Python 스크립트를 사용하여 Catalyst 9000 스위치 자동화

목차

소개

이 문서에서는 Catalyst 9000 스위치의 구성과 데이터 수집을 자동화하기 위해 Python 스크립트를 사용하여 EEM을 확장하는 방법에 대해 설명합니다.

사전 요구 사항

요구 사항

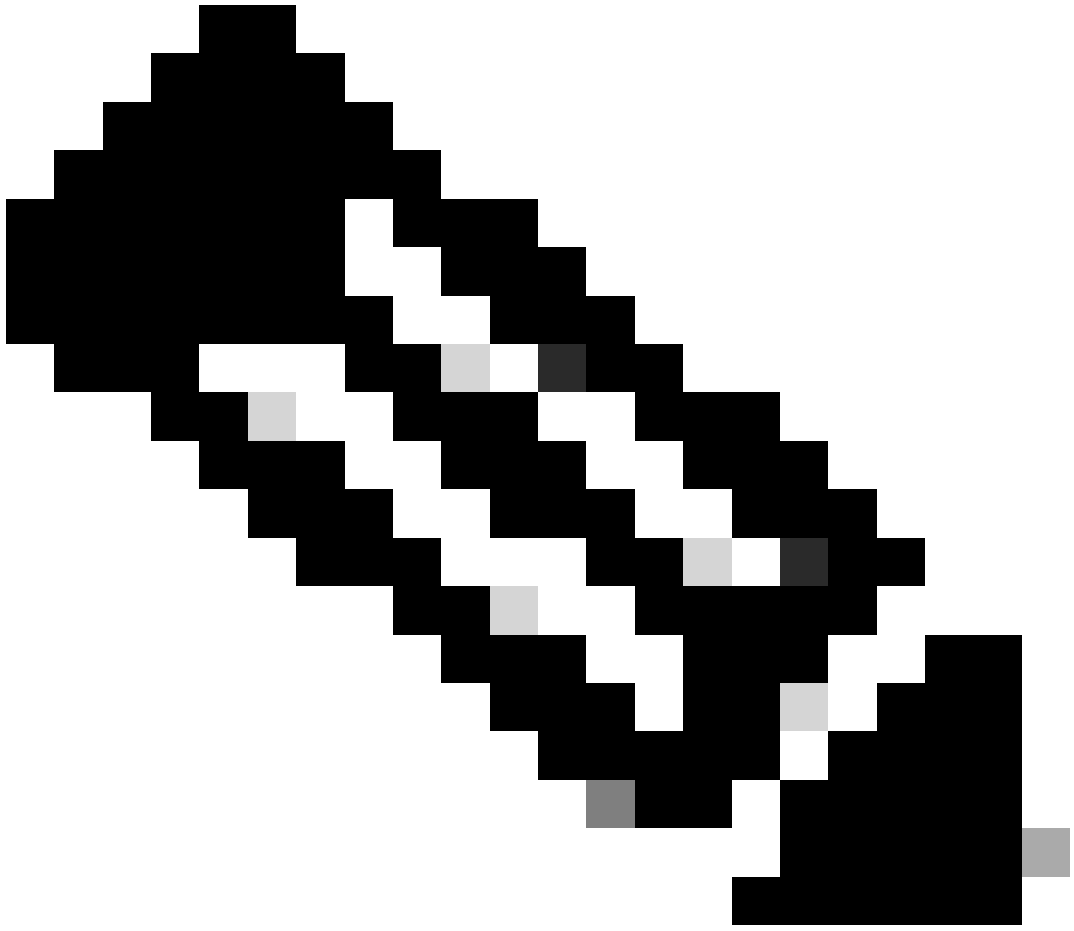
Cisco에서는 다음 항목에 대해 잘 알고 숙지할 것을 권장합니다.

