Automazione degli switch Catalyst 9000 con script Python

Sommario

Introduzione

Questo documento descrive come estendere EEM con script Python per automatizzare la configurazione e la raccolta dei dati sugli switch Catalyst 9000.

Prerequisiti

Requisiti

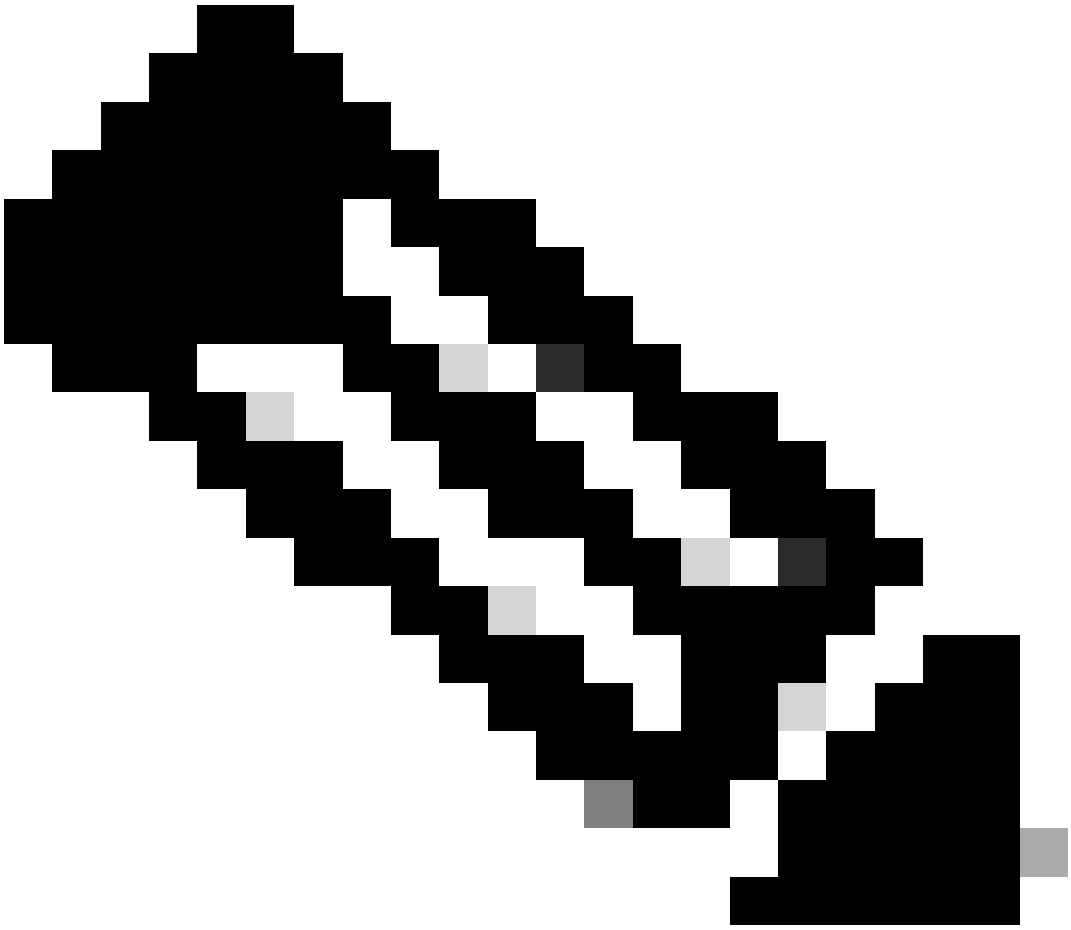
Cisco raccomanda la conoscenza e la familiarità con questi argomenti:

- Cisco IOS® e Cisco IOS® XE EEM
- Hosting di applicazioni e shell guest
- Script Python
- Comandi Linux

