Automatización de switches Catalyst 9000 mediante scripts Python

Contenido

Introducción

Este documento describe cómo extender EEM con scripts Python para automatizar la configuración y la recolección de datos en switches Catalyst 9000.

Prerequisites

Requirements

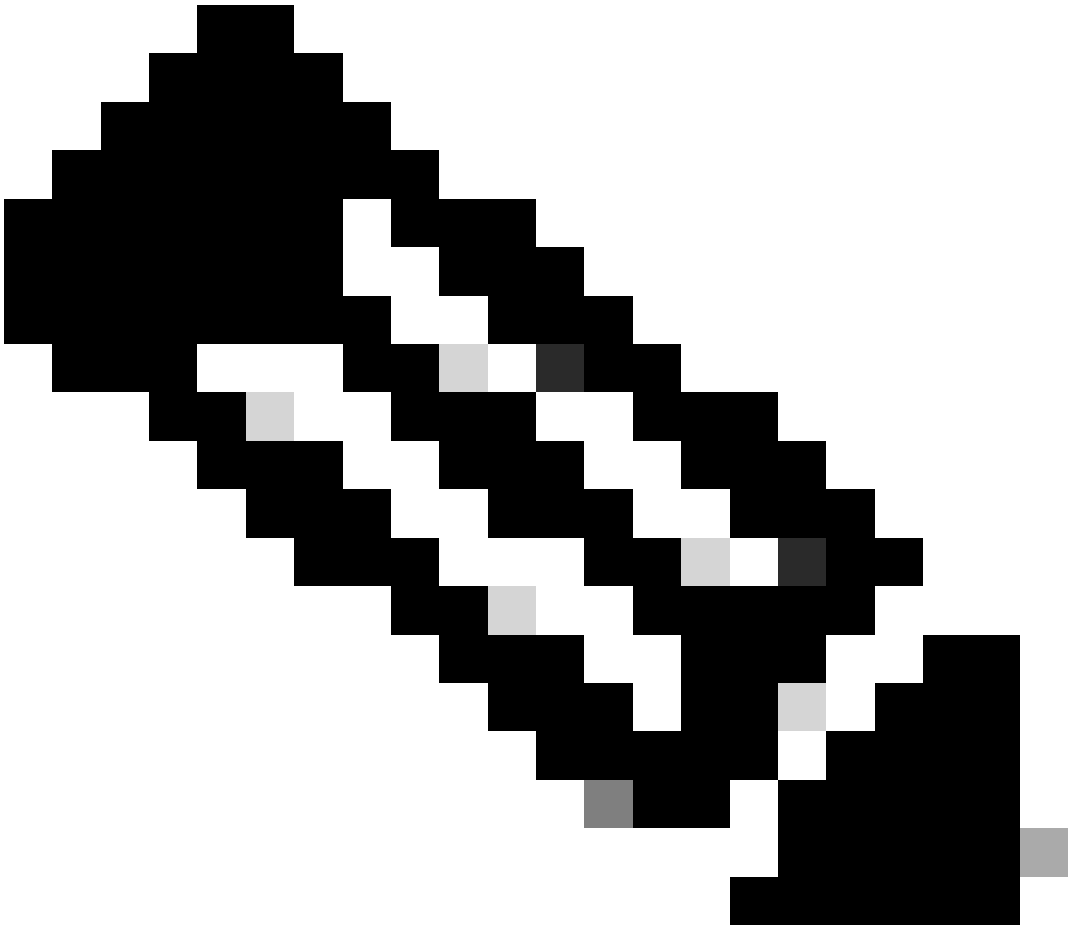
Cisco recomienda que conozca y esté familiarizado con los siguientes temas:

- Cisco IOS® y Cisco IOS® XE EEM
- Alojamiento de aplicaciones y shell de invitado
- scripting con Python
- Comandos de Linux