- Cisco IOS® 및 Cisco IOS® XE EEM
- 애플리케이션 호스팅 및 게스트 셸
- Python 스크립팅
- Linux 명령

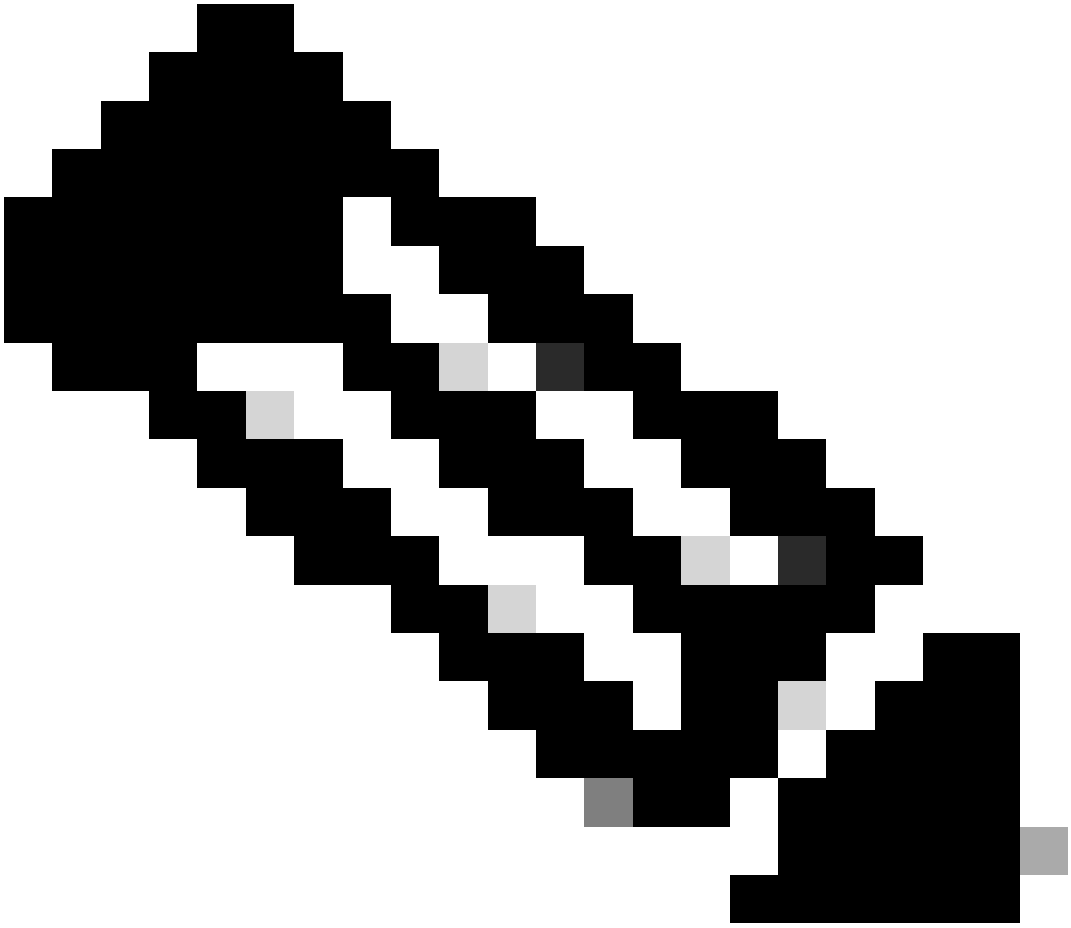
사용되는 구성 요소

이 문서의 정보는 다음 소프트웨어 및 하드웨어 버전을 기반으로 합니다.

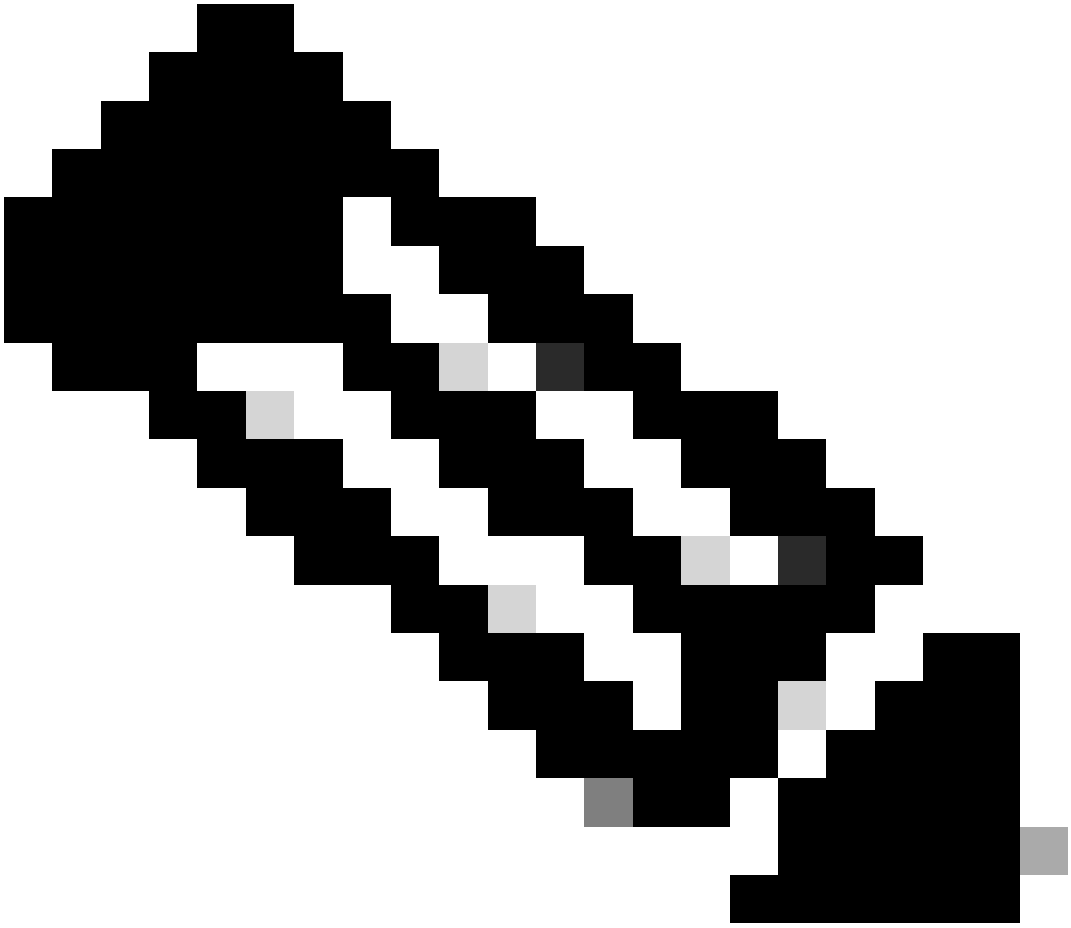
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1 이상 버전



