Automatizar Switches Catalyst 9000 Usando Scripts Python

Contents

Introdução

Este documento descreve como estender o EEM com scripts Python para automatizar a configuração e a coleta de dados em switches Catalyst 9000.

Pré-requisitos

Requisitos

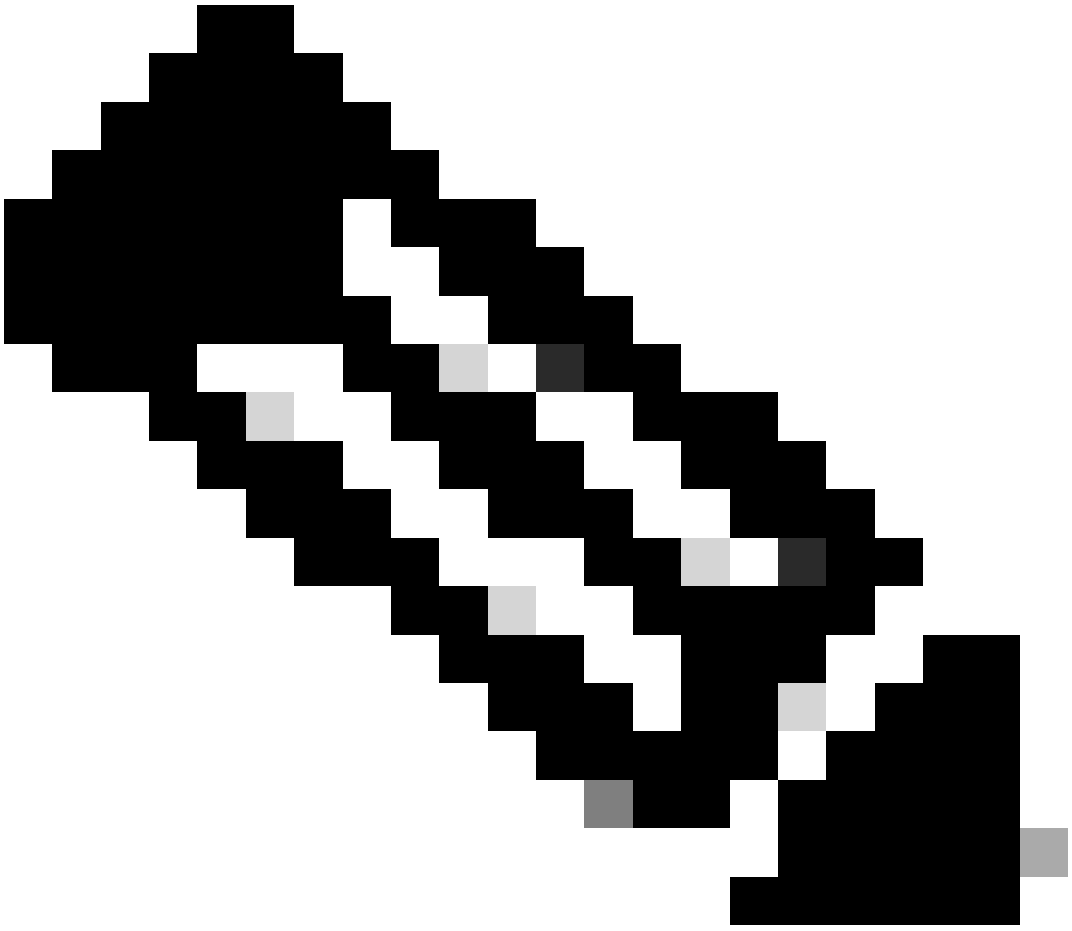
A Cisco recomenda que você tenha conhecimento e familiaridade com estes tópicos:

- Cisco IOS® e Cisco IOS® XE EEM
- Hospedagem de aplicativos e shell de convidado
- Script Python
- Comandos do Linux

