Automatiseer Catalyst 9000 Switches met Python-scripts

Inhoud

Inleiding

Dit document beschrijft hoe u EEM met Python-scripts kunt uitbreiden voor de automatisering van configuratie en gegevensverzameling op Catalyst 9000 switches.

Voorwaarden

Vereisten

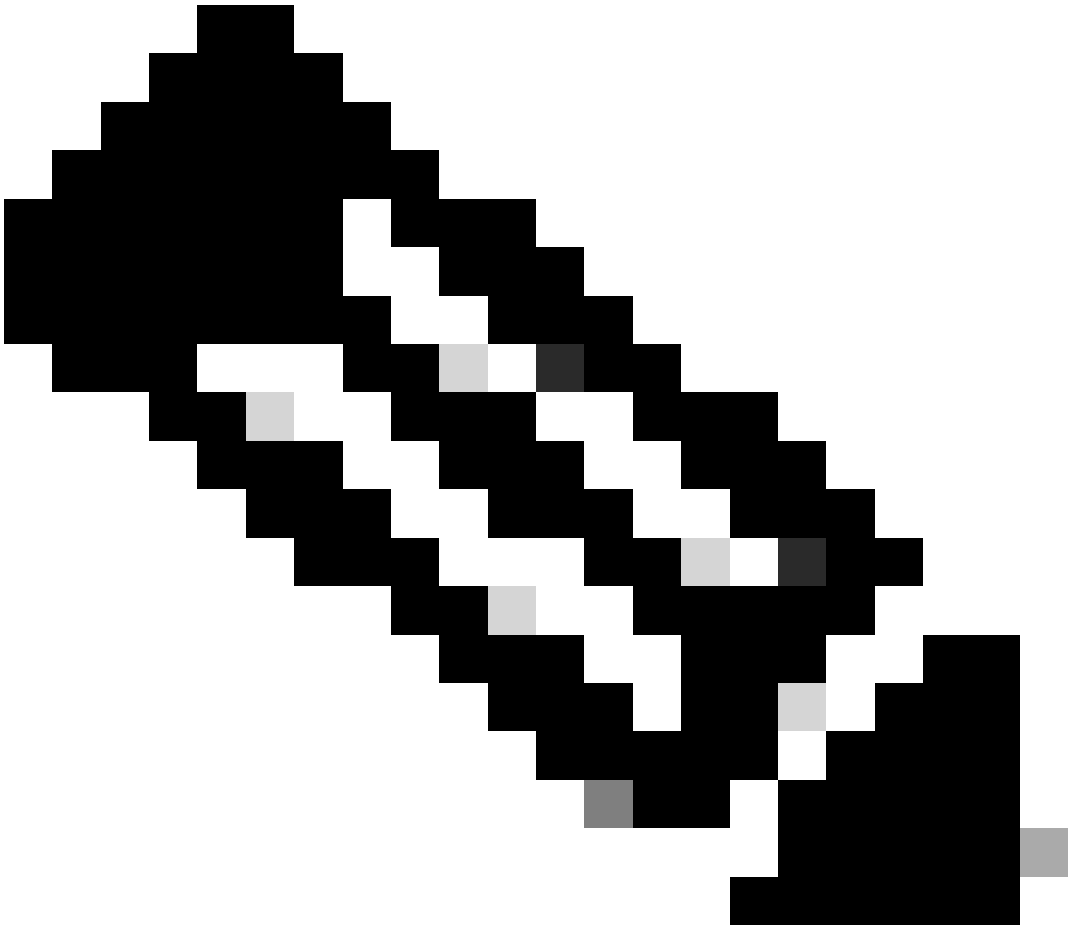
Cisco raadt u aan kennis te hebben van en vertrouwd te zijn met deze onderwerpen:

- Cisco IOS® en Cisco IOS® XE EM
- Application Hosting en Guest Shell
- Python-scripting
- Linux-opdrachten