참고: 다른 Cisco 플랫폼에서 이러한 기능을 활성화하는 데 사용되는 명령에 대해서는 해당 구성 가이드를 참조하십시오.



참고: Catalyst 9200L 스위치는 게스트 쉘을 지원하지 않습니다.



참고: 이러한 스크립트는 Cisco TAC에서 지원하지 않으며 교육 목적으로 있는 그대로 제공됩니다.

이 문서의 정보는 특정 랩 환경의 디바이스를 토대로 작성되었습니다. 이 문서에 사용된 모든 디바이스는 초기화된(기본) 컨피그레이션으로 시작되었습니다. 현재 네트워크가 작동 중인 경우 모든 명령의 잠재적인 영향을 미리 숙지하시기 바랍니다.

표기 규칙

문서 규칙에 대한 자세한 내용은 Cisco 기술 팁 표기 규칙을 참조하십시오.

배경 정보

게스트 셸 및 Python 스크립트

Cisco Catalyst 9000 스위치 제품군의 애플리케이션 호스팅은 네트워크 장치를 애플리케이션 런타

임 환경과 병합할 수 있으므로 파트너와 개발자에게 혁신 기회를 제공합니다.

컨테이너화된 애플리케이션을 지원하므로 기본 운영 체제 및 Cisco IOS XE Kernel과 완전히 격리됩니다. 이러한 분리는 호스팅된 애플리케이션에 대한 리소스 할당이 코어 라우팅 및 스위칭 기능과 구별되도록 합니다.

Cisco IOS XE 장치용 애플리케이션 호스팅 인프라는 IOx(Cisco IOS + Linux)로 알려져 있으며, Cisco, 파트너 및 타사 개발자가 개발한 애플리케이션과 서비스를 네트워크 장치에 호스팅하여 다양한 하드웨어 플랫폼 간에 원활하게 통합됩니다.

특수 컨테이너 구축인 Guest Shell은 시스템 구축에 유용한 애플리케이션을 예시합니다.

Guest Shell은 Python을 비롯한 맞춤형 Linux 애플리케이션을 실행하도록 설계된 가상화된 Linux 기반 환경을 제공하여 Cisco 디바이스의 자동화된 제어 및 관리를 가능하게 합니다. 게스트 셸 컨테이너를 사용하면 사용자가 시스템 내에서 스크립트와 앱을 실행할 수 있습니다. 특히 인텔 x86 플랫폼에서 게스트 셸 컨테이너는 CentOS 8.0 최소 루트 파일 시스템을 갖춘 LXC(Linux Container)입니다. Cisco IOS XE Amsterdam 17.3.1 이상 릴리스에서는 Python V3.6만 지원됩니다. 런타임 시 CentOS 8.0의 Yum 유틸리티를 사용하여 Python 라이브러리를 추가로 설치할 수 있습니다. Python 패키지는 PIP 설치 패키지(PIP)를 사용하여 설치하거나 업데이트할 수도 있습니다.