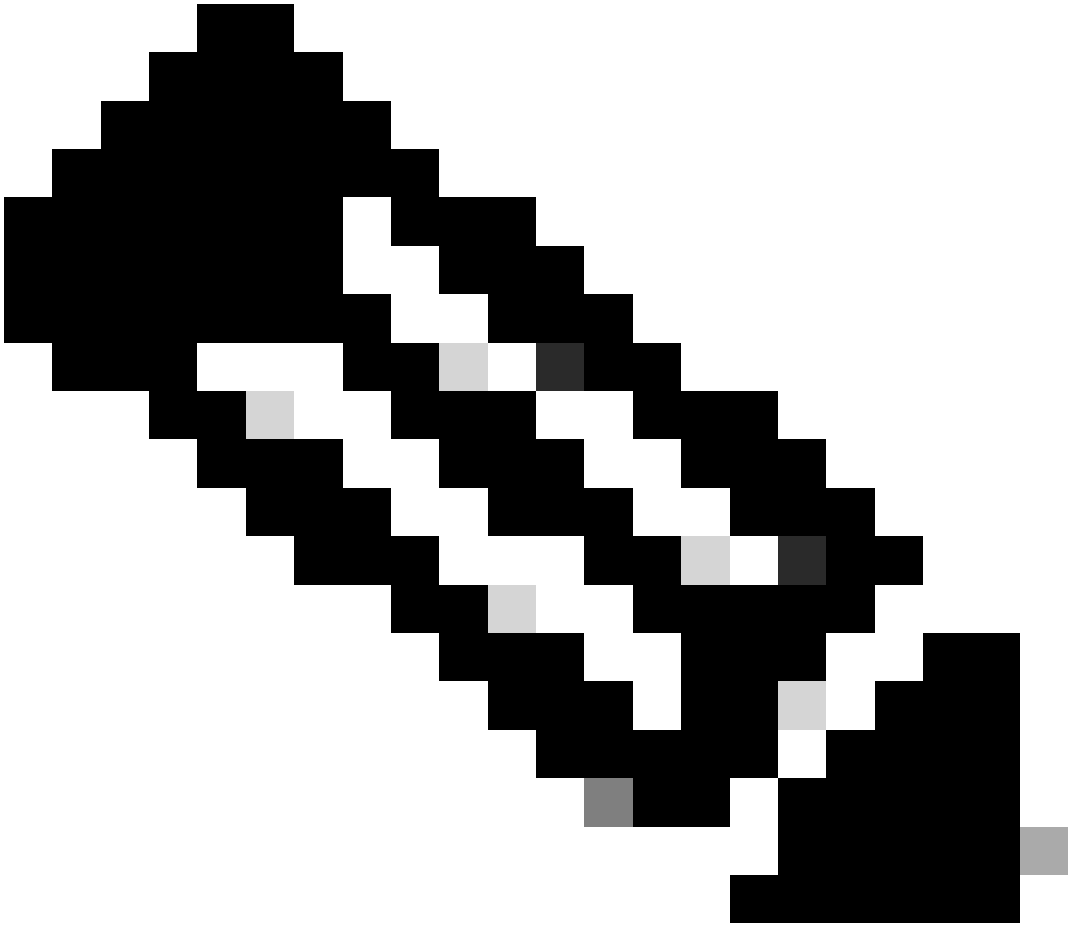
Componenti usati

Le informazioni fornite in questo documento si basano sulle seguenti versioni software e hardware:

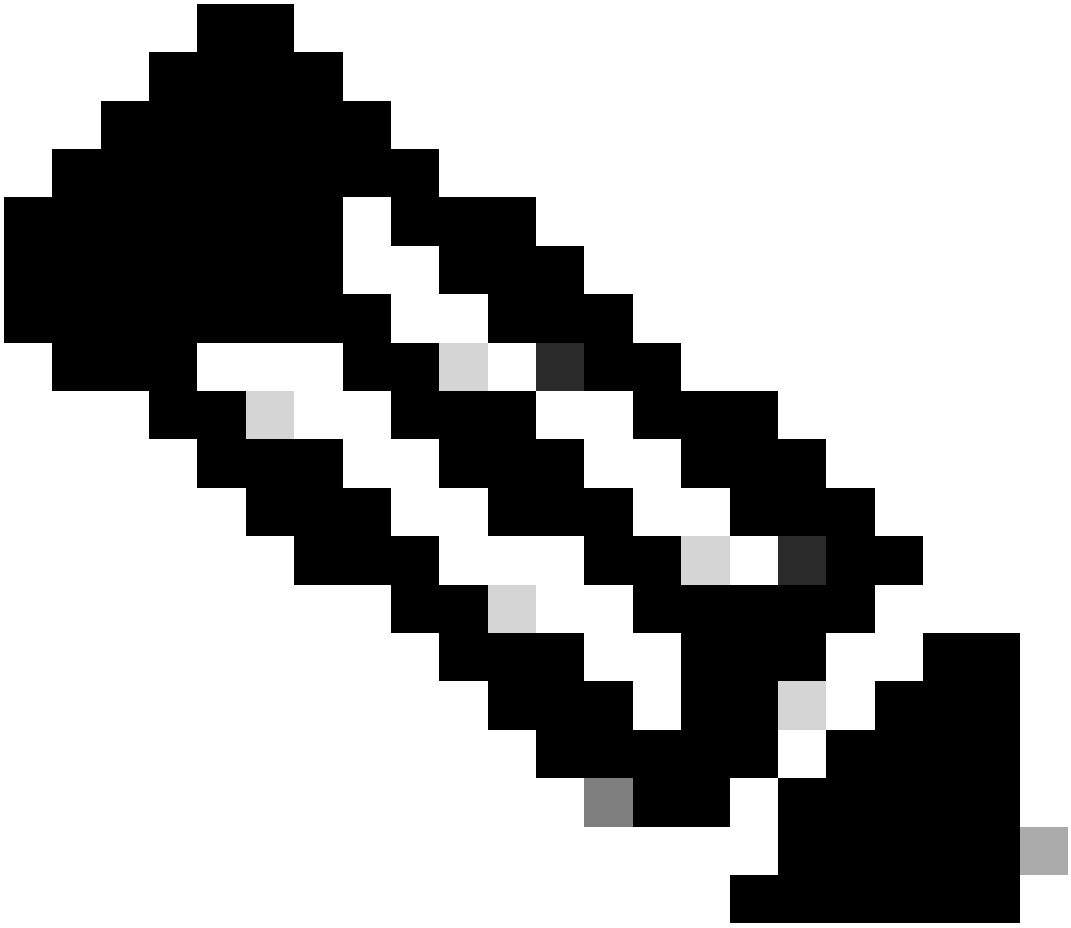
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1 e versioni successive



> Nota: per i comandi usati per attivare queste funzionalità su altre piattaforme Cisco, consultare le relative guide alla configurazione.



Nota: Gli switch Catalyst 9200L non supportano Guest Shell.



Nota: Questi script non sono supportati da Cisco TAC e vengono forniti così come sono per scopi educativi.

Le informazioni discusse in questo documento fanno riferimento a dispositivi usati in uno specifico ambiente di emulazione. Su tutti i dispositivi menzionati nel documento la configurazione è stata ripristinata ai valori predefiniti. Se la rete è operativa, valutare attentamente eventuali conseguenze derivanti dall'uso dei comandi.

Convenzioni

Per ulteriori informazioni sulle convenzioni usate, consultare il documento Cisco sulle convenzioni nei suggerimenti tecnici.

Premesse

Shell guest e script Python

L'hosting delle applicazioni sulla famiglia di switch Cisco Catalyst 9000 offre opportunità di innovazione a partner e sviluppatori, poiché i dispositivi di rete possono essere uniti a un ambiente di runtime dell'applicazione.

Supporta applicazioni containerizzate, fornendo un isolamento completo dal sistema operativo principale e dal kernel Cisco IOS XE. Questa separazione garantisce che le allocazioni di risorse per le applicazioni ospitate siano distinte dalle funzioni principali di routing e switching.

L'infrastruttura di hosting delle applicazioni per i dispositivi Cisco IOS XE è nota come IOx (Cisco IOS + Linux), che facilita l'hosting di applicazioni e servizi sviluppati da Cisco, partner e sviluppatori di terze parti sui dispositivi di rete, garantendo una perfetta integrazione tra diverse piattaforme hardware.

Guest Shell, una distribuzione di contenitori specializzati, illustra un'applicazione vantaggiosa per la distribuzione del sistema.