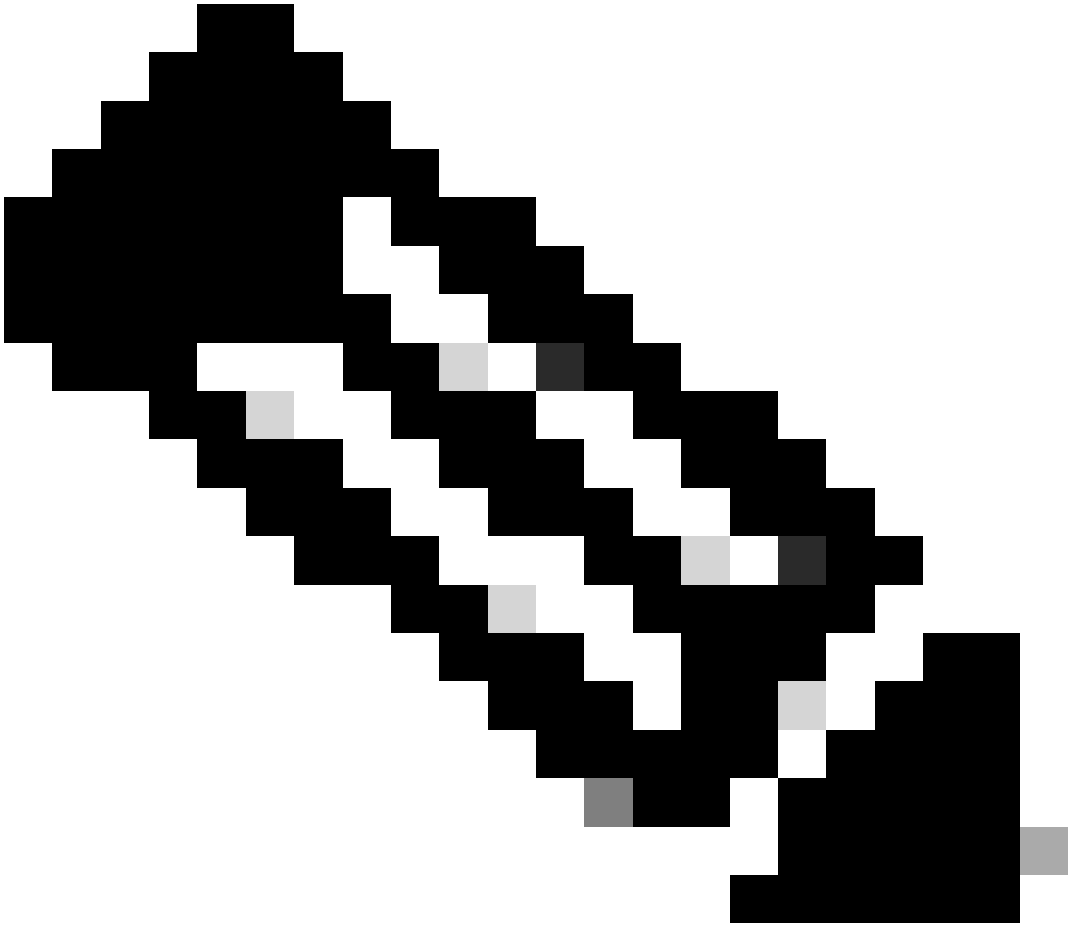
Gebruikte componenten

De informatie in dit document is gebaseerd op de volgende software- en hardware-versies:

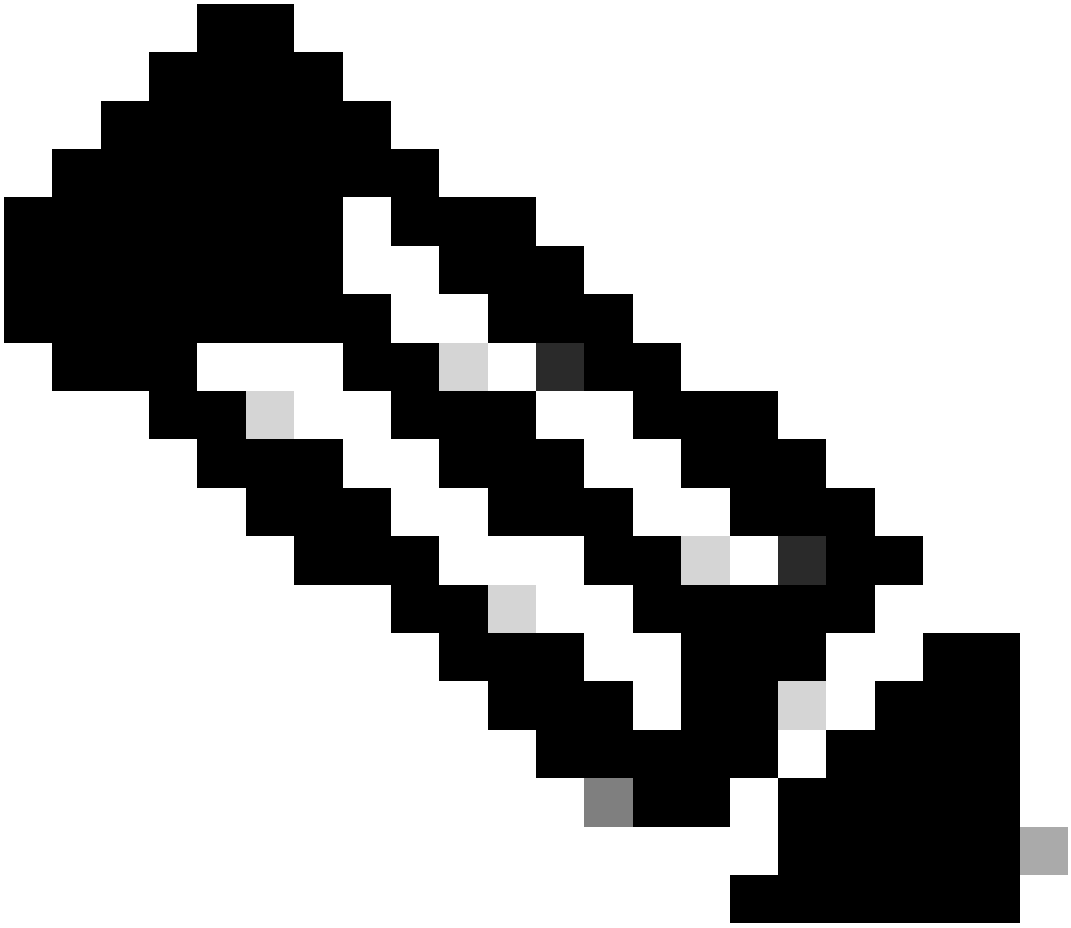
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1 en latere versies



Opmerking: Raadpleeg de betreffende configuratiehandleiding voor de opdrachten die worden gebruikt om deze functies op andere Cisco-platforms in te schakelen.



Opmerking: Catalyst 9200L switches ondersteunen Guest Shell niet.



Opmerking: Deze scripts worden niet ondersteund door Cisco TAC en worden op een as-is-basis voor onderwijsdoeleinden geleverd.

De informatie in dit document is gebaseerd op de apparaten in een specifieke laboratoriumomgeving. Alle apparaten die in dit document worden beschreven, hadden een opgeschoonde (standaard)configuratie. Als uw netwerk live is, moet u zorgen dat u de potentiële impact van elke opdracht begrijpt.

Conventies

Raadpleeg Conventies voor technische tips van Cisco voor informatie over documentconventies.

Achtergrondinformatie

Guest Shell en Python scripts

Toepassingshosting op de Cisco Catalyst 9000-reeks switches biedt innovatiemogelijkheden voor partners en ontwikkelaars, aangezien netwerkapparaten kunnen worden samengevoegd met een runtime-omgeving voor toepassingen.

Het ondersteunt containertoepassingen en biedt volledige isolatie van het hoofdbesturingssysteem en de Cisco IOS XE Kernel. Deze scheiding zorgt ervoor dat de middeltoewijzingen voor ontvangen toepassingen van de kern die en omschakelingsfuncties verschillend zijn.

De toepassingshostinginfrastructuur voor Cisco IOS XE-apparaten staat bekend als IOx (Cisco IOS + Linux), wat de hosting van toepassingen en services vergemakkelijkt die zijn ontwikkeld door Cisco, partners en externe ontwikkelaars op netwerkapparaten, waardoor een naadloze integratie over verschillende hardwareplatforms wordt gegarandeerd.

De Guest Shell, een gespecialiseerde containerinzet, is een voorbeeld van een toepassing die voordelig is voor systeemimplementatie.

De Guest Shell biedt een gevirtualiseerde Linux-gebaseerde omgeving die ontworpen is om aangepaste Linux-toepassingen, waaronder Python, uit te voeren om geautomatiseerde controle en beheer van Cisco-apparaten mogelijk te maken. De container van de Gast Shell staat gebruikers toe om scripts en apps binnen het systeem uit te voeren. Op Intel x86 platforms is de Guest Shell container een Linux Container (LXC) met een CentOS 8.0 minimale root bestandssysteem. In Cisco IOS XE Amsterdam 17.3.1 en latere releases wordt alleen Python V3.6 ondersteund. Extra Python-bibliotheken kunnen tijdens de uitvoering worden geïnstalleerd met behulp van het Yum-hulpprogramma in CentOS 8.0. Python-pakketten kunnen ook worden geïnstalleerd of bijgewerkt met behulp van Pip Install Packages (PIP).