게스트 셸에는 Python CLI 모듈을 사용하여 Cisco IOS XE 명령을 실행할 수 있는 Python API(Application Programming Interface)가 포함되어 있습니다. 이러한 방식으로 Python 스크립트는 자동화 기능을 강화하여 네트워크 엔지니어에게 구성 및 데이터 수집 작업을 자동화하는 스크립트를 개발할 수 있는 다양한 도구를 제공합니다. 이러한 스크립트는 CLI를 통해 수동으로 실행할 수 있지만, EEM 스크립트와 함께 사용하여 syslog 메시지, 인터페이스 이벤트 또는 명령 실행과 같은 특정 이벤트에 응답할 수도 있습니다. 실제로 EEM 스크립트를 트리거할 수 있는 모든 이벤트를 사용하여 Python 스크립트를 트리거할 수 있으므로 Cisco Catalyst 9000 스위치 내에서 자동화 잠재력이 확장됩니다.

Guest Shell을 설치하면 플래시 파일 시스템에 게스트 공유 디렉토리가 자동으로 생성됩니다. Python 스크립트 및 게스트 셸에서 액세스할 수 있는 파일 시스템입니다. 스태킹을 사용할 때 올바른 동기화를 보장하려면 이 폴더를 50MB 미만으로 유지하십시오.

EEM 스크립트와 함께 Python을 사용할 때의 이점

- Python은 Python 스크립트 내에서 복잡한 논리(예: 정규식, 루프, 일치)를 처리할 수 있도록 하여 EEM 스크립트의 자동화 기능을 확장합니다. 이 기능을 통해 더욱 강력한 EEM 스크립트를 작성할 수 있습니다.
- 잘 알려진 프로그래밍 언어인 Python은 Cisco IOS XE 장치를 자동화하려는 네트워크 엔지니어의 진입 장벽을 낮춥니다. 또한 유지 관리 및 가독성을 제공합니다.
- 또한 Python은 강력한 표준 라이브러리뿐 아니라 오류 처리 기능도 제공합니다.

EEM 스크립트와 함께 Python 사용 시 고려 사항

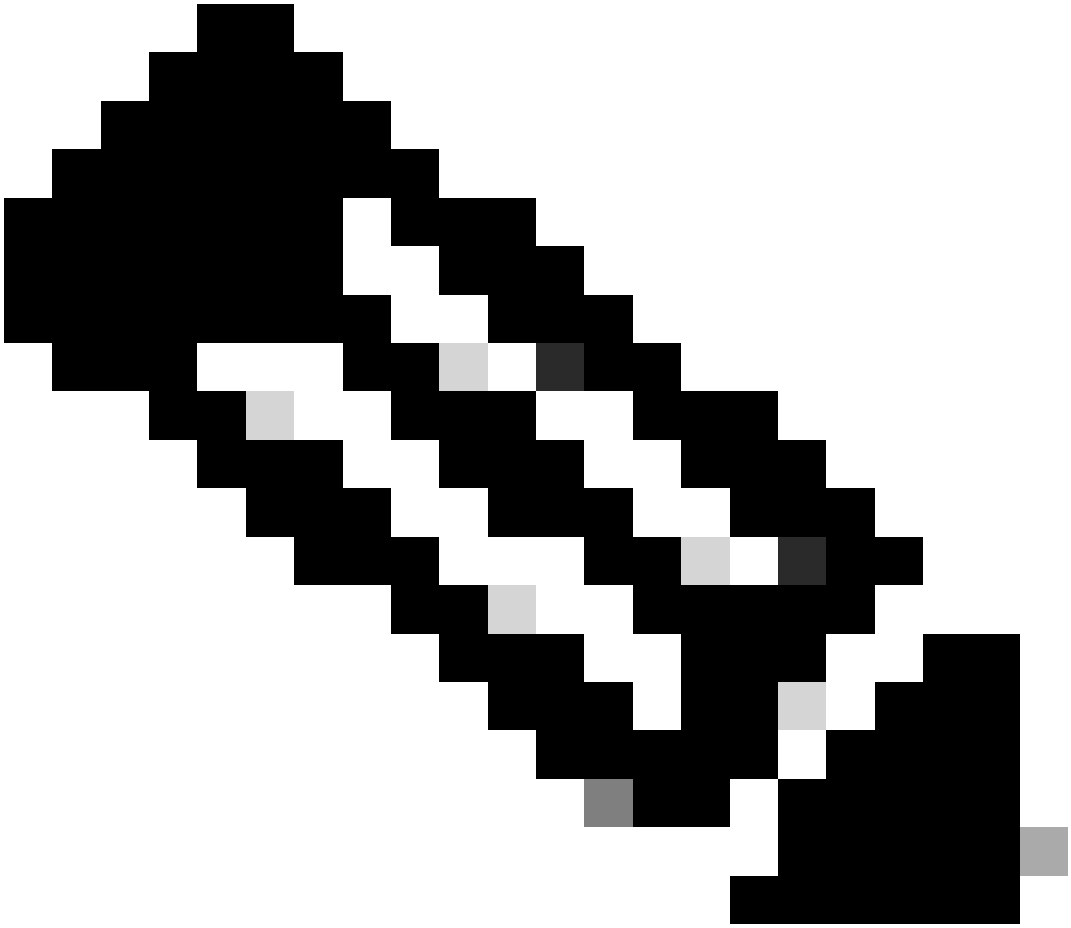
- 게스트 셸은 기본적으로 활성화되어 있지 않으므로 Python 스크립트를 실행하려면 먼저 활성화되어 있어야 합니다.