Guest Shell offre un ambiente virtualizzato basato su Linux progettato per eseguire applicazioni Linux personalizzate, incluso Python, per consentire il controllo e la gestione automatizzati dei dispositivi Cisco. Il contenitore della shell guest consente agli utenti di eseguire script e app nel sistema. In particolare, sulle piattaforme Intel x86, il contenitore Guest Shell è un contenitore Linux (LXC) con un file system radice CentOS 8.0 minimo. In Cisco IOS XE Amsterdam 17.3.1 e versioni successive, è supportato solo Python V3.6. È possibile installare ulteriori librerie Python durante il runtime utilizzando l'utilità Yum di CentOS 8.0. I pacchetti Python possono inoltre essere installati o aggiornati utilizzando Pip Install Packages (PIP).

Guest Shell include un'API (Application Programming Interface) Python, che consente di eseguire i comandi Cisco IOS XE utilizzando il modulo Python CLI. In questo modo, gli script Python migliorano le funzionalità di automazione, fornendo ai tecnici di rete uno strumento versatile per sviluppare script per automatizzare le attività di configurazione e raccolta dei dati. Questi script possono essere eseguiti manualmente dalla CLI, ma possono anche essere utilizzati insieme agli script EEM per rispondere a eventi specifici, come messaggi syslog, eventi di interfaccia o esecuzioni di comandi. Praticamente, qualsiasi evento che possa attivare uno script EEM può essere utilizzato anche per attivare uno script Python, espandendo il potenziale di automazione negli switch Cisco Catalyst 9000.

Quando si installa Guest Shell, viene creata automaticamente una directory di condivisione guest nel file system flash. Si tratta del file system al quale è possibile accedere dagli script Python e dalla shell guest. Per garantire la corretta sincronizzazione quando si utilizza lo stack, mantenere questa cartella al di sotto dei 50 MB.

Vantaggi dell'utilizzo di Python con gli script EEM

- Python estende le funzionalità di automazione degli script EEM consentendo la gestione di logica complessa (come espressioni regolari, cicli e corrispondenze) all'interno dello script Python. Questa funzionalità offre l'opportunità di creare script EEM più potenti.
- In qualità di linguaggio di programmazione conosciuto, Python riduce le barriere all'ingresso per i tecnici di rete che desiderano automatizzare i dispositivi Cisco IOS XE. Offre inoltre

manutenibilità e leggibilità.

- Python fornisce anche funzionalità di gestione degli errori e una potente libreria standard.

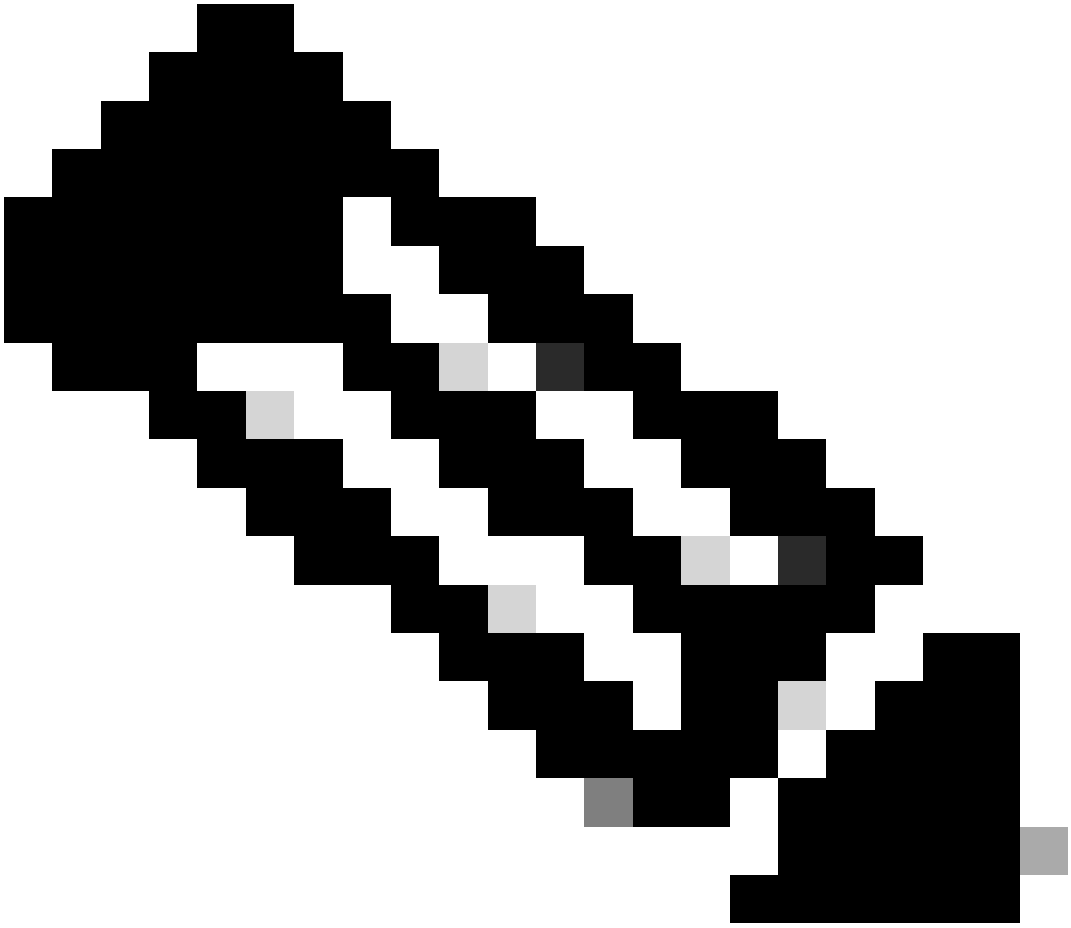
Considerazioni sull'utilizzo di Python con gli script EEM