Guest Shell bevat een Python Application Programming Interface (API), waarmee Cisco IOS XE-opdrachten kunnen worden uitgevoerd met behulp van de Python CLI-module. Op deze manier verbeteren Python-scripts de automatiseringsmogelijkheden, waardoor netwerkengineers beschikken over een veelzijdig hulpmiddel om scripts te ontwikkelen voor het automatiseren van configuratie- en gegevensverzamelingstaken. Terwijl deze scripts handmatig door de CLI kunnen worden uitgevoerd, kunnen ze ook naast EEM-scripts worden gebruikt om te reageren op specifieke gebeurtenissen, zoals syslog-berichten, interfacegebeurtenissen of opdrachtruns. In de praktijk kan elke gebeurtenis die een EEM-script kan activeren ook worden gebruikt om een Python-script te activeren, waardoor het automatiseringspotentieel binnen Cisco Catalyst 9000 switches wordt uitgebreid.

Wanneer Guest Shell is geïnstalleerd, wordt er automatisch een guest-share map aangemaakt in het flash-bestandssysteem. Dit is het bestandssysteem dat kan worden benaderd vanuit de Python-scripts en de Guest Shell. Om bij het stapelen een goede synchronisatie te waarborgen, moet u deze map bewaren onder 50 MB.

Voordelen van het gebruik van Python met EEM-scripts

- Python breidt de automatiseringsmogelijkheden van EEM-scripts uit door complexe logica (zoals reguliere expressies, lussen en overeenkomsten) toe te laten om te worden verwerkt in het Python-script. Deze mogelijkheid biedt de mogelijkheid om krachtigere EEM-scripts te maken.

- Als bekende programmeertaal verlaagt Python de toegangsdrempel voor netwerkengineers die Cisco IOS XE-apparaten willen automatiseren. Bovendien biedt het onderhoudbaarheid en leesbaarheid.
- Python biedt ook mogelijkheden voor foutenbehandeling en een krachtige standaardbibliotheek.

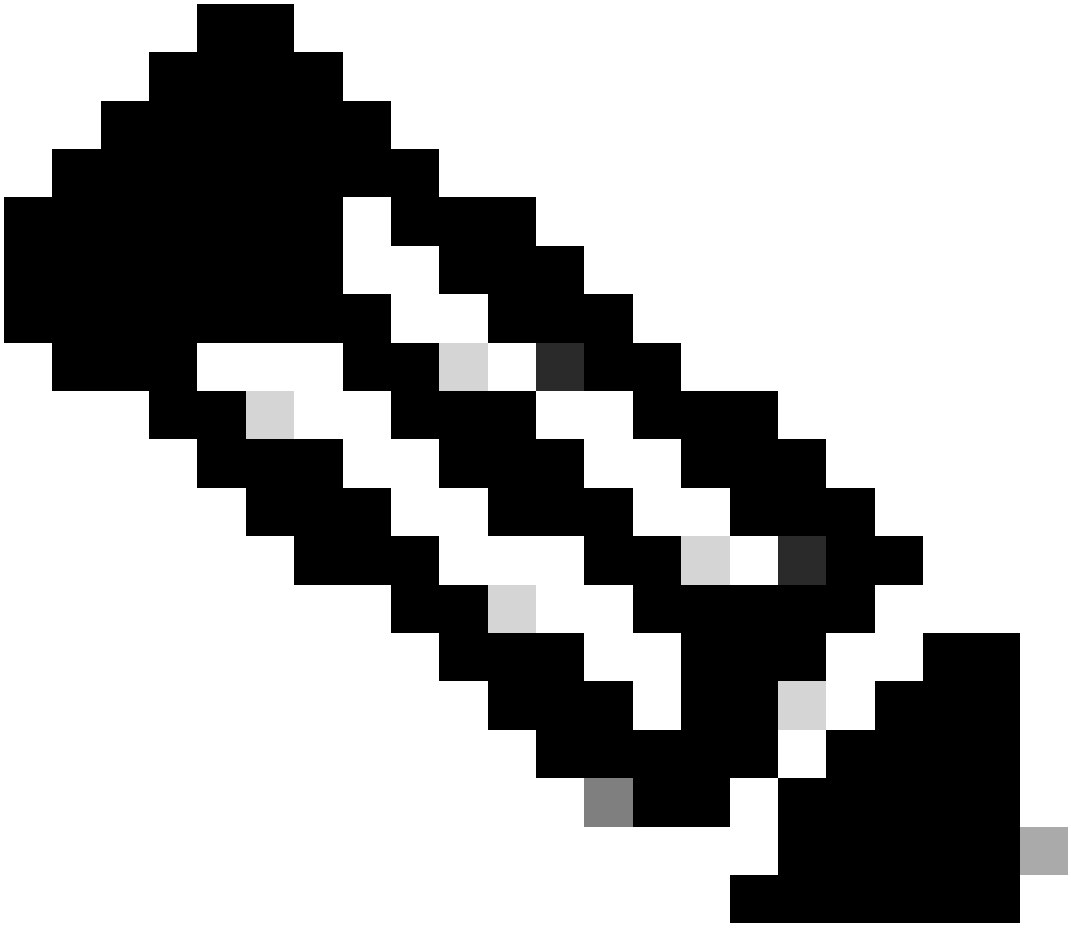
Overwegingen bij het gebruik van Python met EEM-scripts