- Python 스크립트는 CLI에서 직접 생성할 수 없습니다. 개발 환경에서 먼저 개발한 다음 스위치의 플래시 메모리에 복사해야 합니다.

Cisco IOS XE의 SELinux

Cisco IOS XE 17.8.1부터 SELinux(Security-Enhanced Linux)를 지원하여 프로세스, 사용자 및 파일의 상호 작용 방식을 제어하는 정책으로 보안을 강화했습니다. SELinux 정책은 프로세스 또는 사용자가 액세스할 수 있는 작업 및 리소스를 정의합니다. 사용자 또는 프로세스가 정책에 허용되지 않는 작업(예: 리소스에 액세스하거나 명령을 실행)을 수행하려고 할 때 위반이 발생할 수 있습니다. SELinux는 2가지 모드로 작동할 수 있습니다.

1. 허용 모드: SELinux는 어떤 정책도 시행하지 않습니다. 그러나 모든 위반은 거부된 것처럼 기록합니다.
2. 시행: SELinux는 시스템에서 보안 정책을 적극적으로 시행합니다. 작업이 SELinux 정책을 위반하면 작업이 거부되고 기록됩니다.



참고: 기본 모드는 Cisco IOS XE 17.8.1에 도입되었을 때 허용으로 설정되었지만 버전

17.14.1부터 SELinux가 시행 모드에서 활성화됩니다.

Guest Shell을 사용하는 경우 시행 모드를 사용할 때 일부 리소스에 대한 액세스가 거부될 수 있습니다. 게스트 셸 또는 Python 스크립트를 사용하여 작업을 수행하려고 할 때 다음 로그와 유사한 권한 거부 오류가 발생하는 경우:

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

스크립트가 SELinux에 의해 거부되고 있는지 확인하려면 명령을 사용하여show platform software audit summary거부 수가 증가하고 있는지 확인합니다. 또한 SELinuxshow platform software audit all에 의해 차단된 작업의 로그를 표시합니다. AVC(Access Vector Cache)는 SELinux에서 액세스 제어 결정을 기록하는 데 사용되는 메커니즘입니다. 따라서 이 명령을 사용할 경우 type=AVC로 시작하는 로그를 찾습니다.

스크립트가 거부 및 차단된 경우 명령을 사용하여 SELinux를 허용 모드로 설정할 수 있습니다set platform software selinux permissive. 이 변경 사항은 실행 중인 컨피그레이션 또는 시작 컨피그레이션에 저장되지 않으므로 다시 로드한 후 모드가 적용으로 돌아갑니다. 따라서 스위치가 다시 로드될 때마다 이러한 변경 사항을 다시 적용해야 합니다. 를 사용하여 변경 사항을 확인할 수 있습니다show platform software selinux.

구성

EEM 및 Python 스크립트를 사용하여 스위치에서 작업을 자동화하려면 다음 단계를 수행합니다.

1. 게스트 셸을 활성화합니다.
2. Python 스크립트를 /flash/guest-share/ 디렉토리에 복사합니다. Cisco IOS XE에서 사용 가능한 모든 복사 메커니즘(예: SCP, FTP 또는 WebUI의 File Manager)을 사용할 수 있습니다. Python 스크립트가 플래시 메모리에 있으면 명령을 사용하여 실행할 수 있습니다guestshell run python3 /flash/guest-share/cat9k_script.py.
3. Python 스크립트를 실행하는 EEM 스크립트를 구성합니다. 이 설정을 사용하면 EEM 스크립트에서 제공하는 여러 이벤트 탐지기(예: syslog 메시지, CLI 패턴, Cron 스케줄러)를 사용하여 Python 스크립트를 트리거할 수 있습니다.