- La shell guest non è abilitata per impostazione predefinita, pertanto deve essere abilitata prima che gli script Python possano essere eseguiti.
- gli script Python non possono essere creati direttamente nella CLI; devono essere sviluppate prima in un ambiente di sviluppo e quindi copiate nella memoria flash dello switch.

SELinux in Cisco IOS XE

A partire da Cisco IOS XE 17.8.1, è stato introdotto il supporto per Security-Enhanced Linux (SELinux) per applicare la sicurezza con policy che governano il modo in cui i processi, gli utenti e i file interagiscono tra loro. Il criterio SELinux definisce quali azioni e risorse possono essere utilizzate da un processo o da un utente. Una violazione può verificarsi quando un utente o un processo tenta di eseguire un'azione non consentita dal criterio, ad esempio l'accesso a una risorsa o l'esecuzione di un comando. SELinux può funzionare in due diverse modalità:

1. Modalità permissiva: SELinux non applica alcuna policy. Tuttavia, continua a registrare qualsiasi violazione come se fosse stata negata.
2. Applicazione: SELinux applica attivamente le policy di sicurezza sul sistema. Se un'azione viola la policy SELinux, l'azione viene negata e registrata.



Nota: La modalità predefinita è stata impostata su permissiva quando è stata introdotta in Cisco IOS XE 17.8.1. Tuttavia, a partire dalla versione 17.14.1, SELinux è abilitato in modalità di imposizione.

Quando si utilizza la shell guest, l'accesso ad alcune risorse può essere negato quando si utilizza la modalità di imposizione. Se quando si tenta di eseguire un'azione utilizzando la shell guest o uno script Python viene generato un errore di tipo autorizzazione negata, simile al seguente log:

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

Per verificare se uno script viene rifiutato da SELinux, utilizzare il comando`show platform software audit summary`per verificare se il numero di negazioni è in aumento. Inoltre,`show platform software audit all`visualizza un registro delle azioni bloccate da SELinux. Access Vector Cache (AVC) è il meccanismo utilizzato per registrare le decisioni di controllo dell'accesso in SELinux, quindi

quando si utilizza questo comando, cercare i registri che iniziano con type=AVC.

Se uno script viene negato e bloccato, è possibile impostare SELinux in modalità permissiva utilizzando il comando`set platform software selinux permissive`. Questa modifica non viene memorizzata nella configurazione di esecuzione o di avvio, quindi dopo un ricaricamento, la modalità torna a essere applicata. Pertanto, ogni volta che lo switch viene ricaricato, questa modifica deve essere riapplicata. La modifica può essere verificata utilizzando`show platform software selinux`.

Configurazione

Per automatizzare le operazioni sullo switch utilizzando gli script EEM e Python, attenersi alla seguente procedura:

1. Abilitare la shell guest.
2. Copiare lo script Python nella directory /flash/guest-share/. È possibile utilizzare qualsiasi meccanismo di copia disponibile in Cisco IOS XE, ad esempio SCP, FTP o File Manager in WebUI. Una volta che lo script Python si trova nella memoria flash, è possibile eseguirlo utilizzando il comando`guestshell run python3 /flash/guest-share/cat9k_script.py`.
3. Configurare uno script EEM che esegua lo script Python. Questa configurazione consente di attivare lo script Python utilizzando uno dei più rilevatori di eventi forniti dagli script EEM, come un messaggio syslog, un pattern CLI e un Cron scheduler.