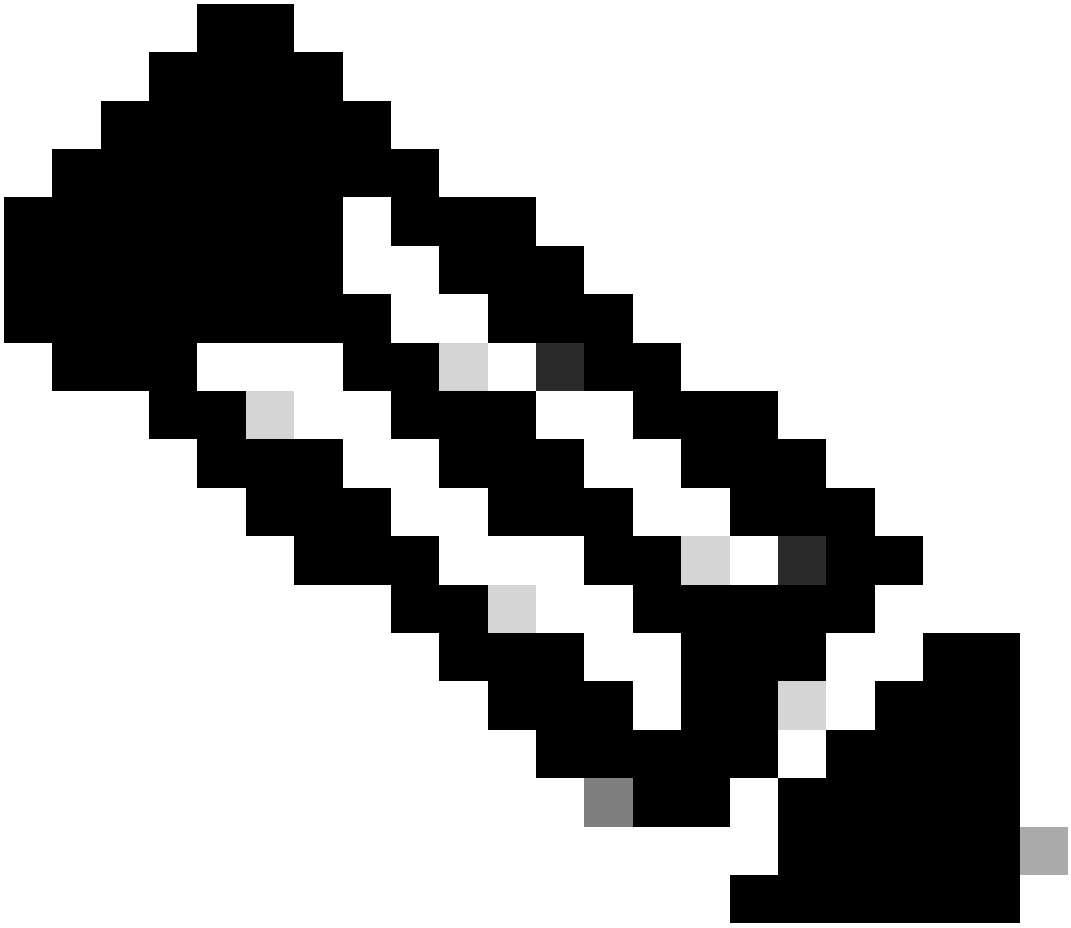
Componentes Utilizados

As informações neste documento são baseadas nestas versões de software e hardware:

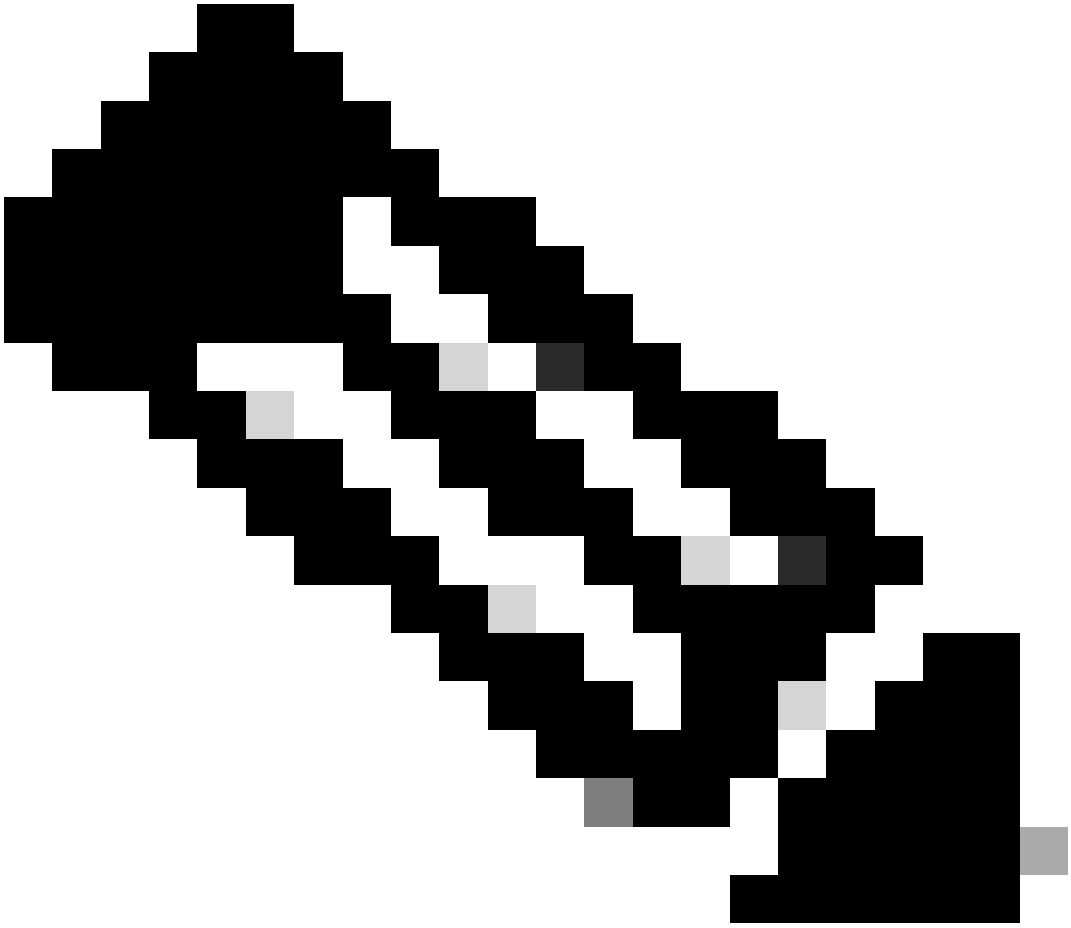
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1 e versões posteriores



Note: Consulte o guia de configuração apropriado para obter os comandos usados para ativar esses recursos em outras plataformas da Cisco.



Note: Os switches Catalyst 9200L não oferecem suporte a Guest Shell.



Note: Esses scripts não são suportados pelo Cisco TAC e são fornecidos no estado em que se encontram para fins educacionais.

As informações neste documento foram criadas a partir de dispositivos em um ambiente de laboratório específico. Todos os dispositivos utilizados neste documento foram iniciados com uma configuração (padrão) inicial. Se a rede estiver ativa, certifique-se de que você entenda o impacto potencial de qualquer comando.

Conventions

Consulte as Convenções de Dicas Técnicas da Cisco para obter informações sobre convenções de documentos.

Informações de Apoio

Shell de convidado e scripts Python

A hospedagem de aplicativos na família de switches Cisco Catalyst 9000 apresenta oportunidades de inovação para parceiros e desenvolvedores, pois os dispositivos de rede podem ser mesclados com um ambiente de tempo de execução de aplicativos.

Ele suporta aplicativos em contêineres, fornecendo isolamento completo do sistema operacional principal e do Kernel Cisco IOS XE. Essa separação garante que as alocações de recursos para aplicativos hospedados sejam diferentes das funções centrais de roteamento e comutação.

A infraestrutura de hospedagem de aplicativos para dispositivos Cisco IOS XE é conhecida como IOx (Cisco IOS + Linux), que facilita a hospedagem de aplicativos e serviços desenvolvidos pela Cisco, parceiros e desenvolvedores de terceiros em dispositivos de rede, garantindo integração transparente em diversas plataformas de hardware.

O Guest Shell, uma implantação de contêiner especializada, exemplifica uma aplicação benéfica para a implantação do sistema.