- Guest Shell is standaard niet ingeschakeld, dus het moet ingeschakeld worden voordat Python-scripts kunnen worden uitgevoerd.
- Python-scripts kunnen niet rechtstreeks in de CLI worden gemaakt. ze moeten eerst in een ontwikkelingsomgeving worden ontwikkeld en daarna worden gekopieerd naar het flash-geheugen van de switch.

SELinux in Cisco IOS XE

Beginnend met Cisco IOS XE 17.8.1, werd de steun voor Security-Enhanced Linux (SELinux) geïntroduceerd om veiligheid met beleid af te dwingen dat bepaalt hoe processen, gebruikers, en dossiers met elkaar in wisselwerking staan. Het SELinux-beleid bepaalt welke acties en middelen toegankelijk zijn voor een proces of gebruiker. Een schending kan voorkomen wanneer een gebruiker of een proces probeert om een actie uit te voeren die niet is toegestaan door het beleid, bijvoorbeeld toegang tot een bron of het uitvoeren van een opdracht. SELinux kan in 2 verschillende modi werken:

1. Toegestane modus: SELinux dwingt geen enkel beleid af. Toch worden er nog steeds schendingen geregistreerd alsof ze werden ontkend.
2. Handhaving: SELinux dwingt het beveiligingsbeleid op het systeem actief af. Als een actie het SELinux-beleid schendt, wordt de actie geweigerd en geregistreerd.



Opmerking: De standaardmodus is op permissive ingesteld toen deze in Cisco IOS XE 17.8.1 werd geïntroduceerd. SELinux, beginnend met versie 17.14.1, is echter ingeschakeld in de afdwingingsmodus.

Wanneer het gebruiken van Guest Shell, kan toegang tot sommige middelen worden ontzegd wanneer het gebruiken van afdwingende wijze. Als bij het uitvoeren van een actie met behulp van Guest Shell of een Python-script een fout met toegangsrechten wordt ontkend, vergelijkbaar met dit logbestand:

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

Om te verifiëren of een script wordt ontkend door SELinux, gebruikt u de opdracht`show platform software audit summary`om te controleren of het aantal ontkenningen toeneemt. Daarnaast`show platform software audit all`wordt een overzicht weergegeven van de acties die door SELinux zijn geblokkeerd. De Access

Vector Cache (AVC) is het mechanisme dat wordt gebruikt om toegangscontrole beslissingen te registreren in SELinux, dus wanneer u deze opdracht gebruikt, zoekt u naar logboeken die beginnen met type=AVC.

Als een script wordt geweigerd en geblokkeerd, kan SELinux met behulp van de opdracht worden ingesteld op de modus Permissive`set platform software selinux permissive`. Deze verandering wordt niet opgeslagen in de lopende of startconfiguratie, zodat na een herlading, de wijze aan het afdwingen terugkeert. Daarom moet deze wijziging elke keer dat de switch opnieuw wordt geladen, opnieuw worden toegepast. De wijziging kan worden geverifieerd met`show platform software selinux`.

Configureren

Volg de volgende stappen om taken op de switch te automatiseren met EEM- en Python-scripts:

1. Schakel de Guest Shell in.
2. Kopieer het Python script naar /flash/guest-share/ directory. U kunt elk kopieermechanisme gebruiken dat beschikbaar is in Cisco IOS XE, zoals SCP, FTP of FormFiller in de WebUI. Zodra het Python-script in het flitsgeheugen is opgeslagen, kunt u het uitvoeren met behulp van de opdracht`guestshell run python3 /flash/guest-share/cat9k_script.py`.
3. Configureer een EEM script dat het Python script uitvoert. Deze instelling maakt het mogelijk om het Python script te activeren met behulp van een van de meerdere gebeurtenisdetectoren die EEM scripts, zoals een syslog bericht, een CLI patroon en een Cron planner.

