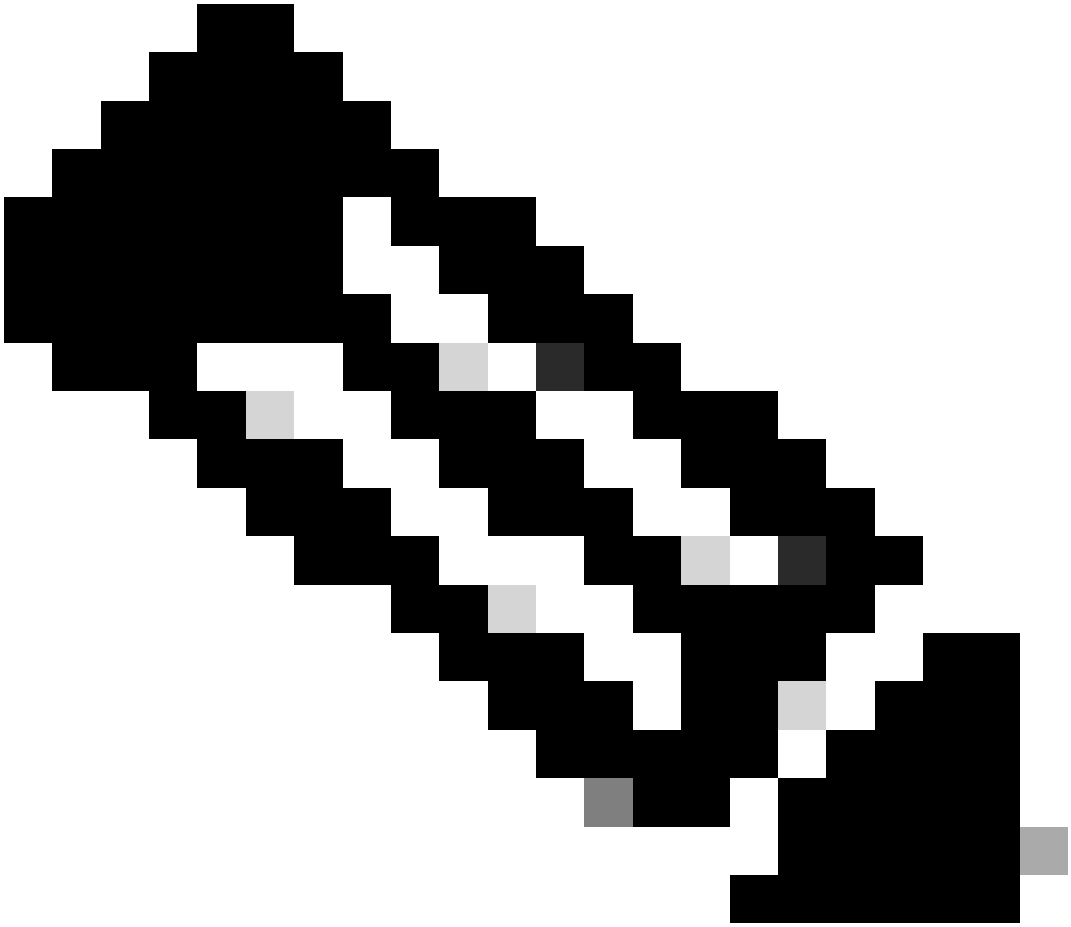
思科建議您瞭解並熟悉以下主題：

- Cisco IOS®和Cisco IOS® XE EEM
- 應用託管和訪客外殼
- Python指令碼
- Linux命令

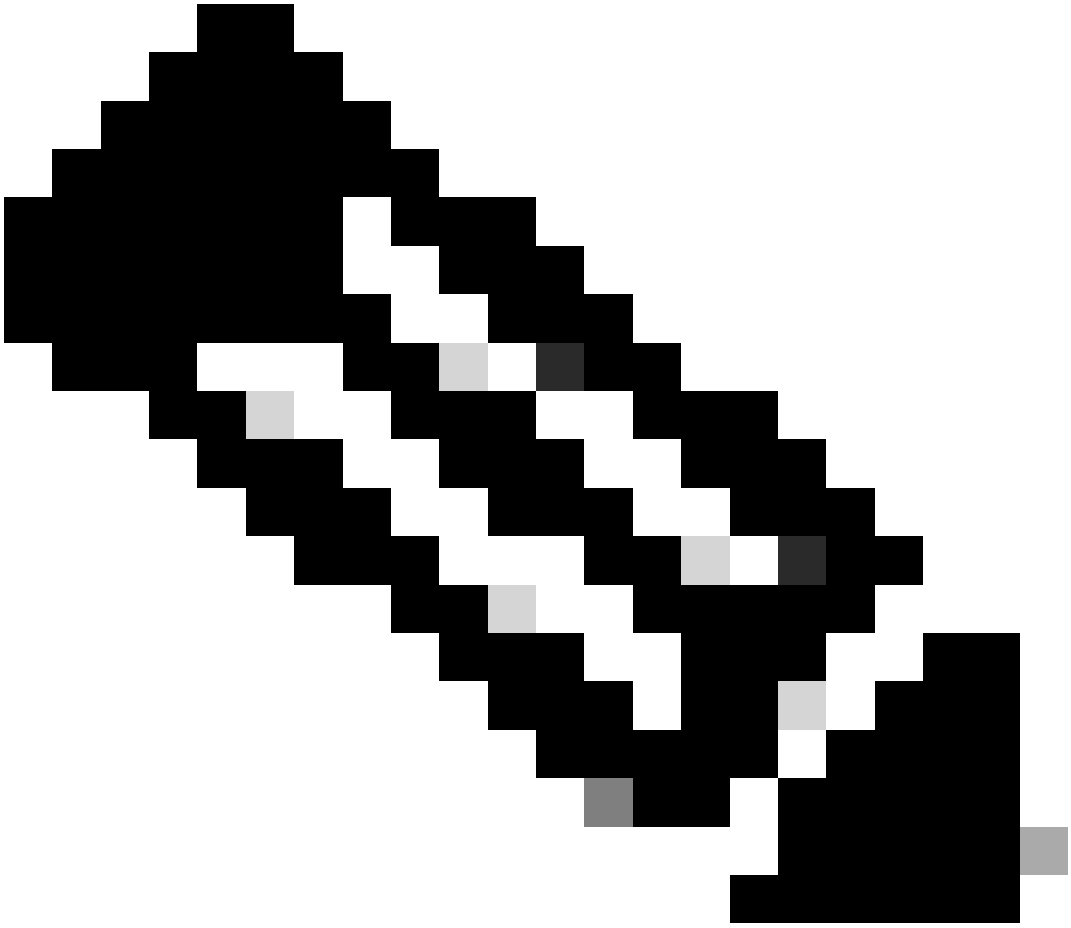