O Guest Shell oferece um ambiente baseado em Linux virtualizado projetado para executar aplicativos Linux personalizados, incluindo Python, para permitir controle e gerenciamento automatizados de dispositivos Cisco. O contêiner Guest Shell permite que os usuários executem scripts e aplicativos no sistema. Especificamente, nas plataformas Intel x86, o contêiner Guest Shell é um Linux Container (LXC) com um sistema de arquivos raiz mínimo CentOS 8.0. No Cisco IOS XE Amsterdam 17.3.1 e versões posteriores, somente Python V3.6 é suportado. Bibliotecas Python adicionais podem ser instaladas durante o tempo de execução usando o utilitário Yum no CentOS 8.0. Pacotes Python também podem ser instalados ou atualizados usando Pacotes de Instalação Pip (PIP).

O Guest Shell inclui uma API (Application Programming Interface, interface de programação de aplicativos) Python, que permite executar comandos do Cisco IOS XE usando o módulo CLI Python. Dessa forma, os scripts Python melhoram os recursos de automação, fornecendo aos engenheiros de rede uma ferramenta versátil para desenvolver scripts para automatizar tarefas de configuração e coleta de dados. Embora esses scripts possam ser executados manualmente por meio da CLI, eles também podem ser empregados junto com scripts EEM para responder a eventos específicos, como mensagens de syslog, eventos de interface ou execuções de comandos. Praticamente, qualquer evento que possa acionar um script EEM também pode ser usado para acionar um script Python, expandindo o potencial de automação nos switches Cisco Catalyst 9000.

Quando o Guest Shell é instalado, um diretório de compartilhamento de convidados é criado automaticamente no sistema de arquivos flash. Este é o sistema de arquivos que pode ser acessado a partir dos scripts Python e do Guest Shell. Para garantir a sincronização adequada ao usar o empilhamento, mantenha esta pasta abaixo de 50 MB.

Benefícios de usar Python com scripts EEM

- O Python estende os recursos de automação dos scripts EEM permitindo que a lógica complexa (como expressões regulares, loops e correspondências) seja manipulada dentro do script Python. Esse recurso oferece uma oportunidade para criar scripts EEM mais poderosos.

- Como uma linguagem de programação bem conhecida, o Python reduz a barreira de entrada para engenheiros de rede que desejam automatizar os dispositivos Cisco IOS XE. Além disso, ele oferece capacidade de manutenção e leitura.
- O Python também fornece recursos de manipulação de erros, bem como uma biblioteca padrão poderosa.

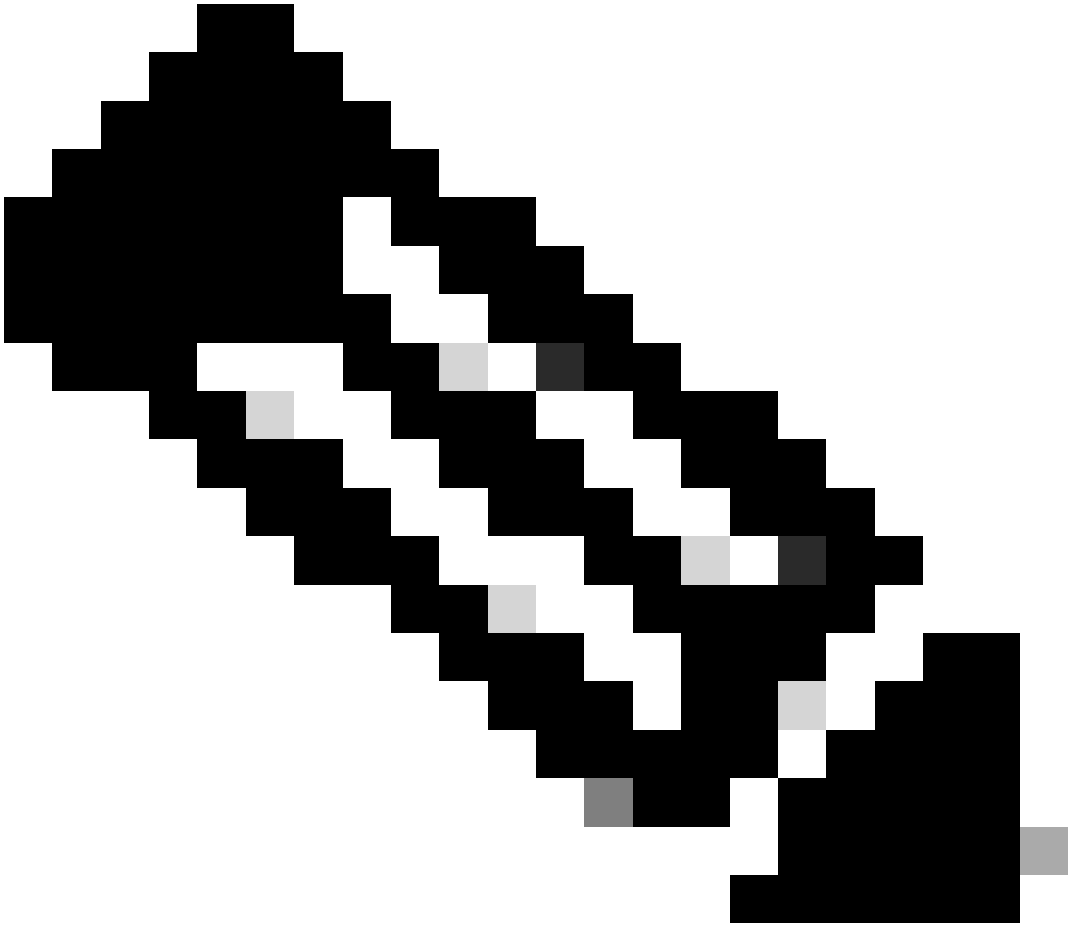
Considerações sobre o uso de Python com scripts EEM

