Automatisierung von Catalyst Switches der Serie 9000 mit Python-Skripten

Inhalt

Einleitung

Voraussetzungen

- Anforderungen
- Verwendete Komponenten
- Konventionen

Hintergrundinformationen

- Guest Shell- und Python-Skripts
- Vorteile von Python mit EEM-Skripten
- Überlegungen zur Verwendung von Python mit EEM-Skripten
- SELinux in Cisco IOS XE

Konfigurieren

- Gastshell mit statischer IP-Adresse aktivieren
- Aktivieren der Gast-Shell mit einer DHCP-IP-Adresse

Anwendungsfälle

- Anwendungsfall 1: Automatisches Speichern von Konfigurationsänderungen auf einem SCP-Server
- Anwendungsfall 2: Überwachung von Erhöhungen bei STP-Topologieänderungen

Zugehörige Informationen

Einleitung

In diesem Dokument wird beschrieben, wie EEM mit Python-Skripts für die Automatisierung der Konfiguration und Datenerfassung auf Catalyst 9000-Switches erweitert wird.

Voraussetzungen

Anforderungen

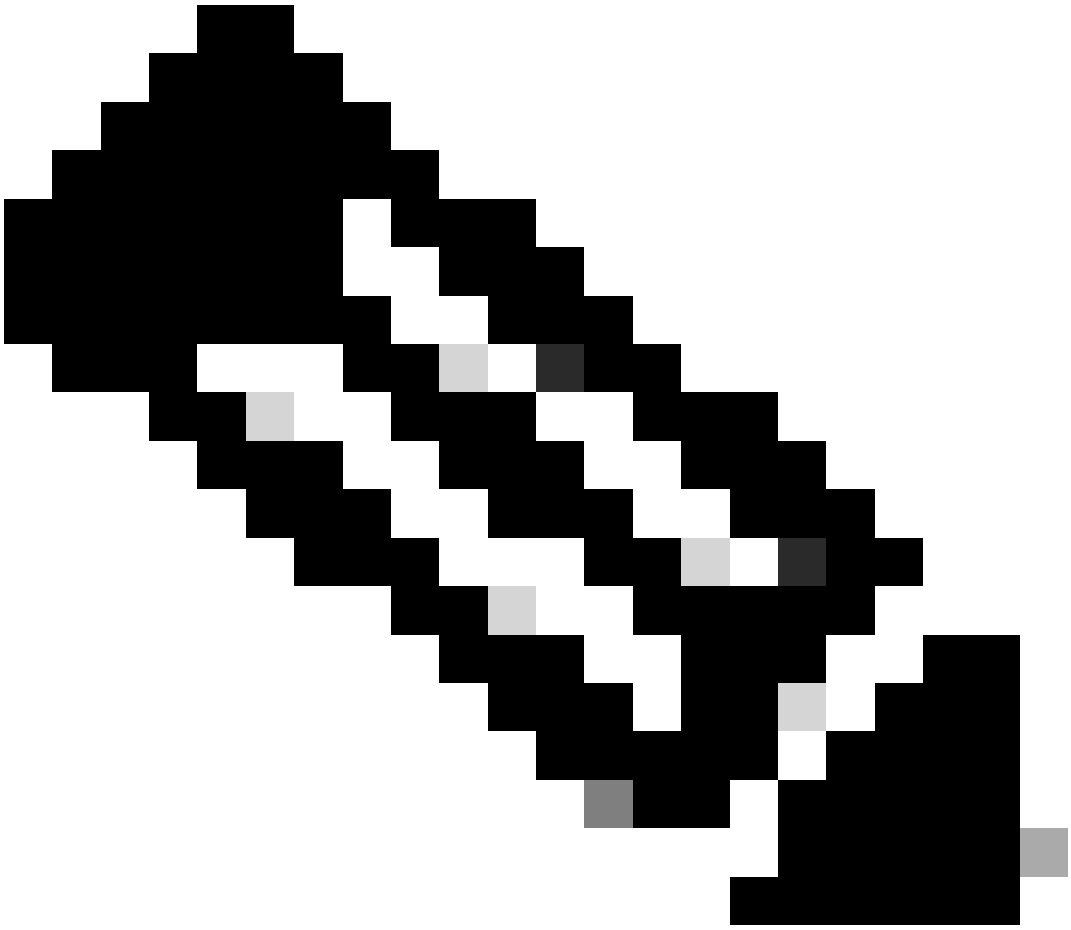
Cisco empfiehlt, dass Sie mit den folgenden Themen vertraut sind:

- Cisco IOS® und Cisco IOS® XE EEM
- Anwendungshosting und Gast-Shell
- Python-Scripting
- Linux-Befehle

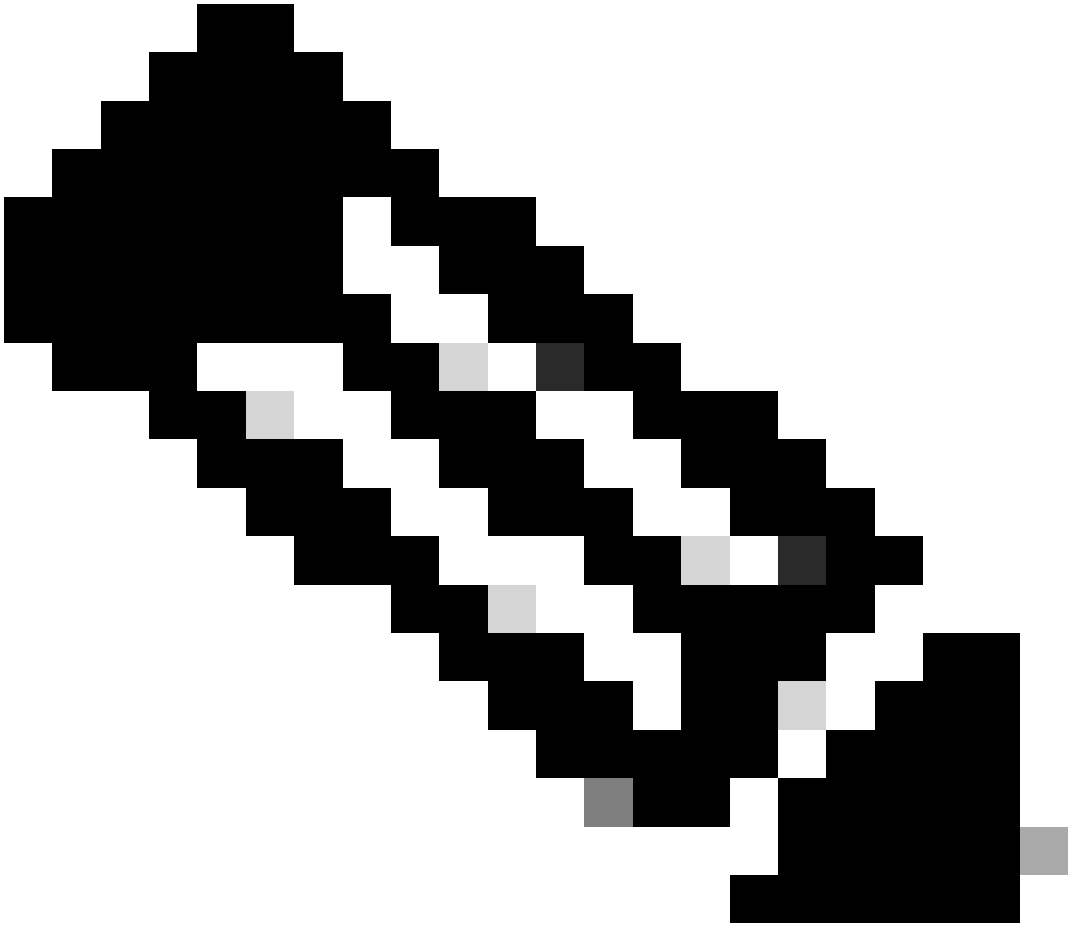
Verwendete Komponenten

Die Informationen in diesem Dokument basierend auf folgenden Software- und Hardware-Versionen:

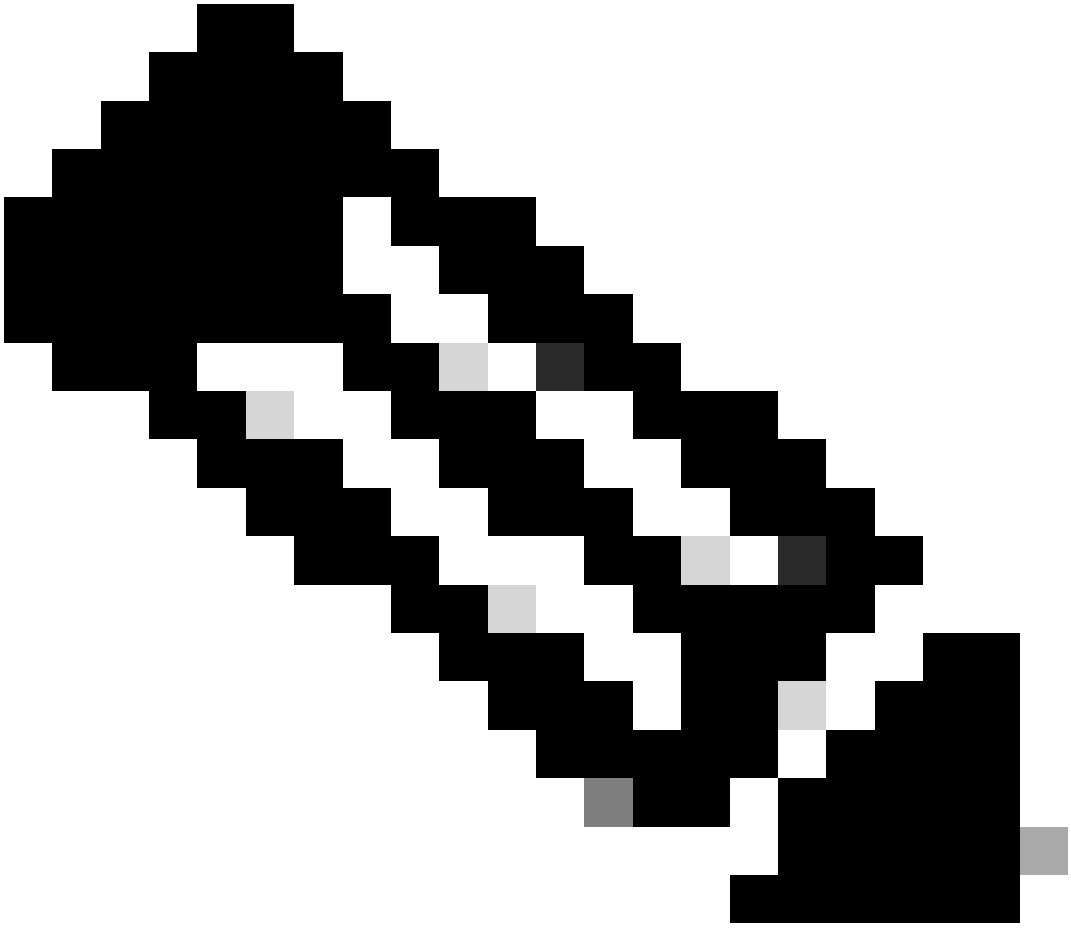
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1 und neuere Versionen



Anmerkung: Die Befehle, um diese Funktionen auf anderen Cisco Plattformen zu aktivieren, finden Sie im entsprechenden Konfigurationsleitfaden.



Anmerkung: Catalyst 9200L-Switches unterstützen keine Guest Shell.



Anmerkung: Diese Skripte werden vom Cisco TAC nicht unterstützt und werden zu Informationszwecken wie besehen bereitgestellt.

Die Informationen in diesem Dokument beziehen sich auf Geräte in einer speziell eingerichteten Testumgebung. Alle Geräte, die in diesem Dokument benutzt wurden, begannen mit einer gelöschten (Nichterfüllungs) Konfiguration. Wenn Ihr Netzwerk in Betrieb ist, stellen Sie sicher, dass Sie die möglichen Auswirkungen aller Befehle kennen.

Konventionen

Informationen zu Dokumentkonventionen finden Sie unter Cisco Technical Tips Conventions (Technische Tipps von Cisco zu Konventionen).