採用元件

本文中的資訊係根據以下軟體和硬體版本：

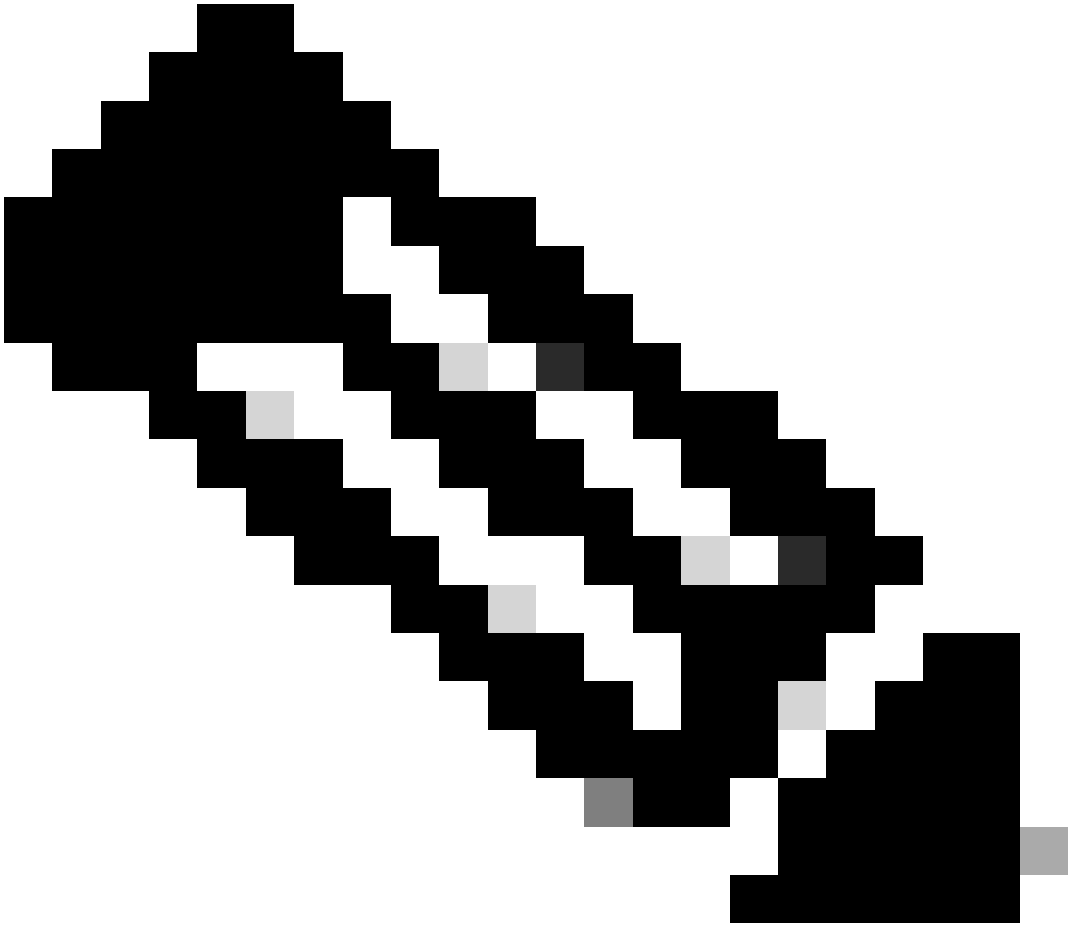
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1及更高版本