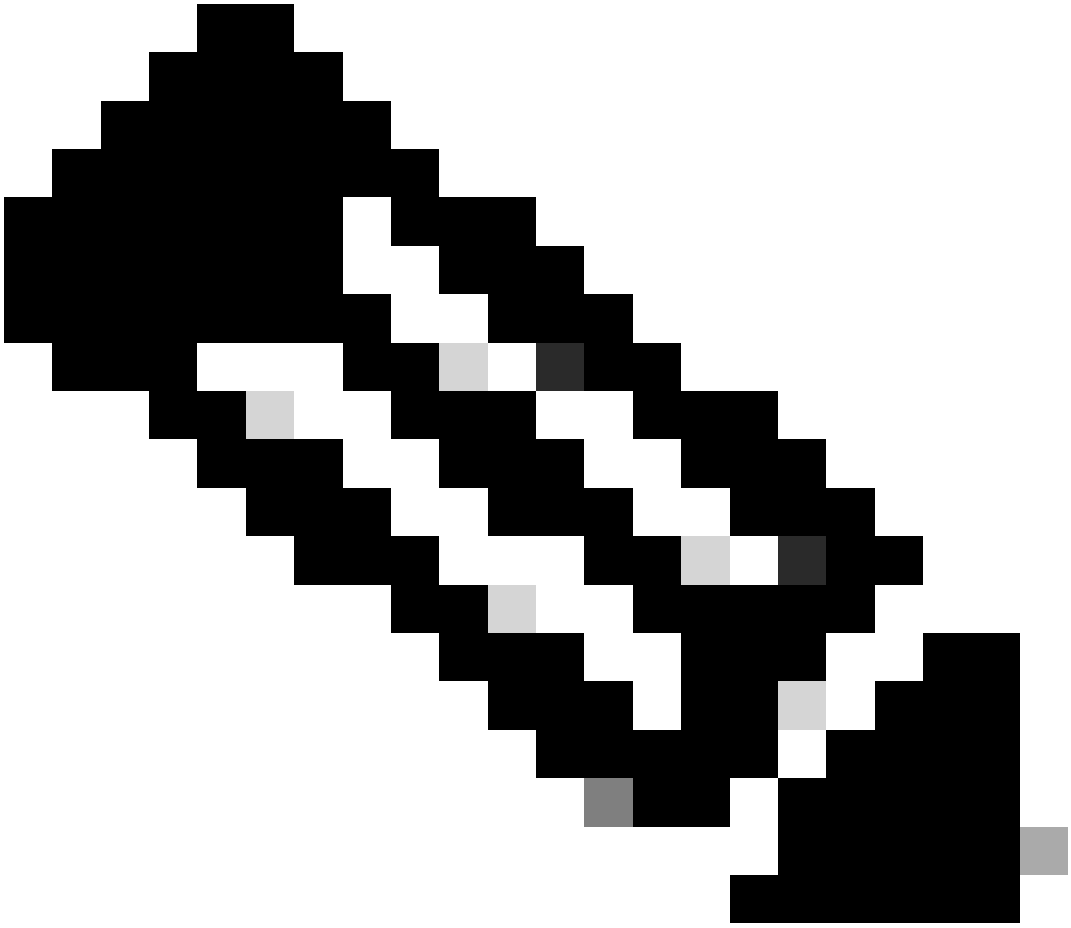
Componentes Utilizados

La información que contiene este documento se basa en las siguientes versiones de software y hardware.