이 섹션에서는 1단계에 대해 설명합니다. 다음 섹션에서는 2단계와 3단계를 구현하는 방법을 보여주는 예를 제공합니다.

고정 IP 주소로 게스트 셸 활성화

게스트 셸을 활성화하려면 다음 프로세스를 따르십시오.

1. IOx를 활성화합니다.

<#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. 게스트 셸에 대한 애플리케이션 호스팅 네트워크를 구성합니다. 이 예에서는 AppGigabitEthernet 인터페이스를 사용하여 네트워크 액세스를 제공합니다. 그러나 관리 인터페이스(Gi0/0)도 사용할 수 있습니다.

```
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. 게스트 셸을 활성화합니다.

<#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. 게스트 셸의 유효성을 검사합니다. 이 예에서는 기본 게이트웨이와 cisco.com에 연결할 수 있는지 확인합니다. 또한 Python 3을 게스트 셸에서 실행할 수 있는지 확인합니다.

<#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#
```

DHCP IP 주소로 게스트 셸 활성화

일반적으로 게스트 셸 컨테이너에는 기본적으로 DHCP 클라이언트 서비스가 없기 때문에 게스트 셸은 고정 IP 주소로 구성됩니다. 게스트 셸에서 IP 주소를 동적으로 요청해야 하는 경우 DHCP 클라이언트 서비스를 설치해야 합니다. 다음 프로세스를 따릅니다.

1. 고정 IP 주소로 게스트 셸을 활성화하는 단계를 수행합니다. 그러나 이번에는 2단계 중에 앱 호스팅 구성에서 IP 주소를 할당하지 마십시오. 대신 다음 구성을 사용하십시오.

```
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. DHCP 클라이언트는 이 명령과 함께 Yum 유틸리티를 사용하여 설치할 수 있습니다sudo yum install dhcp-client. 그러나 CentOS Stream 8용 리포지토리는 서비스 해제되었습니다. 이를 해결하기 위해 DHCP 클라이언트 패키지를 수동으로 다운로드하여 설치할 수 있습니다. PC의 경우 CentOS Stream 8 볼트에서 이러한 패키지를 다운로드하여 tar 파일로 패키지합니다.

   - 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. 스위치의dhcp-client.tar/flash/guest-share/ 디렉토리에 파일을 복사합니다.

4. Guest Shell bash 세션을 입력하고 Linux 명령을 실행하여 DHCP 클라이언트를 설치하고 IP 주소를 요청합니다.

<#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
```

활용 사례

활용 사례 1: SCP 서버의 컨피그레이션 변경 사항 자동 저장

경우에 따라 명령을 사용할 때마다 스위치 컨피그레이션을 서버에 자동으로 저장하는write memory것이 좋습니다. 이 방법은 변경 사항의 기록을 유지 관리하는 데 도움이 되며 필요한 경우 컨피그레이션 롤백을 허용합니다. 서버를 선택할 때 TFTP와 SCP를 모두 사용할 수 있지만 SCP 서버는 추가 보안 레이어를 제공합니다.

Cisco IOS 아카이브 기능은 이 기능을 제공합니다. 그러나 SCP 자격 증명을 컨피그레이션에 숨길

수 없다는 것이 큰 단점입니다. 서버 경로는 실행 중인 컨피그레이션과 시작 컨피그레이션에서 모두 일반 텍스트로 표시됩니다.

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

Guest Shell 및 Python을 사용하면 자격 증명을 숨긴 상태로 동일한 기능을 구현할 수 있습니다. 이는 실제 SCP 자격 증명을 저장하기 위해 게스트 셸 내의 환경 변수를 활용하여 수행됩니다. 따라서 SCP 서버 자격 증명은 실행 중인 컨피그레이션에 표시되지 않습니다.

이 접근 방식에서는 실행 중인 컨피그레이션에 EEM 스크립트만 표시되는 반면, Python 스크립트는 자격 증명으로`copy running-config scp:`명령을 구성하고 실행할 EEM 스크립트로 전달합니다.

이 예에서는 다음 단계를 수행합니다.

1. Python 스크립트를 /flash/guest-share 디렉토리에 복사합니다. 이 스크립트는 환경 변수 SCP_USER 및 SCP_PASSWORD를 읽고 EEM 스크립트가`copy startup-config scp:`실행할 수 있도록 명령을 반환합니다. 스크립트에는 인수로 SCP 서버 IP 주소가 필요합니다. 또한 이 스크립트는 /flash/guest-share/TAC-write-memory-log.txt에 있는 영구 파일에서 명령`write memory`이 실행될 때마다 로그를 유지합니다. 다음은 Python 스크립트입니다.

```
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
```

이 예에서는 Python 스크립트가 TFTP 서버를 사용하여 스위치에 복사됩니다.