附註：請參閱適當的組態設定指南來瞭解使用的命令，以便在其他思科平台上啟用這些功能。



附註：Catalyst 9200L交換器不支援訪客Shell。



附註：Cisco TAC不支援這些指令碼，這些指令碼按原樣提供，用於教學目的。

本文中的資訊是根據特定實驗室環境內的裝置所建立。文中使用到的所有裝置皆從已清除（預設）的組態來啟動。如果您的網路運作中，請確保您瞭解任何指令可能造成的影響。

慣例

如需檔案慣例的相關資訊，請參閱思科技術提示慣例。

背景資訊

Guest Shell和Python指令碼

在Cisco Catalyst 9000系列交換機上託管應用可為合作夥伴和開發人員帶來創新機會，因為網路裝置可以與應用運行時環境合併。

它支援容器化應用，提供與主作業系統和Cisco IOS XE核心的完全隔離。這種分離可確保託管應用

的資源分配與核心路由和交換功能不同。

適用於Cisco IOS XE裝置的應用託管基礎設施稱為IOx(Cisco IOS + Linux)，它有助於將思科、合作夥伴和第三方開發人員開發的應用和服務託管到網路裝置上，確保跨各種硬體平台無縫整合。

Guest Shell是一種專用容器部署，它舉例說明了有利於系統部署的應用程式。

Guest Shell提供基於Linux的虛擬化環境，旨在運行自定義Linux應用程式（包括Python），以實現對思科裝置的自動控制和管理。Guest Shell容器允許使用者在系統中運行指令碼和應用。具體來說，在Intel x86平台上，Guest Shell容器是一個Linux容器(LXC)，其中包含CentOS 8.0最小根檔案系統。在Cisco IOS XE Amsterdam 17.3.1及更高版本中，僅支援Python V3.6。在運行時可使用CentOS 8.0中的Yum實用程式安裝其他Python庫。也可以使用Pip安裝包(PIP)安裝或更新Python包。

Guest Shell包括Python應用程式程式設計介面(API)，允許使用Python CLI模組運行Cisco IOS XE命令。通過這種方式，Python指令碼增強了自動化功能，為網路工程師提供了一種多功能工具，用於開發用於自動執行配置和資料收集任務的指令碼。雖然這些指令碼可以通過CLI手動運行，但是也可以與EEM指令碼一起用於響應特定事件，如系統日誌消息、介面事件或命令運行。實際上，任何可能觸發EEM指令碼的事件也可用於觸發Python指令碼，從而擴大Cisco Catalyst 9000交換機內部的自動化潛力。

安裝Guest Shell後，將在快閃記憶體檔案系統中自動建立來賓共用目錄。這是可以通過Python指令碼和訪客Shell訪問的檔案系統。為確保使用堆疊時正確同步，請將此資料夾保持在50 MB以下。