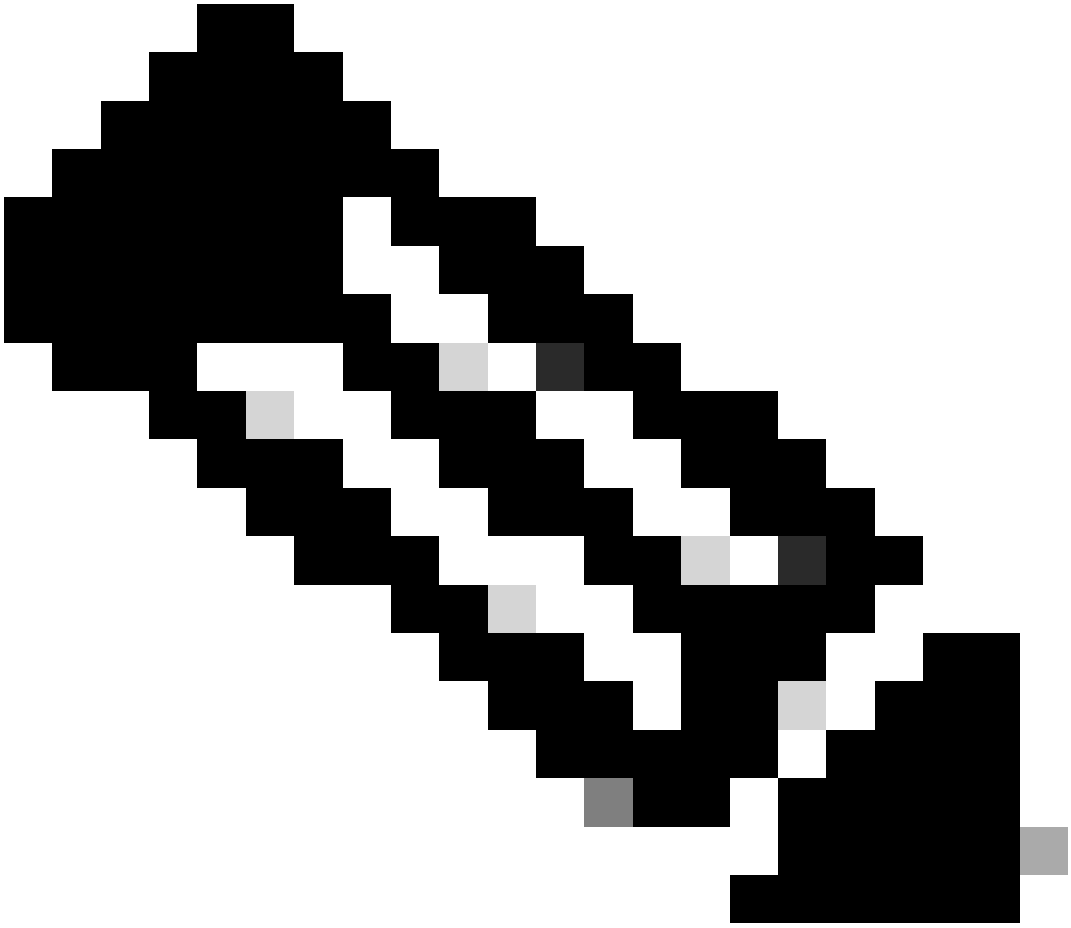
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1 y versiones posteriores



> Nota: Consulte la guía de configuración correspondiente para conocer los comandos que se utilizan para activar estas funciones en otras plataformas de Cisco.



Nota: Los switches Catalyst 9200L no soportan Guest Shell.



Nota: Estos scripts no son compatibles con Cisco TAC y se proporcionan tal cual con fines educativos.

La información que contiene este documento se creó a partir de los dispositivos en un ambiente de laboratorio específico. Todos los dispositivos que se utilizan en este documento se pusieron en funcionamiento con una configuración verificada (predeterminada). Si tiene una red en vivo, asegúrese de entender el posible impacto de cualquier comando.

Convenciones

Consulte Convenciones de Consejos Técnicos de Cisco para obtener información sobre las convenciones sobre documentos.

Antecedentes

Shell de invitado y scripts Python

El alojamiento de aplicaciones en la familia de switches Cisco Catalyst 9000 ofrece oportunidades de innovación para partners y desarrolladores, ya que los dispositivos de red se pueden combinar con un entorno de tiempo de ejecución de aplicaciones.

Admite aplicaciones en contenedores, lo que proporciona un aislamiento completo del sistema operativo principal y del núcleo de Cisco IOS XE. Esta separación garantiza que las asignaciones de recursos para aplicaciones alojadas sean distintas de las funciones de routing y switching principales.

La infraestructura de alojamiento de aplicaciones para dispositivos Cisco IOS XE se conoce como IOSx (Cisco IOS + Linux), que facilita el alojamiento de aplicaciones y servicios desarrollados por Cisco, partners y desarrolladores de terceros en dispositivos de red, lo que garantiza una integración perfecta entre diversas plataformas de hardware.

El shell de invitado, una implementación de contenedor especializada, ejemplifica una aplicación beneficiosa para la implementación del sistema.