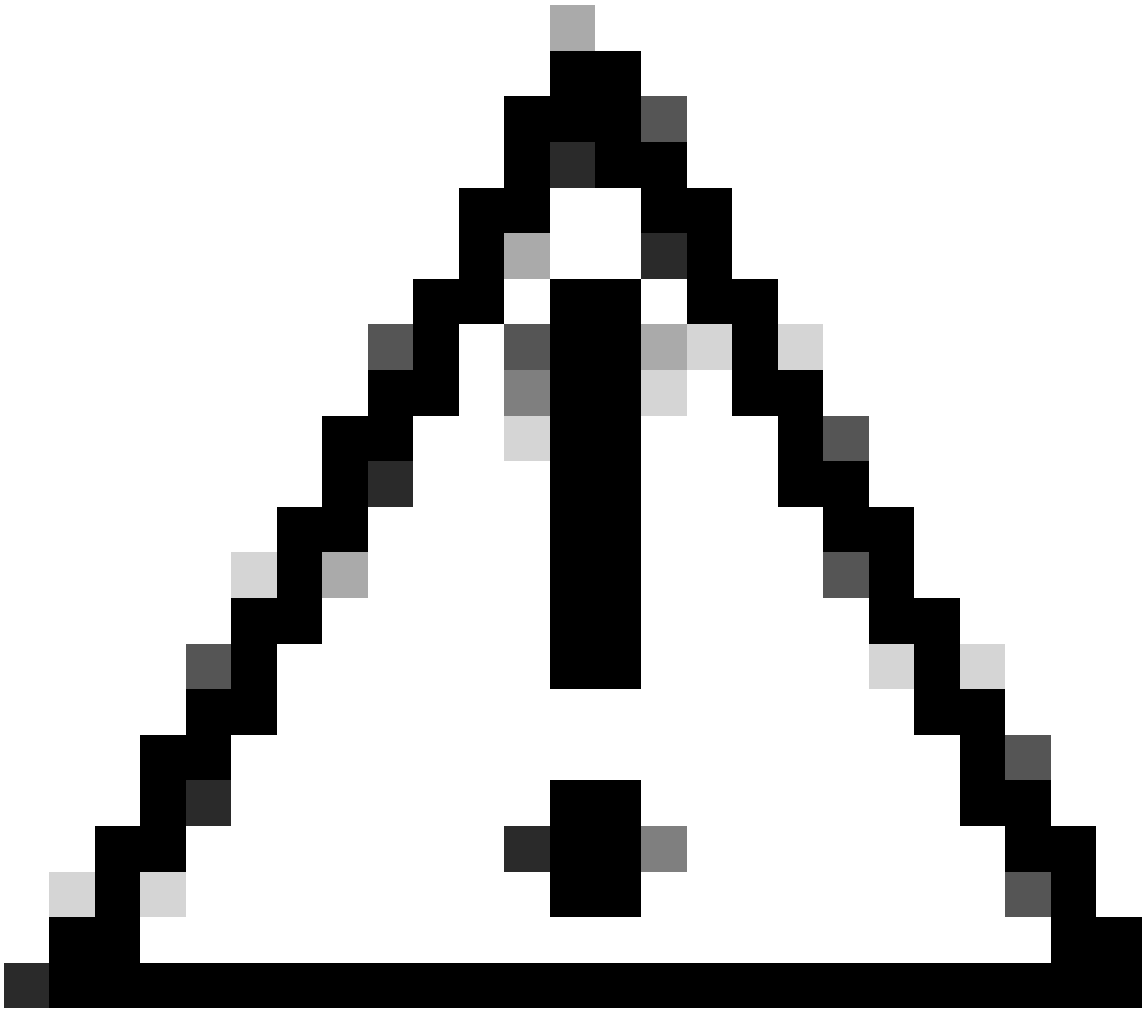
```
<#root>

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. EEM 스크립트를 설치합니다. 이 스크립트는 Python 스크립트를 호출합니다. 는copy startup-config scp:SCP 서버에 컨피그레이션을 저장하는 데 필요한 명령을 반환합니다. 그런 다음 EEM 스크립트는 Python 스크립트에서 반환한 명령을 실행합니다.

```
event manager applet Python-config-backup authorization bypass
 event cli pattern "^write|write memory|copy running-config startup-config" sync no skip no maxrun
 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. 파일에 게스트 셸 환경 변수를 삽입하여~/.bashrc설정합니다. 이렇게 하면 스위치가 다시 로드된 후에도 게스트 셸을 열 때마다 환경 변수가 지속됩니다. 다음 두 행을 추가합니다.