將Python與EEM指令碼配合使用的優點

- Python允許在Python指令碼中處理複雜的邏輯（如正規表示式、循環和匹配），從而擴展了EEM指令碼的自動化功能。此功能提供了建立更強大的EEM指令碼的機會。
- 作為一種著名的程式語言，Python降低了希望自動化Cisco IOS XE裝置的網路工程師的進入障礙。此外，它還提供了可維護性和可讀性。
- Python還提供錯誤處理功能以及強大的標準庫。

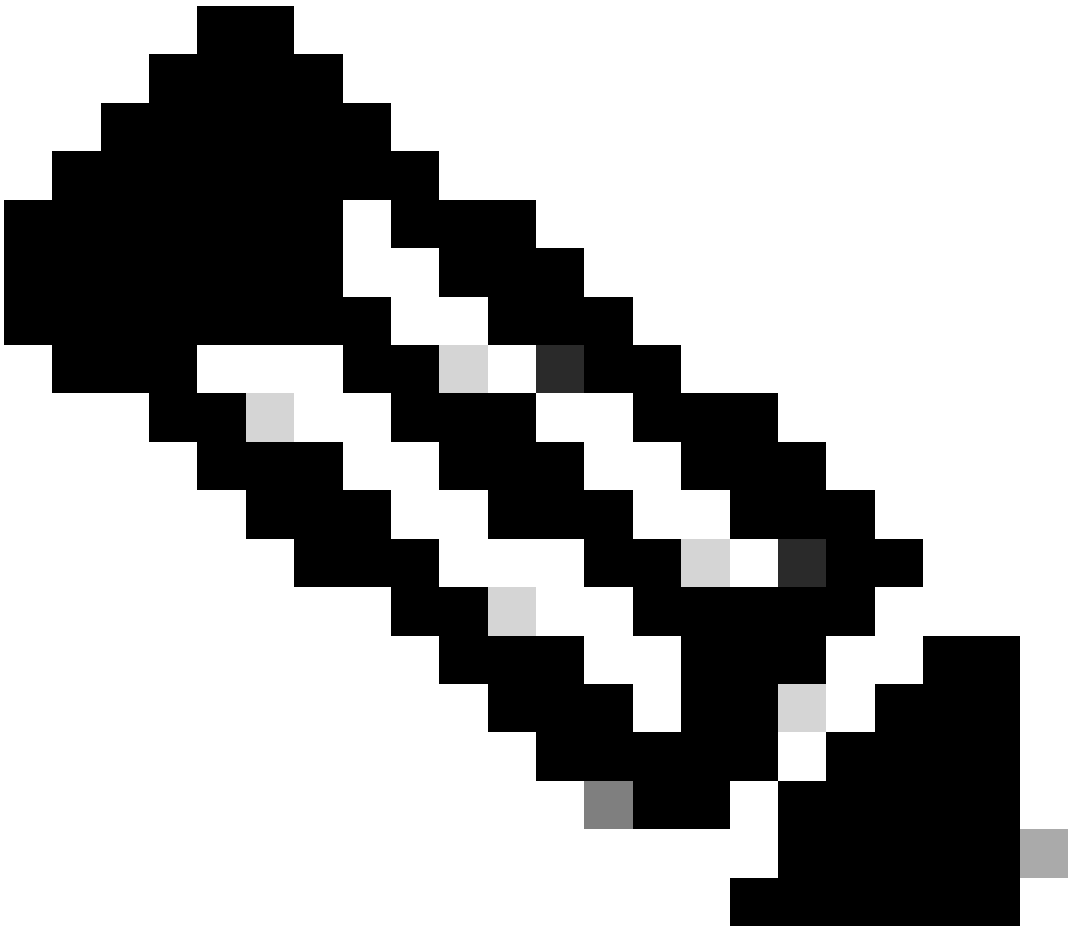
對EEM指令碼使用Python的注意事項

- Guest Shell在預設情況下未啟用，因此需要先啟用它，然後才能運行Python指令碼。
- 無法直接在CLI中建立Python指令碼；它們需要首先在開發環境中開發，然後複製到交換機的快閃記憶體中。