In questa sezione viene spiegato il passo 1. Nella sezione successiva vengono forniti esempi che illustrano come implementare i passaggi 2 e 3.

Abilitare la shell guest con un indirizzo IP statico

Per abilitare la shell guest, attenersi alla procedura descritta di seguito.

1. Abilitare 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. Configurare la rete di hosting applicazioni per la shell guest. In questo esempio viene utilizzata l'interfaccia AppGigabit Ethernet per fornire l'accesso alla rete; tuttavia, è possibile utilizzare anche l'interfaccia di gestione (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. Abilitare la shell guest.

<#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. Convalidare la shell guest. In questo esempio viene verificato che sia possibile raggiungere il gateway predefinito e il percorso cisco.com. Verificare inoltre che Python 3 possa essere eseguito dalla shell guest.

<#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#
```

Abilitare la shell guest con un indirizzo IP DHCP

In genere, la shell guest è configurata con un indirizzo IP statico perché per impostazione predefinita il contenitore della shell guest non dispone del servizio client DHCP. Se è necessario che la shell guest richieda dinamicamente un indirizzo IP, è necessario installare il servizio client DHCP. Attenersi alla procedura seguente:

1. Seguire la procedura per abilitare la shell guest con un indirizzo IP statico. Tuttavia, questa volta, non assegnare l'indirizzo IP nella configurazione di hosting dell'app durante il passaggio 2. Usare invece questa configurazione:

```
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. Il client DHCP può essere installato usando l'utility Yum con il comando`sudo yum install dhcp-client`. Tuttavia, i repository per CentOS Stream 8 sono stati smantellati. Per risolvere questo problema, i pacchetti client DHCP possono essere scaricati e installati manualmente. Su un PC, scaricare questi pacchetti dall'archivio CentOS Stream 8 e inserirli in un file 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. Copiare il file nelladhcp-client.tardirectory /flash/guest-share/ nello switch.
4. Immettere una sessione Guest Shell bash ed eseguire i comandi Linux per installare il client DHCP e richiedere un indirizzo 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
```

Scenari d'uso

Caso di utilizzo 1: Salvataggio automatico delle modifiche alla configurazione su un server SCP

In alcune situazioni, è utile salvare automaticamente le configurazioni dello switch su un server ogni volta che viene utilizzato il`write memory`comando. Questa procedura consente di tenere traccia delle modifiche e di eseguire il rollback della configurazione, se necessario. Quando si sceglie un server, è possibile utilizzare sia TFTP che SCP, ma un server SCP offre un ulteriore livello di sicurezza.

Questa funzionalità è disponibile nell'archivio di Cisco IOS. Tuttavia, uno svantaggio significativo è che le credenziali SCP non possono essere nascoste nella configurazione; il percorso del server viene visualizzato in formato testo normale sia nella configurazione di esecuzione che in quella di avvio.

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

Utilizzando Guest Shell e Python, è possibile ottenere la stessa funzionalità mantenendo nascoste le credenziali. A tale scopo, utilizzare le variabili di ambiente all'interno della shell guest per archiviare le credenziali SCP effettive. Di conseguenza, le credenziali del server SCP non sono visibili nella configurazione in esecuzione.

In questo approccio, la configurazione in esecuzione visualizza solo lo script EEM, mentre gli script Python costruiscono il comando con le credenziali e lo passano allo script EEM da eseguire`copy running-config scp:`.

Per questo esempio, attenersi alla procedura seguente:

1. Copiare lo script Python nella directory /flash/guest-share. Questo script legge le variabili di ambiente SCP_USER e SCP_PASSWORD e restituisce il`copy startup-config scp:`comando in modo che lo script EEM possa eseguirlo. Lo script richiede l'indirizzo IP del server SCP come argomento. Inoltre, lo script conserva un registro di ogni volta che il comando`write memory`viene eseguito in un file persistente situato in /flash/guest-share/TAC-write-memory-log.txt. Questo è lo 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
```