Guest Shell ofrece un entorno virtualizado basado en Linux diseñado para ejecutar aplicaciones de Linux personalizadas, incluida Python, para permitir el control y la gestión automatizados de los dispositivos de Cisco. El contenedor de shell de invitado permite a los usuarios ejecutar scripts y aplicaciones en el sistema. Específicamente, en las plataformas Intel x86, el contenedor Guest Shell es un contenedor Linux (LXC) con un sistema de archivos root mínimo CentOS 8.0. En Cisco IOS XE Amsterdam 17.3.1 y versiones posteriores, solo se admite Python V3.6. Las bibliotecas adicionales de Python se pueden instalar durante el tiempo de ejecución mediante la utilidad Yum en CentOS 8.0. Los paquetes de Python también se pueden instalar o actualizar mediante Pip Install Packages (PIP).

Guest Shell incluye una interfaz de programación de aplicaciones (API) de Python, que permite ejecutar comandos de Cisco IOS XE mediante el módulo de la CLI de Python. De esta manera, los scripts Python mejoran las capacidades de automatización, proporcionando a los ingenieros de red una herramienta versátil para desarrollar scripts para automatizar las tareas de configuración y recopilación de datos. Aunque estas secuencias de comandos se pueden ejecutar manualmente a través de la CLI, también se pueden emplear junto con las secuencias de comandos de EEM para responder a eventos específicos, como mensajes de syslog, eventos de interfaz o ejecuciones de comandos. Prácticamente, cualquier evento que pueda activar un script EEM también se puede utilizar para activar un script Python, lo que aumenta el potencial de automatización de los switches Cisco Catalyst 9000.

Cuando se instala el shell de invitado, se crea automáticamente un directorio compartido de invitado en el sistema de archivos flash. Este es el sistema de archivos al que se puede acceder desde los scripts Python y el shell de invitado. Para garantizar una sincronización correcta al utilizar el apilamiento, mantenga esta carpeta por debajo de 50 MB.

Ventajas de usar Python con scripts EEM

- Python amplía las capacidades de automatización de los scripts EEM al permitir que la lógica compleja (como expresiones regulares, bucles y coincidencias) se administre dentro del script Python. Esta capacidad ofrece la oportunidad de crear scripts EEM más potentes.

- Como lenguaje de programación conocido, Python reduce la barrera de entrada para los ingenieros de redes que deseen automatizar los dispositivos Cisco IOS XE. Además, ofrece mantenimiento y legibilidad.
- Python también proporciona capacidades de gestión de errores, así como una potente biblioteca estándar.

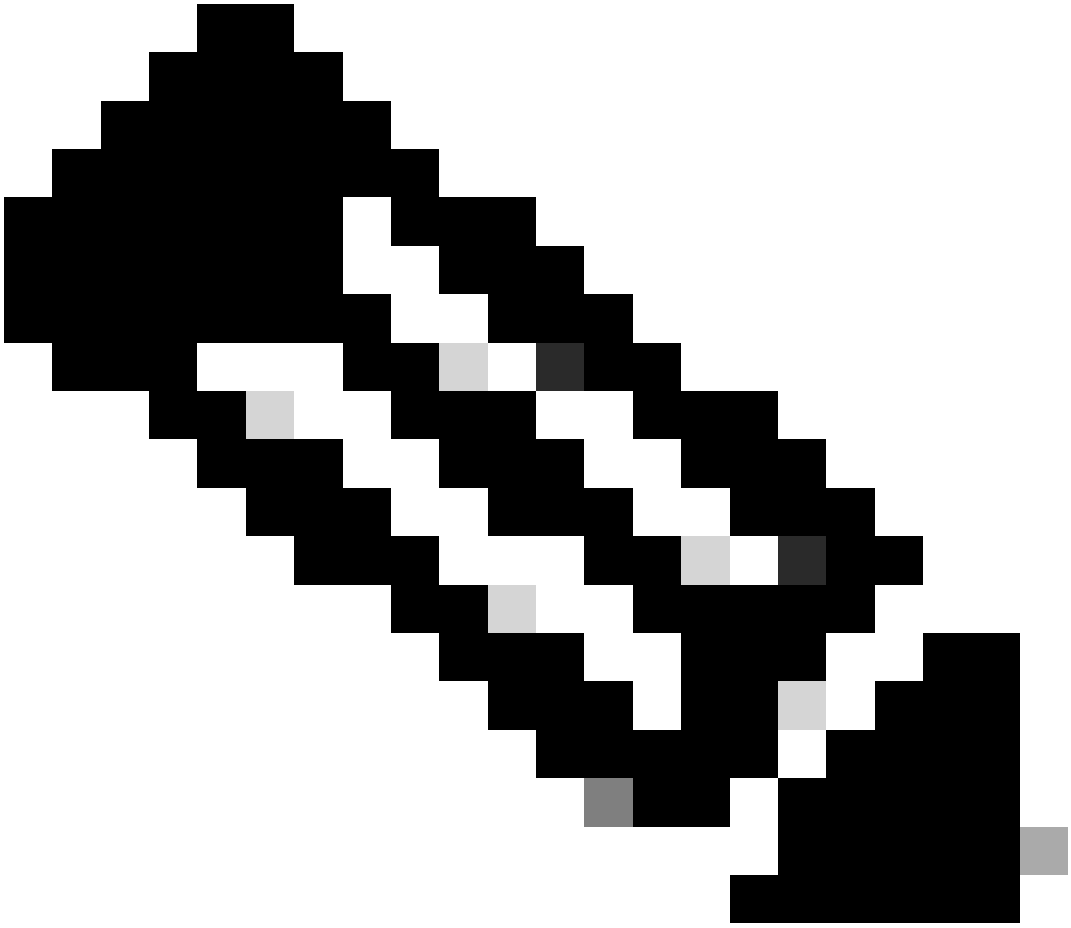
Consideraciones sobre el uso de Python con scripts EEM