In deze sectie, wordt Stap 1 verklaard. De volgende sectie geeft voorbeelden die aantonen hoe de stappen 2 en 3 moeten worden uitgevoerd.

Schakel de Guest Shell in met een statisch IP-adres

Volg dit proces om de Guest Shell in te schakelen:

1. IOx inschakelen.

<#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. Configureer het Application Hosting Network voor de Guest Shell. Dit voorbeeld gebruikt de interface AppGigabit Ethernet om netwerktoegang te verlenen; de Management-interface

(Gi0/0) kan echter ook worden gebruikt.

```
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. Schakel de Guest Shell in.

<#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. Bevestig de Guest Shell. Dit voorbeeld bevestigt dat er bereikbaarheid is met de standaardgateway en met cisco.com. Controleer ook of Python 3 vanuit de Guest Shell kan worden uitgevoerd.

<#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#
```

De Guest Shell met een DHCP-IP-adres inschakelen

Meestal, de Guest Shell is geconfigureerd met een statisch IP-adres omdat de Guest Shell-container niet standaard de DHCP-clientservice heeft. Als het voor de Gast Shell noodzakelijk is om een IP-adres dynamisch aan te vragen, moet de DHCP-clientservice worden geïnstalleerd. Volg dit proces:

1. Volg de stappen om de Guest Shell in te schakelen met een statisch IP-adres. Bij stap 2 moet u echter niet het IP-adres in de configuratie van de app-hosting toewijzen. Gebruik in plaats daarvan deze configuratie:

```
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. De DHCP-client kan worden geïnstalleerd met behulp van het Yum-hulpprogramma met de opdracht`sudo yum install dhcp-client`. De opslagplaatsen voor CentOS Stream 8 zijn echter ontmanteld. Om dit te verhelpen, kunnen de DHCP-clientpakketten handmatig worden gedownload en geïnstalleerd. Download deze pakketten op een pc uit de CentOS Stream 8-kluis en verpak ze in een tar-bestand.

- bind-export-libs-9.11.36-13.el8.x86_64.rpm
- DHCP-client-4.3.6-50.el8.x86_64.rpm
- DHCP-gemeenschappelijk-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. Kopieer het dhcp-client.tar bestand naar de map /flash/guest-share/ in de switch.

4. Voer een Guest Shell bash sessie in en voer de Linux opdrachten uit om de DHCP client te installeren en een IP-adres op te vragen.

<#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
```

Use cases

Use Case 1: Automatische opslag van configuratiewijzigingen op een SCP-server

In bepaalde situaties is het handig om automatisch switch configuraties op te slaan naar een server telkens wanneer de opdracht wordt`write memory`gebruikt. Deze praktijk helpt bij het bijhouden van een register van wijzigingen en maakt indien nodig configuratie-terugdraaiing mogelijk. Wanneer het kiezen van een server, zowel kunnen TFTP als SCP worden gebruikt, maar een SCP server biedt een extra beveiligingslaag.

De Cisco IOS-archieffunctie biedt deze functionaliteit. Een belangrijk nadeel is echter dat SCP-referenties niet in de configuratie kunnen worden verborgen. het serverpad wordt in onbewerkte tekst weergegeven in zowel de actieve als de opstartconfiguraties.

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

Door gebruik te maken van de Guest Shell en Python, is het mogelijk om dezelfde functionaliteit te bereiken terwijl de referenties verborgen blijven. Dit wordt gedaan door milieu variabelen binnen de Gast Shell leveraging om de daadwerkelijke geloofsbrieven op te slaan SCP. Dientengevolge, zijn de SCP servergeloofsbrieven niet zichtbaar in de lopende configuratie.

In deze benadering geeft de actieve configuratie alleen het EEM-script weer, terwijl de Python-scripts de opdracht met de referenties construerencopy running-config scp:en deze doorgeven aan het EEM-script dat moet worden uitgevoerd.