- O Guest Shell não está habilitado por padrão, portanto, ele precisa ser habilitado para que os scripts Python possam ser executados.
- Scripts Python não podem ser criados diretamente no CLI; eles precisam ser desenvolvidos primeiro em um ambiente de desenvolvimento e depois copiados para a memória flash do switch.

SELinux no Cisco IOS XE

A partir do Cisco IOS XE 17.8.1, o suporte para Security-Enhanced Linux (SELinux) foi introduzido para reforçar a segurança com políticas que controlam como processos, usuários e arquivos interagem entre si. A política do SELinux define quais ações e recursos podem ser acessados por um processo ou usuário. Uma violação pode ocorrer quando um usuário ou processo tenta executar uma ação não permitida pela política, por exemplo, acessar um recurso ou executar um comando. SELinux pode operar em 2 modos diferentes:

1. Modo de permissão: O SELinux não impõe nenhuma política. No entanto, ele ainda registra todas as violações como se elas estivessem sendo negadas.
2. Aplicação: O SELinux aplica ativamente as políticas de segurança no sistema. Se uma ação violar a política do SELinux, a ação será negada e registrada.



Note: O modo padrão foi definido como permissivo quando foi introduzido no Cisco IOS XE 17.8.1. No entanto, a partir da versão 17.14.1, o SELinux é habilitado no modo de imposição.

Ao usar o Shell do Convidado, o acesso a alguns recursos pode ser negado ao usar o modo de imposição. Se ao tentar executar uma ação usando o Shell do Convidado ou um script Python resultar em um erro de permissão negada, semelhante a este log:

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

Para verificar se um script está sendo negado pelo SELinux, use o comando`show platform software audit summary`para verificar se as contagens de negação estão aumentando. Além disso,`show platform software audit all`exibe um registro de ações bloqueadas pelo SELinux. O Cache de Vetor de Acesso (AVC) é o mecanismo usado para registrar as decisões de controle de acesso no SELinux, portanto, ao

usar esse comando, procure logs que comecem com type=AVC.

Se um script estiver sendo negado e bloqueado, o SELinux pode ser definido para o modo permissivo usando o comando`set platform software selinux permissive`. Essa alteração não é armazenada na configuração de execução ou de inicialização, portanto, após um recarregamento, o modo volta a ser imposto. Portanto, cada vez que o switch for recarregado, essa alteração precisará ser reaplicada. A alteração pode ser verificada usando`show platform software selinux`.

Configurar

Para automatizar tarefas no switch usando scripts EEM e Python, siga estas etapas:

1. Habilite o Shell do Convidado.
2. Copie o script Python para o diretório /flash/guest-share/. Você pode usar qualquer mecanismo de cópia disponível no Cisco IOS XE, como SCP, FTP ou o Gerenciador de arquivos na WebUI. Quando o script Python estiver na memória flash, você poderá executá-lo usando o comando`guestshell run python3 /flash/guest-share/cat9k_script.py`.
3. Configure um script EEM que execute o script Python. Essa configuração permite que o script Python seja acionado usando qualquer um dos vários detectores de eventos fornecidos pelos scripts EEM, como uma mensagem de syslog, um padrão CLI e um agendador Cron.