- El shell de invitado no está habilitado de forma predeterminada, por lo que debe habilitarse antes de que se puedan ejecutar los scripts Python.
- Las secuencias de comandos de Python no se pueden crear directamente en la CLI; primero deben desarrollarse en un entorno de desarrollo y luego copiarse en la memoria flash del switch.

SELinux en Cisco IOS XE

A partir de Cisco IOS XE 17.8.1, se introdujo la compatibilidad con Linux con seguridad mejorada (SELinux) para aplicar la seguridad con políticas que rigen la forma en que los procesos, los usuarios y los archivos interactúan entre sí. La política de SELinux define qué acciones y recursos están permitidos para ser accedidos por un proceso o usuario. Una infracción puede producirse cuando un usuario o un proceso intenta realizar una acción no permitida por la directiva, por ejemplo, obtener acceso a un recurso o ejecutar un comando. SELinux puede operar en 2 modos diferentes:

1. Modo permisivo: SELinux no aplica ninguna política. Sin embargo, sigue registrando las infracciones como si se hubieran denegado.
2. Aplicación: SELinux aplica activamente las políticas de seguridad en el sistema. Si una acción viola la política de SELinux, la acción es denegada y registrada.



Nota: El modo predeterminado se configuró en permisivo cuando se introdujo en Cisco IOS XE 17.8.1. Sin embargo, a partir de la versión 17.14.1, SELinux está habilitado en el modo de aplicación.

Cuando se utiliza Guest Shell, se puede denegar el acceso a algunos recursos cuando se utiliza el modo de aplicación. Si al intentar realizar una acción mediante el shell de invitado o un script Python se produce un error de permiso denegado, similar a este registro:

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

Para verificar si SELinux está negando un script, utilice el comando`show platform software audit summary`para verificar si los recuentos de negación están aumentando. Además,`show platform software audit all`muestra un registro de las acciones bloqueadas por SELinux. El Access Vector Cache (AVC) es el mecanismo utilizado para registrar las decisiones de control de acceso en SELinux, por lo que

al utilizar este comando, busque los registros que comiencen con type=AVC.

Si un script está siendo denegado y bloqueado, SELinux puede configurarse en modo permisivo usando el comando`set platform software selinux permissive`. Este cambio no se almacena en la configuración de ejecución o de inicio, por lo que después de una recarga, el modo vuelve a aplicarse. Por lo tanto, cada vez que el switch se recarga, este cambio debe ser reaplicado. El cambio se puede comprobar mediante`show platform software selinux`.