Cisco IOS XE中的SELinux

從Cisco IOS XE 17.8.1開始，引入了對安全增強型Linux(SELinux)的支援，以通過管理進程、使用者和檔案相互互動方式的策略來實施安全。SELinux策略定義允許進程或使用者訪問哪些操作和資源。當使用者或進程嘗試執行策略不允許的操作（例如訪問資源或運行命令）時，可能會發生衝突。SELinux可以在兩種不同模式下運行：

1. 允許模式：SELinux不會實施任何策略。但是，它仍會記錄任何違規行為，好像這些行為被拒絕一樣。
2. 實施：SELinux會在系統上主動實施安全策略。如果操作違反了SELinux策略，則拒絕並記錄該操作。



附註：在Cisco IOS XE 17.8.1中引入預設模式時，該模式被設定為允許模式。但是，從版本17.14.1開始，SELinux在執行模式下啟用。

使用Guest Shell時，在使用強制模式時可能會拒絕訪問某些資源。如果嘗試使用Guest Shell或Python指令碼執行操作時導致許可權被拒絕錯誤，則與此日誌類似：

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

要驗證SELinux是否拒絕指令碼，請使用命令show platform software audit summary檢查「拒絕計數」是否增加

。此外show platform software audit all，還會顯示被SELinux阻止的操作的日誌。訪問向量快取(AVC)是用於記錄SELinux中的訪問控制決策的機制，因此，使用此命令時，請查詢以type=AVC開頭的日誌。

如果指令碼被拒絕和阻止，則可以使用命令將SELinux設定為允許模式set platform software selinux permissive。此變更不會儲存在執行或啟動組態中，因此重新載入後，模式會回覆為強制執行。因此，每次重新載入交換機時，都需要重新應用此更改。可以使用驗證更改show platform software selinux。

設定

要使用EEM和Python指令碼自動執行交換機上的任務，請執行以下步驟：

1. 啟用Guest Shell。
2. 將Python指令碼複製到/flash/guest-share/目錄。您可以使用Cisco IOS XE中可用的任何複製機制，例如SCP、FTP或WebUI中的File Manager。Python指令碼在快閃記憶體中後，可以使用命令guestshell run python3 /flash/guest-share/cat9k_script.py運行該指令碼。
3. 配置運行Python腳本的EEM指令碼。此設定允許使用EEM指令碼提供的任意多個事件檢測器（如系統日誌消息、CLI模式和Cron排程程式）觸發Python指令碼。

本節將介紹步驟1。下一部分提供了演示如何實施步驟2和步驟3的示例。

使用靜態IP地址啟用來賓外殼

按照以下過程啟用訪客外殼：

1. 啟用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. 為訪客Shell配置應用宿主網路。此示例使用AppGigabitEthernet介面提供網路訪問；但是，也可以使用管理介面(Gi0/0)。

```
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. 啟用Guest Shell。

<#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. 驗證Guest Shell。此範例驗證預設閘道與cisco.com是否可連線。此外，驗證Python 3是否可從訪客外殼運行。

<#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#
```

使用DHCP IP地址啟用來賓外殼

通常，訪客Shell配置有靜態IP地址，因為預設情況下Guest Shell容器沒有DHCP客戶端服務。如果訪客外殼需要動態請求IP地址，則需要安裝DHCP客戶端服務。請遵循以下流程：

1. 按照以下步驟啟用具有靜態IP地址的來賓外殼。但是，這一次，請勿在步驟2中分配應用託管配置中的IP地址。請改用以下配置：

```
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. 可以使用命令sudo yum install dhcp-client的Yum實用程式安裝DHCP客戶端。然而，CentOS Stream 8的儲存庫已經停用。為了解決此問題，可以手動下載和安裝DHCP客戶端軟體包。在PC上，從CentOS Stream 8 vault下載這些軟體包並將其打包為tar檔案。
    - 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. 將檔案dhcp-client.tar，複製到交換器中的/flash/guest-share/目錄。
4. 輸入Guest Shell bash session，然後運行Linux命令以安裝DHCP客戶端並請求IP地址。