Nesta seção, a Etapa 1 é explicada. A próxima seção fornece exemplos que demonstram como implementar as Etapas 2 e 3.

Habilite o shell convidado com um endereço IP estático

Siga este processo para ativar o Guest Shell:

1. Habilite o 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 a Rede de Hospedagem de Aplicativos para o Shell do Convidado. Este exemplo usa a interface AppGigabitEthernet para fornecer acesso à rede; no entanto, a interface de gerenciamento (Gi0/0) também pode ser usada.

```
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 o Shell do Convidado.

<#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 o Shell do Convidado. Este exemplo valida que há acessibilidade com o gateway padrão e também com cisco.com. Além disso, confirme se o Python 3 pode ser executado a partir do Guest Shell.

<#root>

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

show app-hosting list

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

guestshell

RUNNING

```
Switch#
```

guestshell run bash

```
[guestshell@guestshell ~]$

! Validate that the IP address of the Guest Shell is configured correctly.
[guestshell@guestshell ~]$

sudo ifconfig

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


inet 10.20.1.2  netmask 255.255.255.0

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

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

! Validate reachability to the default gateway and ensure that DNS is resolving correctly.
[guestshell@guestshell ~]$

ping 10.20.1.1

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

[guestshell@guestshell ~]$

ping cisco.com

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

! Validate the Python version.
[guestshell@guestshell ~]$

python3 --version

Python 3.6.8
```

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

python3

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

print("Cisco")
Cisco

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

[guestshell@guestshell ~]$ exit
exit

Switch#
```

Ativar o shell convidado com um endereço IP DHCP

Geralmente, o Guest Shell é configurado com um endereço IP estático, pois o contêiner do Guest Shell não tem o serviço de cliente DHCP por padrão. Se for necessário que o Guest Shell solicite um endereço IP dinamicamente, o serviço de cliente DHCP precisará ser instalado. Siga este processo:

1. Siga as etapas para ativar o Guest Shell com um endereço IP estático. No entanto, desta vez, não atribua o endereço IP na configuração de hospedagem do aplicativo durante a etapa 2. Em vez disso, use esta configuração:

```
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. O cliente DHCP pode ser instalado usando o utilitário Yum com o comando`sudo yum install dhcp-client`. No entanto, os repositórios para o CentOS Stream 8 foram desativados. Para resolver isso, os pacotes de cliente DHCP podem ser baixados e instalados manualmente. Em um PC, baixe esses pacotes do cofre CentOS Stream 8 e empacote-os em um arquivo 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 o arquivo dhcp-client.tar para o diretório /flash/guest-share/ no switch.
4. Insira uma sessão bash de shell convidado e execute os comandos Linux para instalar o cliente DHCP e solicitar um endereço 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 de uso 1: Salvamento automático de alterações de configuração em um servidor SCP

Em algumas situações, é vantajoso salvar automaticamente as configurações do switch em um servidor toda vez que o comando`write memory`é usado. Essa prática ajuda a manter um registro das alterações e permite a reversão da configuração, se necessário. Ao escolher um servidor, TFTP e SCP podem ser usados, mas um servidor SCP oferece uma camada adicional de segurança.

O recurso de arquivamento do Cisco IOS fornece essa funcionalidade. No entanto, uma desvantagem significativa é que as credenciais SCP não podem ser ocultas na configuração; o caminho do servidor é exibido em texto simples nas configurações de execução e de inicialização.

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

Usando o Guest Shell e o Python, é possível obter a mesma funcionalidade enquanto as credenciais são mantidas ocultas. Isso é feito aproveitando as variáveis de ambiente no Shell do

convidado para armazenar as credenciais reais do SCP. Como resultado, as credenciais do servidor SCP não são visíveis na configuração atual.

Nessa abordagem, a configuração em execução exibe apenas o script EEM, enquanto os scripts Python constróem o`copy running-config scp:`comando com as credenciais e o passa para o script EEM para ser executado.

Siga estas etapas para este exemplo:

1. Copie o script Python para o diretório /flash/guest-share. Esse script lê as variáveis de ambiente SCP_USER e SCP_PASSWORD e retorna o`copy startup-config scp:`comando, para que o script EEM possa executá-lo. O script requer o endereço IP do servidor SCP como um argumento. Além disso, o script mantém um registro de cada vez que o comando`write memory`é executado em um arquivo persistente localizado em /flash/guest-share/TAC-write-memory-log.txt. Este é o 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
```