Configurar

Para automatizar tareas en el switch mediante scripts EEM y Python, siga estos pasos:

1. Habilite el shell de invitado.
2. Copie el script Python en el directorio /flash/guest-share/. Puede utilizar cualquier mecanismo de copia disponible en Cisco IOS XE, como SCP, FTP o el administrador de archivos de la interfaz de usuario web. Una vez que el script Python está en la memoria flash, puede ejecutarlo usando el comando`guestshell run python3 /flash/guest-share/cat9k_script.py`.
3. Configure un script EEM que ejecute el script Python. Esta configuración permite que el script Python se active utilizando cualquiera de los múltiples detectores de eventos proporcionados por los scripts EEM, como un mensaje syslog, un patrón CLI y un planificador Cron.

En esta sección, se explica el paso 1. En la siguiente sección se proporcionan ejemplos que muestran cómo implementar los pasos 2 y 3.

Habilite el shell de invitado con una dirección IP estática

Siga este proceso para habilitar el shell de invitado:

1. Habilite 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. Configure la red de alojamiento de aplicaciones para el shell de invitado. Este ejemplo utiliza la interfaz AppGigabitEthernet para proporcionar acceso a la red; sin embargo, también se puede utilizar la interfaz de administración (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. Habilite el shell de invitado.

<#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. Valide el shell de invitado. En este ejemplo se valida la disponibilidad con el gateway predeterminado, así como con cisco.com. Además, valide que Python 3 se pueda ejecutar desde el shell de invitado.

<#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#
```

Habilite el shell de invitado con una dirección IP DHCP

Normalmente, el shell de invitado se configura con una dirección IP estática porque el contenedor del shell de invitado no tiene el servicio de cliente DHCP de forma predeterminada. Si es necesario que el shell de invitado solicite una dirección IP dinámicamente, debe instalarse el servicio de cliente DHCP. Siga este proceso:

1. Siga los pasos para habilitar el shell de invitado con una dirección IP estática. Sin embargo, esta vez, no asigne la dirección IP en la configuración de alojamiento de aplicaciones durante el paso 2. En su lugar, utilice esta configuración:

```
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. El cliente DHCP se puede instalar utilizando la utilidad Yum con el comando`sudo yum install dhcp-client`. Sin embargo, los repositorios para CentOS Stream 8 han sido desmantelados. Para solucionar este problema, los paquetes del cliente DHCP se pueden descargar e instalar manualmente. En una PC, descargue estos paquetes de la bóveda CentOS Stream 8 y empaquételos en un archivo .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. Copie el dhcp-client.tar archivo en el directorio /flash/guest-share/ del switch.
4. Ingrese una sesión bash de shell de invitado y ejecute los comandos Linux para instalar el cliente DHCP y solicitar una dirección 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
```

Casos de uso

Caso práctico 1: Almacenamiento automático de cambios de configuración en un servidor SCP

En algunas situaciones, es conveniente guardar automáticamente las configuraciones del switch en un servidor cada vez que se utiliza el`write memory`comando. Esta práctica ayuda a mantener un registro de los cambios y permite la reversión de la configuración si es necesario. Al elegir un servidor, se pueden utilizar tanto TFTP como SCP, pero un servidor SCP ofrece una capa adicional de seguridad.

La función de archivo del IOS de Cisco proporciona esta funcionalidad. Sin embargo, un inconveniente importante es que las credenciales de SCP no se pueden ocultar en la configuración; la ruta de acceso del servidor se muestra en texto sin formato en las configuraciones de ejecución e inicio.

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

Mediante el uso de Guest Shell y Python, es posible lograr la misma funcionalidad mientras se mantienen ocultas las credenciales. Esto se hace aprovechando las variables de entorno del shell de invitado para almacenar las credenciales reales de SCP. Como resultado, las credenciales del servidor SCP no están visibles en la configuración en ejecución.

En este enfoque, la configuración en ejecución solo muestra el script EEM, mientras que los scripts Python construyen el`copy running-config scp:`comando con las credenciales y lo pasan al script EEM que se ejecutará.

Siga estos pasos para este ejemplo:

1. Copie el script Python en el directorio /flash/guest-share. Este script lee las variables de entorno SCP_USER y SCP_PASSWORD, y devuelve el`copy startup-config scp:`comando, de modo que el script EEM pueda ejecutarlo. La secuencia de comandos requiere la dirección IP del servidor SCP como argumento. Además, la secuencia de comandos mantiene un registro de cada vez que se ejecuta el comando`write memory`en un archivo persistente ubicado en /flash/guest-share/TAC-write-memory-log.txt. Este es el guion de 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
```