```
<#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
```

使用案例

使用案例1:在SCP伺服器上自動儲存配置更改

write memory 在某些情況下，每次使用命令時自動將交換機配置儲存到服務器是有益的。這種做法有助於維護更改記錄，並在必要時允許配置回滾。選擇伺服器時，可以使用TFTP和SCP，但SCP伺服器可提供額外的安全層。

Cisco IOS存檔功能提供此功能。但是，一個重大缺點是配置中不能隱藏SCP憑證；伺服器路徑在運行配置和啟動配置中均以純文字檔案顯示。

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

通過使用Guest Shell和Python，可以在保持憑證隱藏的同時實現相同的功能。這是通過利用訪客Shell中的環境變數儲存實際SCP憑據來實現的。因此，SCP伺服器憑據在運行配置中不可見。

在此方法中，運行配置僅顯示EEM指令碼，而Python指令碼使用憑證構建命令copy running-config scp:，並將其傳遞給要運行的EEM指令碼。

請依照以下步驟操作本範例：

1. 將Python指令碼複製到/flash/guest-share目錄。此指令碼讀取環境變數SCP_USER和SCP_PASSWORD，並返回copy startup-config scp:命令，以便EEM指令碼可以運行它。指令碼需要SCP伺服器IP地址作為引數。此外，指令碼會維護每次在位於/flash/guest-share/TAC-write-memory-log.txt的永久文件中運行命令時的日誌write memory。以下是Python指令碼：

```
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
```

在以下範例中，使用TFTP伺服器將Python指令碼複製到交換器：

```
<#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)
```

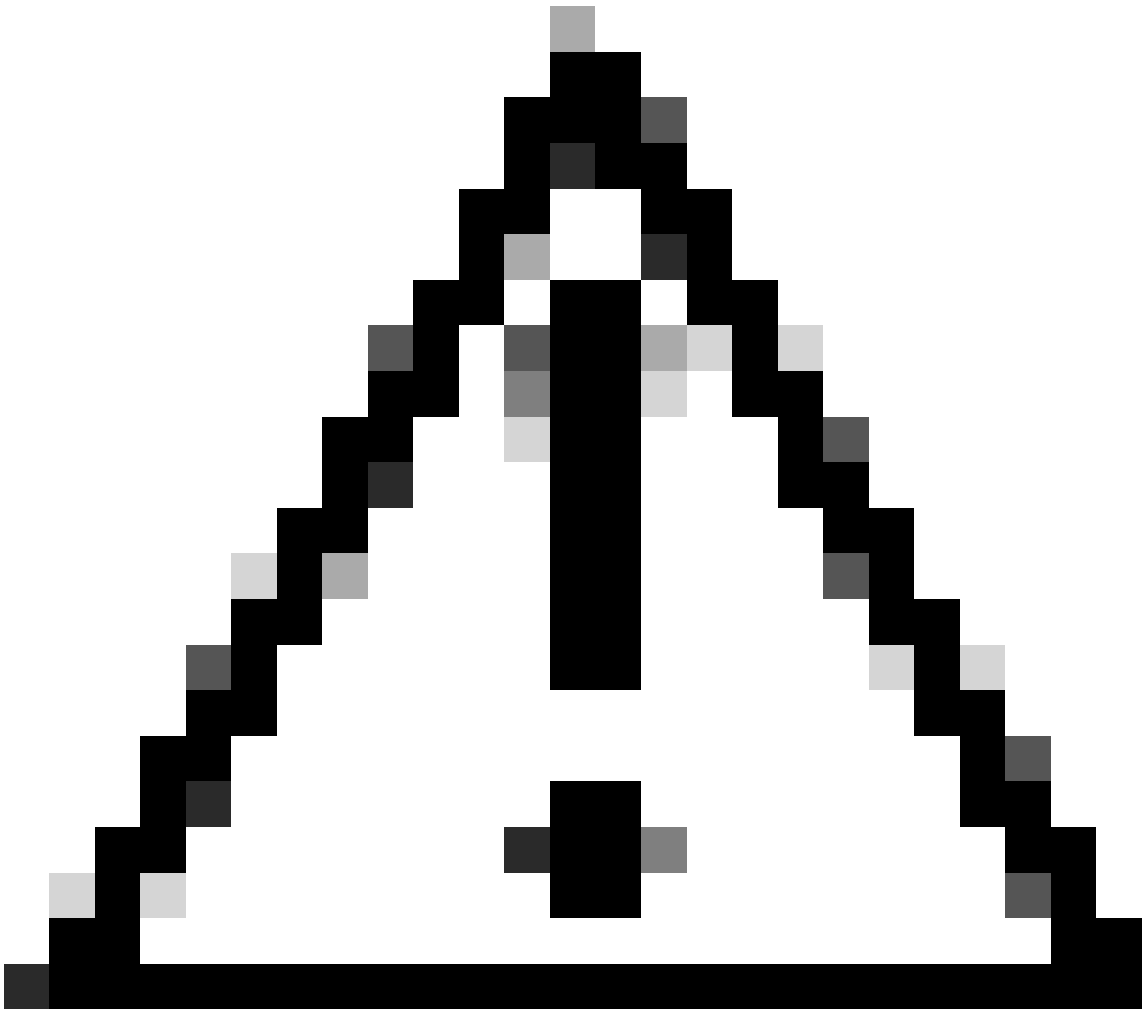
2. 安裝EEM指令碼。此指令碼呼叫Python指令碼。返回在SCP伺服器上儲存配置所需的命令copy startup-config scp:。然後，EEM指令碼運行Python指令碼返回的命令。