Volg deze stappen bij dit voorbeeld:

1. Kopieer het Python-script naar de map /flash/guest-share. Dit script leest de omgevingsvariabelen SCP_USER en SCP_PASSWORD en geeft de opdracht terug, zodat hetcopy startup-config scp:EEM script het kan uitvoeren. Het script vereist als argument het IP-adres van de SCP-server. Daarnaast houdt het script een logbestand bij elke keer dat de opdrachtwrite memorywordt uitgevoerd in een persistent bestand op /flash/guest-share/TAC-write-memory-log.txt. Dit is het Python-script:

```
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
```

In dit voorbeeld wordt het Python-script naar de switch gekopieerd met behulp van een TFTP-server:

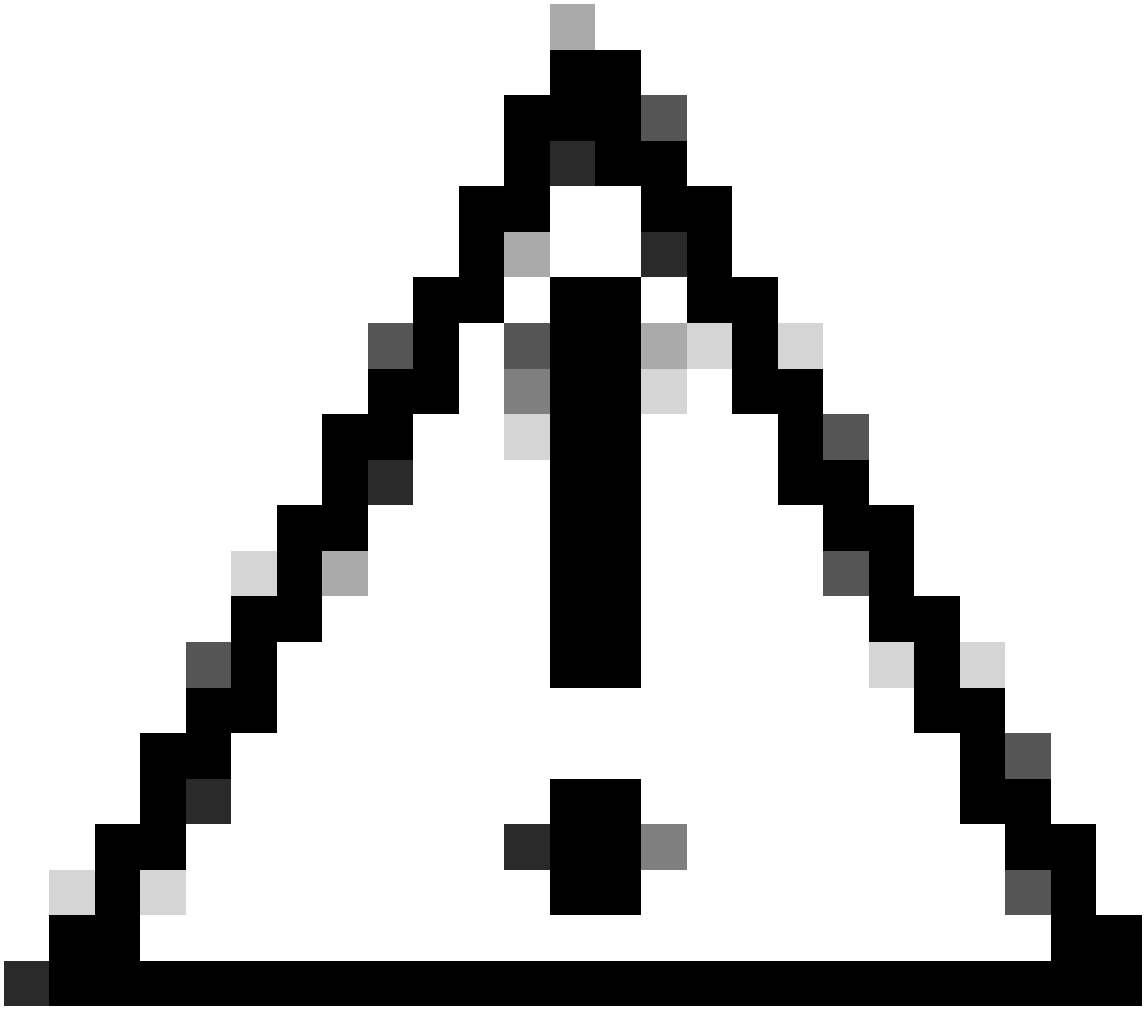
```
<#root>

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. Installeer het EEM script. Dit script noemt het Python script. die het`copy startup-config scp:`bevel terugkeert nodig om de configuratie op de SCP server op te slaan. Het EEM-script voert vervolgens de opdracht uit die wordt teruggegeven door het Python-script.