En este ejemplo, el script Python se copia en el switch mediante un servidor TFTP:

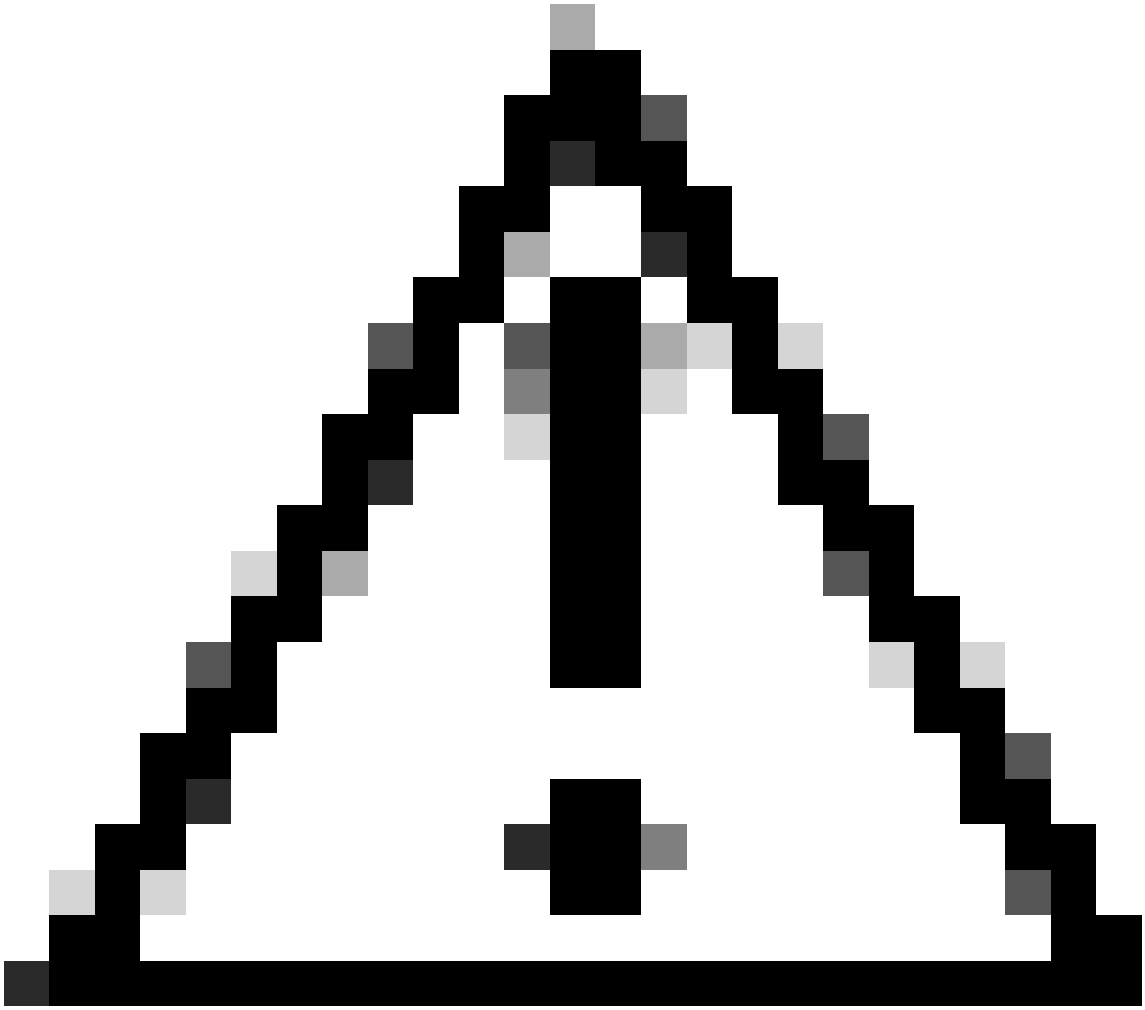
```
<#root>

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. Instale el script EEM. Este script llama al script Python. que devuelve el`copy startup-config scp:`comando necesario para guardar la configuración en el servidor SCP. La secuencia de comandos EEM ejecuta el comando devuelto por la secuencia de comandos 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. Establezca las variables de entorno del shell de invitado insertándolas en el`~/.bashrc`archivo. Esto garantiza que las variables de entorno sean persistentes cada vez que se abre un shell de invitado, incluso después de que se recarga el switch. Agregue estas dos líneas:

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



Precaución: Las credenciales utilizadas en este ejemplo son solo para fines educativos. No están pensados para su uso en entornos de producción. Los usuarios deben reemplazar estas credenciales por sus propias credenciales seguras específicas del entorno.

Este es el proceso para agregar estas variables al~/.bashrcarchivo:

<#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. Ejecute el script Python manualmente para validar que devuelve el comando `copy` correcto y registra la operación en el archivo persistente `TAC-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. Pruebe el script EEM. Esta secuencia de comandos de EEM también envía un registro del sistema con el resultado de la operación de copia, tanto si se realiza correctamente como si no. A continuación se muestra un ejemplo de una ejecución exitosa:

```
<#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.
```

Para probar una transferencia fallida, el servidor SCP se apaga. Este es el resultado de esta ejecución fallida:

```
<#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 práctico 2: Monitorear los incrementos en los cambios de topología STP

Este ejemplo es útil para supervisar los problemas relacionados con la inestabilidad del árbol de extensión e identificar qué interfaz recibe las notificaciones de cambio de topología (TCN). El

script EEM se ejecuta periódicamente en un intervalo de tiempo especificado y llama a un script Python que ejecuta un comando show y verifica si las TCN han aumentado.

La creación de este script usando solo comandos EEM requeriría el uso de bucles for y múltiples coincidencias regex, lo que sería engorroso. Por lo tanto, este ejemplo muestra cómo el script EEM delega esta lógica compleja a Python.

Siga estos pasos para este ejemplo:

1. Copie el script Python en el directorio /flash/guest-share/. Esta secuencia de comandos realiza estas tareas:

    1. Ejecuta el`show spanning-tree detail`comando y analiza el resultado para guardar la información de TCN para cada VLAN en un diccionario.

    2. Compara la información de TCN analizada con los datos del archivo JSON de la ejecución anterior del script. Para cada VLAN, si las TCN han aumentado, se envía un mensaje de syslog con información similar a este ejemplo:

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

    3. Guarda la información de TCN actual en un archivo JSON para compararla durante la siguiente ejecución. Este es el guion de 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()
```

En este ejemplo, el script Python se copia en el switch mediante un servidor 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. Instale el script EEM. En este ejemplo, la única tarea del script EEM es ejecutar el script Python, que envía un mensaje de registro si se detecta un incremento TCN. En este ejemplo, el script EEM se ejecuta cada 5 minutos.

```
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. Para validar la funcionalidad del script, puede ejecutar el script Python manualmente o esperar cinco minutos para que el script EEM lo llame. En ambos casos, se envía un syslog sólo si las TCN han aumentado para 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. Pruebe el script EEM esperando a que se ejecute cada cinco minutos. Si las TCN han aumentado para cualquier VLAN, se envía un mensaje de syslog. En este ejemplo en particular, se observa que las TCN aumentan constantemente en la VLAN 30, y la interfaz que recibe estas TCN constantes es 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.
```

Información Relacionada

- Guía de configuración de capacidad de programación, Cisco IOS XE Cupertino 17.9.x (Capítulo: Shell de invitado)
- Comprender las prácticas recomendadas y los guiones útiles para EEM
- Informe técnico sobre el alojamiento de aplicaciones en switches Catalyst de Cisco serie 9000

Acerca de esta traducción
Cisco ha traducido este documento combinando la traducción automática y los recursos humanos a fin de ofrecer a nuestros usuarios en todo el mundo contenido en su propio idioma.

Tenga en cuenta que incluso la mejor traducción automática podría no ser tan precisa como la proporcionada por un traductor profesional.

Cisco Systems, Inc. no asume ninguna responsabilidad por la precisión de estas traducciones y recomienda remitirse siempre al documento original escrito en inglés (insertar vínculo URL).