Neste exemplo, o script Python é copiado para o switch usando um servidor TFTP:

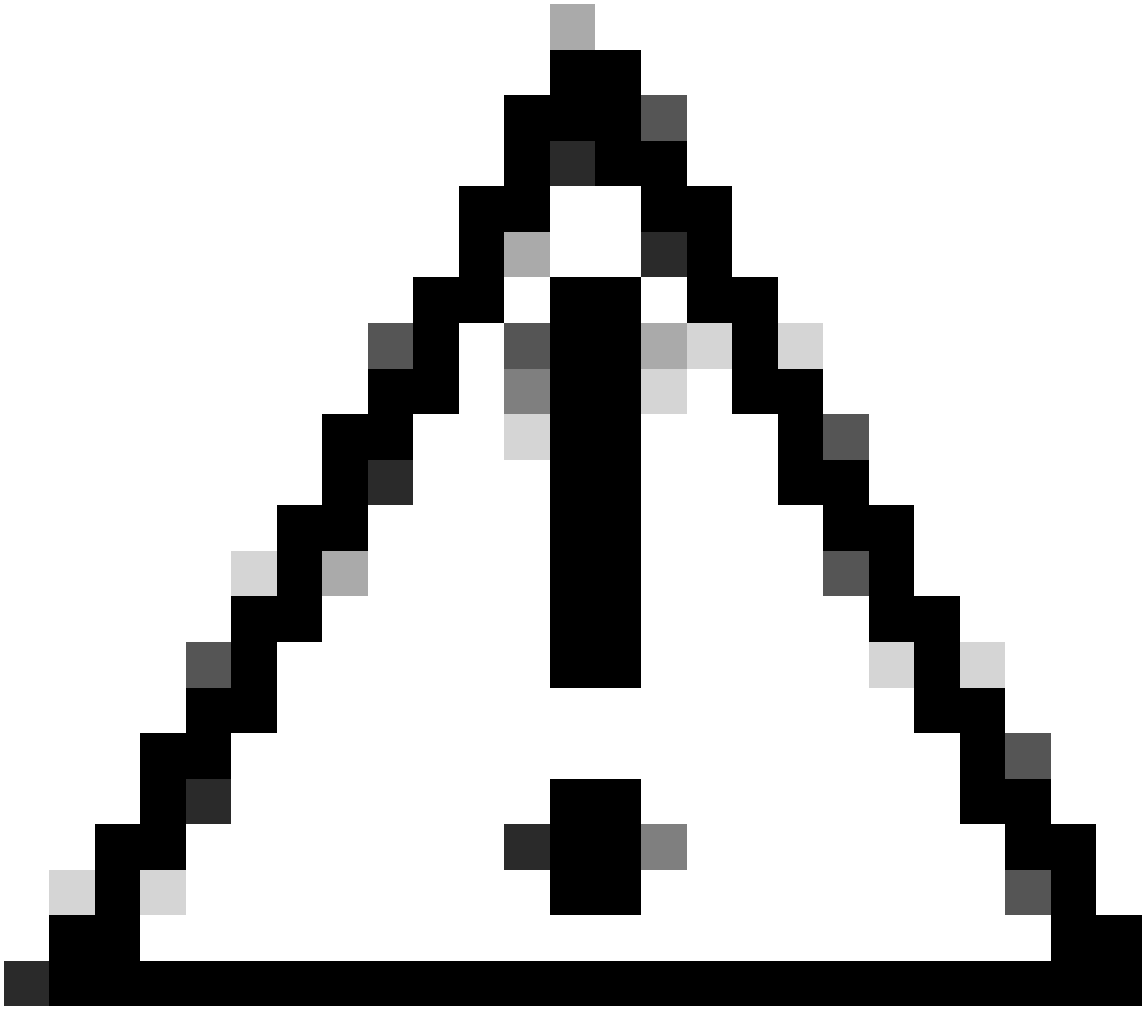
```
<#root>

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 o script EEM. Este script chama o script Python. que retorna o comando`copy startup-config scp:`necessário para salvar a configuração no servidor SCP. O script EEM executa o comando retornado pelo 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. Defina as variáveis de ambiente do Guest Shell inserindo-as no arquivo`~/.bashrc`. Isso garante que as variáveis de ambiente sejam persistentes toda vez que um Guest Shell é aberto, mesmo após o switch ser recarregado. Adicione estas duas linhas:

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



Caution: As credenciais usadas neste exemplo são apenas para fins educacionais. Eles não se destinam ao uso em ambientes de produção. Os usuários devem substituir essas credenciais por suas próprias credenciais seguras e específicas do ambiente.

Este é o processo para adicionar estas variáveis ao arquivo~/.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. Execute o script Python manualmente para validar que ele retorna o comando correto`copy`e registra a operação no arquivo 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. Teste o script EEM. Esse script EEM também envia um syslog com o resultado da operação de cópia, seja bem-sucedido ou com falha. Aqui está um exemplo de uma execução bem-sucedida:

```
<#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 testar uma transferência com falha, o servidor SCP é desligado. Este é o resultado desta execução com falha:

```
<#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 de uso 2: Monitorar Incrementos em Alterações de Topologia STP

Este exemplo é útil para monitorar problemas relacionados à instabilidade do Spanning Tree e identificar qual interface está recebendo TCNs (Topology Change Notifications). O script EEM é

executado periodicamente em um intervalo de tempo especificado e chama um script Python que executa um comando show e verifica se os TCNs aumentaram.

A criação desse script usando somente comandos EEM exigiria o uso de loops for e várias correspondências regex, o que seria incômodo. Portanto, este exemplo mostra como o script EEM delega essa lógica complexa para Python.

Siga estas etapas para este exemplo:

1. Copie o script Python para o diretório /flash/guest-share/. Este script executa estas tarefas:

    1. Ele executa o comando`show spanning-tree detail`e analisa a saída para salvar informações de TCN para cada VLAN em um dicionário.

    2. Ele compara as informações TCN analisadas com os dados no arquivo JSON da execução do script anterior. Para cada VLAN, se os TCNs tiverem aumentado, uma mensagem de syslog será enviada com informações semelhantes a este exemplo:

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

    3. Salva as informações atuais do TCN em um arquivo JSON para serem comparadas durante a próxima execução. Este é o 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()
```

Neste exemplo, o script Python é copiado para o switch usando um 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 o script EEM. Neste exemplo, a única tarefa do script EEM é executar o script Python, que envia uma mensagem de log se um incremento de TCN for detectado. Neste exemplo, o script EEM é executado a 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 a funcionalidade do script, você pode executar o script Python manualmente ou esperar cinco minutos para que o script EEM o chame. Em ambos os casos, um syslog é enviado somente se os TCNs tiverem aumentado para uma 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. Teste o script EEM esperando que ele seja executado a cada cinco minutos. Se os TCNs tiverem aumentado para qualquer VLAN, uma mensagem de syslog será enviada. Neste exemplo específico, observe-se que os TCNs estão constantemente aumentando na VLAN 30, e a interface que recebe esses TCNs constantes é 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.
```

Informações Relacionadas

- Guia de configuração de programabilidade, Cisco IOS XE Cupertino 17.9.x (Capítulo: Shell do convidado)
- Compreender as práticas recomendadas e os scripts úteis para o EEM
- Documentação de hospedagem de aplicativos no Cisco Catalyst 9000 Series Switches

Sobre esta tradução
A Cisco traduziu este documento com a ajuda de tecnologias de tradução automática e humana para oferecer conteúdo de suporte aos seus usuários no seu próprio idioma, independentemente da localização.

Observe que mesmo a melhor tradução automática não será tão precisa quanto as realizadas por um tradutor profissional.

A Cisco Systems, Inc. não se responsabiliza pela precisão destas traduções e recomenda que o documento original em inglês (link fornecido) seja sempre consultado.