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



주의: 이 예에서 사용되는 자격 증명은 교육용으로만 사용됩니다. 프로덕션 환경에서는 사용할 수 없습니다. 사용자는 이러한 자격 증명을 자신의 고유한 보안 환경 관련 자격 증명으로 교체해야 합니다.

파일에 다음 변수를 추가하는~/.bashrc프로세스입니다.

<#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. Python 스크립트를 수동으로 실행하여 올바른copy명령을 반환하는지 확인하고 작업을 영구 파일에TAC-write-memory-log.txt기록합니다.

```
<#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. EEM 스크립트를 테스트합니다. 이 EEM 스크립트는 성공 또는 실패 여부에 관계없이 복사 작업의 결과와 함께 syslog도 전송합니다. 다음은 성공적인 실행의 예입니다.

```
<#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.
```

실패한 전송을 테스트하기 위해 SCP 서버가 종료됩니다. 이 실행 실패 결과:

<#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.
```

활용 사례 2: STP 토폴로지 변경의 증분 모니터링

이 예는 스패닝 트리 불안정성과 관련된 문제를 모니터링하고 어떤 인터페이스에서 TCN(Topology Change Notifications)을 수신하는지 식별하는 데 유용합니다. EEM 스크립트는 지정된 시간 간격으로 주기적으로 실행되며 show 명령을 실행하고 TCN이 증가했는지 확인하는 Python 스크립트를 호출합니다.

EEM 명령만 사용하여 이 스크립트를 생성하려면 for 루프와 여러 regex 일치를 사용해야 하므로 번거롭습니다. 따라서 이 예에서는 EEM 스크립트가 이 복잡한 논리를 Python에 위임하는 방법을 보여 줍니다.

이 예에서는 다음 단계를 수행합니다.

1. Python 스크립트를 /flash/guest-share/ 디렉토리에 복사합니다. 이 스크립트는 다음 작업을 수행합니다.
    1. 이 명령은show spanning-tree detail명령을 실행하고 출력을 구문 분석하여 딕셔너리의 각 VLAN에 대한 TCN 정보를 저장합니다.
    2. 구문 분석된 TCN 정보를 이전 스크립트 실행의 JSON 파일에 있는 데이터와 비교합니다. 각 VLAN에 대해 TCN이 증가한 경우 syslog 메시지가 이 예와 유사한 정보와 함께 전송됩니다.

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

    3. 현재 TCN 정보를 JSON 파일에 저장하여 다음 실행 시 비교합니다. 다음은 Python 스크립트입니다.

```
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()
```

이 예에서는 Python 스크립트가 TFTP 서버를 사용하여 스위치에 복사됩니다.

```
<#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. EEM 스크립트를 설치합니다. 이 예에서 EEM 스크립트의 유일한 작업은 Python 스크립트를 실행하는 것입니다. 이 스크립트는 TCN 증분이 탐지되면 로그 메시지를 전송합니다. 이 예에서는 EEM 스크립트가 5분마다 실행됩니다.

```
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. 스크립트의 기능을 검증하기 위해 Python 스크립트를 수동으로 실행하거나 EEM 스크립트가 호출할 때까지 5분 정도 기다릴 수 있습니다. 두 경우 모두 VLAN에 대해 TCN이 증가한 경우에만 syslog가 전송됩니다.

```
<#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. EEM 스크립트가 5분마다 실행될 때까지 기다리면서 테스트합니다. VLAN에 대해 TCN이 증가한 경우 syslog 메시지가 전송됩니다. 이 특정 예에서 TCN은 VLAN 30에서 지속적으로 증가하고 있으며, 이러한 일정한 TCN을 수신하는 인터페이스는 Twe1/0/3입니다.

```
<#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.
```

관련 정보

- Cisco IOS XE Cupertino 17.9.x의 프로그래밍 기능 컨피그레이션 가이드(장: 게스트 셸)
- EEM의 모범 사례 및 유용한 스크립트 이해
- Cisco Catalyst 9000 Series 스위치의 애플리케이션 호스팅 백서

이 번역에 관하여

Cisco는 전 세계 사용자에게 다양한 언어로 지원 콘텐츠를 제공하기 위해 기계 번역 기술과 수작업 번역을 병행하여 이 문서를 번역했습니다. 아무리 품질이 높은 기계 번역이라도 전문 번역가의 번역 결과물만큼 정확하지는 않습니다. Cisco Systems, Inc.는 이 같은 번역에 대해 어떠한 책임도 지지 않으며 항상 원본 영문 문서(링크 제공됨)를 참조할 것을 권장합니다.