```
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. 將Guest Shell環境變數插入到檔案中來設定這些~/.bashrc變數。這可確保每次開啟訪客Shell時環境變數都是持久的，即使在重新載入交換機後也是如此。新增以下兩行：

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



注意：本示例中使用的憑據僅用於教育目的。它們不能用於生產環境。使用者需要用他們自己的安全、特定於環境的憑據替換這些憑據。

這是將這些變數新增到檔案的過程~/.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. 手動運行Python指令碼，以驗證它是否返回正確的命令copy，並將操作記錄在持久檔案TAC-write-memory-log.txt中。

```
<#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. 測試EEM指令碼。此EEM指令碼還會傳送包含複製操作結果的系統日誌，無論複製操作成功還是失敗。以下是成功執行的範例：

```
<#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.
```

要測試失敗的傳輸，SCP伺服器將關閉。這是此失敗運行的結果：

```
<#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.
```

使用案例2:監控STP拓撲更改的增量

此示例對於監控與生成樹不穩定相關的問題，以及確定哪個介面正在接收拓撲更改通知(TCN)非常有用。 EEM指令碼以指定的時間間隔定期運行，並呼叫運行show命令並檢查TCN是否增加的Python指令碼。

僅使用EEM命令建立此指令碼需要用於循環和多個正規表示式匹配，這將非常麻煩。因此，此示例展示EEM指令碼如何將此複雜邏輯委託給Python。

請依照以下步驟操作本範例：

1. 將Python指令碼複製到/flash/guest-share/目錄。此指令碼執行以下任務：
    1. 它會運行show spanning-tree detail命令並解析輸出，以便將每個VLAN的TCN資訊儲存在字典中。
    2. 它將分析的TCN資訊與上次運行指令碼的JSON檔案中的資料進行比較。如果每個VLAN的TCN都增加了，系統就會傳送包含類似以下示例的資訊的syslog消息：

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

    3. 它將當前TCN資訊儲存在JSON檔案中，以便在下次運行期間進行比較。以下是Python指令碼：

```
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()
```

在以下範例中，使用TFTP伺服器將Python指令碼複製到交換器：

```
<#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. 安裝EEM指令碼。在此示例中，EEM指令碼的唯一任務是運行Python指令碼，該指令碼在檢測到TCN增量時傳送日誌消息。在此示例中，EEM指令碼每5分鐘運行一次。

```
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. 要驗證指令碼的功能，您可以手動運行Python指令碼，或者等待5分鐘以讓EEM指令碼呼叫該指令碼。在這兩種情況下，系統日誌僅在某個VLAN的TCN增加時才傳送。

```
<#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. 通過等待EEM指令碼每五分鐘運行一次來測試該指令碼。如果任何VLAN的TCN都增加了，則會傳送系統日誌消息。在此特定範例中，請注意VLAN 30上的TCN不斷增加，而接收這些固定TCN的介面是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.
```

相關資訊

- 可程式設計性配置指南，Cisco IOS XE Cupertino 17.9.x(章節：訪客外殼)
- 瞭解EEM的最佳實踐和有用的指令碼
- Cisco Catalyst 9000系列交換機上的應用託管白皮書

關於此翻譯

思科已使用電腦和人工技術翻譯本文件，讓全世界的使用者能夠以自己的語言理解支援內容。請注意，即使是最佳機器翻譯，也不如專業譯者翻譯的內容準確。Cisco Systems, Inc. 對這些翻譯的準確度概不負責，並建議一律查看原始英文文件（提供連結）。