Nell'esempio, lo script Python viene copiato sullo switch utilizzando un server TFTP:

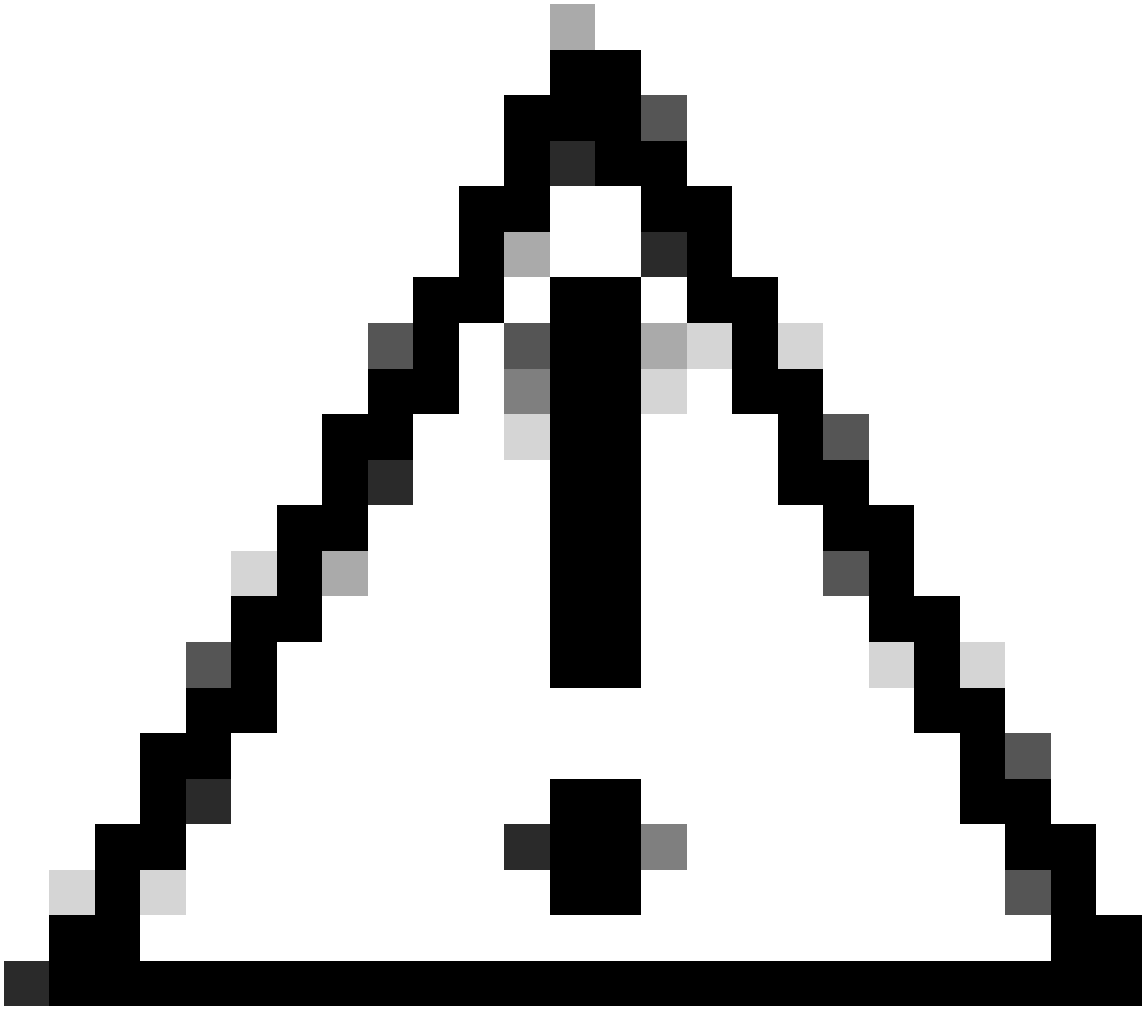
```
<#root>

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. Installare lo script EEM. Questo script chiama lo script Python. che restituisce il comando`copy startup-config scp:`necessario per salvare la configurazione sul server SCP. Lo script EEM esegue quindi il comando restituito dallo 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. Impostare le variabili di ambiente della shell guest inserendole nel`~/.bashrc`file. In questo modo, le variabili di ambiente vengono mantenute ogni volta che si apre una shell guest, anche dopo il ricaricamento dello switch. Aggiungere le due righe seguenti:

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



Attenzione: Le credenziali utilizzate in questo esempio hanno uno scopo puramente educativo. Non sono destinati all'utilizzo in ambienti di produzione. Gli utenti devono sostituire queste credenziali con le proprie credenziali sicure specifiche dell'ambiente.

Questo è il processo per aggiungere queste variabili al~/.bashrcfile:

<#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. Eseguire lo script Python manualmente per verificare che restituisca il comando correttocopye che registri l'operazione nel file persistenteTAC-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. Eseguire il test dello script EEM. Lo script EEM invia inoltre un syslog con il risultato dell'operazione di copia, indipendentemente dal fatto che questa sia stata eseguita correttamente o meno. Di seguito è riportato un esempio di esecuzione riuscita:

```
<#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.
```

Per verificare un trasferimento non riuscito, il server SCP viene arrestato. Questo è il risultato di questa esecuzione non riuscita:

```
<#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.
```

Caso di utilizzo 2: Monitoraggio degli incrementi delle modifiche alla topologia STP

Questo esempio è utile per monitorare i problemi relativi all'instabilità dello Spanning Tree e identificare l'interfaccia che riceve le notifiche di modifica della topologia (TCN). Lo script EEM

viene eseguito periodicamente a un intervallo di tempo specificato e chiama uno script Python che esegue un comando show e verifica se i nomi TCN sono aumentati.

La creazione di questo script utilizzando solo i comandi EEM richiederebbe l'utilizzo di per i loop e più corrispondenze regex, il che sarebbe complicato. Di conseguenza, in questo esempio viene mostrato come lo script EEM deleghi questa logica complessa a Python.

Per questo esempio, attenersi alla procedura seguente:

1. Copiare lo script Python nella directory /flash/guest-share/. Questo script esegue le seguenti attività:

    1. Esegue il comando`show spanning-tree detail`e analizza l'output per salvare le informazioni TCN per ciascuna VLAN in un dizionario.

    2. Confronta le informazioni TCN analizzate con i dati nel file JSON dell'esecuzione dello script precedente. Per ciascuna VLAN, se i TCN sono aumentati, viene inviato un messaggio syslog con informazioni simili a quelle dell'esempio seguente:

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

    3. Le informazioni TCN correnti vengono salvate in un file JSON per essere confrontate durante l'esecuzione successiva. Questo è lo 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()
```

Nell'esempio, lo script Python viene copiato sullo switch utilizzando un server 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. Installare lo script EEM. In questo esempio, l'unica attività dello script EEM è eseguire lo script Python, che invia un messaggio di log se viene rilevato un incremento TCN. In questo esempio, lo script EEM viene eseguito ogni 5 minuti.

```
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. Per convalidare la funzionalità dello script, è possibile eseguire lo script Python manualmente o attendere cinque minuti prima che lo script EEM lo chiami. In entrambi i casi, viene inviato un syslog solo se il numero di TCN è aumentato per una 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. Verificare lo script EEM attendendone l'esecuzione ogni cinque minuti. Se i TCN sono aumentati per una VLAN, viene inviato un messaggio syslog. In questo particolare esempio, si nota che i TCN sono in costante aumento sulla VLAN 30 e l'interfaccia che riceve questi TCN costanti è 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.
```

Informazioni correlate

- Guida alla configurazione della programmabilità, Cisco IOS XE Cupertino 17.9.x (capitolo: Guest Shell
- Comprendere le procedure ottimali e gli script utili per EEM
- White paper sull'hosting delle applicazioni sugli switch Cisco Catalyst serie 9000

Informazioni su questa traduzione

Cisco ha tradotto questo documento utilizzando una combinazione di tecnologie automatiche e umane per offrire ai nostri utenti in tutto il mondo contenuti di supporto nella propria lingua. Si noti che anche la migliore traduzione automatica non sarà mai accurata come quella fornita da un traduttore professionista. Cisco Systems, Inc. non si assume alcuna responsabilità per l’accuratezza di queste traduzioni e consiglia di consultare sempre il documento originale in inglese (disponibile al link fornito).