```
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. Stel de omgevingsvariabelen voor Guest Shell in door ze in het`~/.bashrc`bestand in te voegen. Dit zorgt ervoor dat de omgevingsvariabelen constant zijn telkens wanneer een Guest Shell wordt geopend, zelfs nadat de switch wordt herladen. Voeg deze twee regels toe:

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



Voorzichtig: De aanmeldingsgegevens die in dit voorbeeld worden gebruikt, zijn alleen voor educatieve doeleinden. Ze zijn niet bedoeld voor gebruik in productieomgevingen. Gebruikers moeten deze referenties vervangen door hun eigen veilige, milieucriteria.

Dit is het proces om deze variabelen aan het~/.bashrcbestand toe te voegen:

<#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. copyDraai het Python-script handmatig om te controleren of de juiste opdracht wordt geretourneerd en logt de bewerking in het persistentTAC-write-memory-log.txtbestand in.

```
<#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. Test het EEM script. Dit EEM-script stuurt ook een syslog met het resultaat van de kopieerbewerking, of het nu succesvol is of niet. Hier is een voorbeeld van een succesvolle run:

```
<#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.
```

Om een mislukte overdracht te testen, wordt de SCP server uitgeschakeld. Dit is het resultaat van deze mislukte run:

```
<#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.
```

Use Case 2: Verhogingen in STP-topologiewijzigingen bewaken

Dit voorbeeld is nuttig voor de controle van problemen met betrekking tot Spanning Tree-instabiliteit en het identificeren van welke interface meldingen van topologiewijziging ontvangt

(TCN's). Het EEM-script wordt periodiek met een bepaald tijdsinterval uitgevoerd en roept een Python-script dat een showopdracht uitvoert en controleert of de TCN's zijn verhoogd.

Het creëren van dit script met alleen EEM commando's zou het gebruiken voor loops en meerdere regex overeenkomsten vereisen, wat omslachtig zou zijn. Daarom laat dit voorbeeld zien hoe het EEM script deze complexe logica aan Python delegeert.

Volg deze stappen bij dit voorbeeld:

1. Kopieer het Python script naar /flash/guest-share/ directory. Dit script voert de volgende taken uit:

    1. Het stelt het`show spanning-tree detail`bevel in werking en ontleedt de output om TCN informatie voor elk VLAN in een woordenboek op te slaan.

    2. Het vergelijkt de geparste TCN-informatie met de gegevens in het JSON-bestand van de vorige script-run. Voor elk VLAN, als TCNs is verhoogd, wordt een syslogbericht verzonden met informatie gelijkend op dit voorbeeld:

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

    3. De huidige TCN-informatie wordt in een JSON-bestand opgeslagen om tijdens de volgende run te worden vergeleken. Dit is het Python-script:

```
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()
```

In dit voorbeeld wordt het Python-script naar de switch gekopieerd met behulp van een TFTP-server:

```
<#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. Installeer het EEM script. In dit voorbeeld is de enige taak van het EEM-script om het Python-script uit te voeren, dat een logbericht verstuurt als een TCN-toename wordt gedetecteerd. In dit voorbeeld draait het EEM script elke 5 minuten.

```
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. Om de functionaliteit van het script te valideren, kunt u het Python script handmatig uitvoeren of vijf minuten wachten tot het EEM script het aanroept. In beide gevallen wordt een syslog alleen verzonden als de TCN's voor een VLAN zijn toegenomen.

```
<#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. Test het EEM-script door erop te wachten dat het elke vijf minuten draait. Als de TCN's voor een VLAN zijn verhoogd, wordt er een syslogbericht verzonden. In dit specifieke voorbeeld, wordt opgemerkt dat TCNs constant op VLAN 30 stijgt, en de interface die deze constante TCNs ontvangt is 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.
```

Gerelateerde informatie

- De Gids van de Configuratie van de programmeerbaarheid, Cisco IOS XE Cupertino 17.9.x (Hoofdstuk: Guest Shell)
- Begrijp best practices en bruikbare scripts voor EEM
- Application Hosting op Cisco Catalyst 9000 Series Switches - witboek

Over deze vertaling

Cisco heeft dit document vertaald via een combinatie van machine- en menselijke technologie om onze gebruikers wereldwijd ondersteuningscontent te bieden in hun eigen taal. Houd er rekening mee dat zelfs de beste machinevertaling niet net zo nauwkeurig is als die van een professionele vertaler. Cisco Systems, Inc. is niet aansprakelijk voor de nauwkeurigheid van deze vertalingen en raadt aan altijd het oorspronkelijke Engelstalige document (link) te raadplegen.