Hintergrundinformationen

Guest Shell- und Python-Skripts

Das Anwendungshosting auf den Cisco Catalyst Switches der Serie 9000 bietet Partnern und Entwicklern neue Innovationsmöglichkeiten, da Netzwerkgeräte mit einer Anwendungs-Laufzeitumgebung zusammengeführt werden können.

Sie unterstützt containerisierte Anwendungen und bietet vollständige Isolierung vom Hauptbetriebssystem und dem Cisco IOS XE Kernel. Durch diese Trennung wird sichergestellt, dass sich die Ressourcenzuweisungen für gehostete Anwendungen von den Kern-Routing- und Switching-Funktionen unterscheiden.

Die Infrastruktur für das Anwendungshosting von Cisco IOS XE-Geräten wird als IOx (Cisco IOS + Linux) bezeichnet. Sie vereinfacht das Hosting von Anwendungen und Services, die von Cisco, Partnern und Entwicklern von Drittanbietern auf Netzwerkgeräten entwickelt wurden, und stellt so eine nahtlose Integration über verschiedene Hardwareplattformen hinweg sicher.

Die Guest Shell, eine spezielle Containerbereitstellung, ist ein Beispiel für eine Anwendung, die für die Systembereitstellung von Vorteil ist.

Die Guest Shell bietet eine virtualisierte, auf Linux basierende Umgebung, in der benutzerdefinierte Linux-Anwendungen wie Python ausgeführt werden können, um die automatisierte Steuerung und Verwaltung von Cisco Geräten zu ermöglichen. Der Guest Shell Container erlaubt es Benutzern, Skripte und Apps innerhalb des Systems auszuführen. Insbesondere auf Intel x86-Plattformen ist der Guest Shell-Container ein Linux-Container (LXC) mit einem minimalen CentOS 8.0-Root-Dateisystem. In Cisco IOS XE Amsterdam 17.3.1 und höheren Versionen wird nur Python V3.6 unterstützt. Weitere Python-Bibliotheken können während der Laufzeit mit dem Yum-Dienstprogramm in CentOS 8.0 installiert werden. Python-Pakete können auch mit Pip Install Packages (PIP) installiert oder aktualisiert werden.

Die Guest Shell umfasst eine Python-API (Application Programming Interface), mit der Cisco IOS XE-Befehle über das Python-CLI-Modul ausgeführt werden können. Auf diese Weise erweitern Python-Skripte die Automatisierungsfunktionen und bieten Netzwerktechnikern ein vielseitiges Tool zur Entwicklung von Skripten zur Automatisierung von Konfigurations- und Datenerfassungsaufgaben. Diese Skripts können zwar manuell über die CLI ausgeführt werden, sie können jedoch auch zusammen mit EEM-Skripts verwendet werden, um auf bestimmte Ereignisse zu reagieren, z. B. Syslog-Meldungen, Schnittstellenereignisse oder Befehlsausführungen. Praktisch jedes Ereignis, das ein EEM-Skript auslösen kann, kann auch zum Auslösen eines Python-Skripts verwendet werden, wodurch das Automatisierungspotenzial der Cisco Catalyst Switches der Serie 9000 erweitert wird.

Wenn die Guest Shell installiert ist, wird automatisch ein Guest-Share-Verzeichnis im Flash-Dateisystem erstellt. Dies ist das Dateisystem, auf das von den Python-Skripten und der Guest Shell aus zugegriffen werden kann. Um eine ordnungsgemäße Synchronisierung bei Verwendung des Stacking sicherzustellen, sollten Sie diesen Ordner unter 50 MB halten.

Vorteile von Python mit EEM-Skripten

- Python erweitert die Automatisierungsfunktionen von EEM-Skripts, indem komplexe Logik (wie reguläre Ausdrücke, Schleifen und Übereinstimmungen) innerhalb des Python-Skripts verarbeitet werden kann. Diese Funktion bietet die Möglichkeit, leistungsstärkere EEM-

Skripts zu erstellen.

- Als bekannte Programmiersprache verringert Python die Einstiegshürde für Netzwerktechniker, die Cisco IOS XE-Geräte automatisieren möchten. Darüber hinaus bietet es Wartungsfreundlichkeit und Lesbarkeit.
- Python bietet auch Fehlerbehandlungsfunktionen sowie eine leistungsstarke Standardbibliothek.

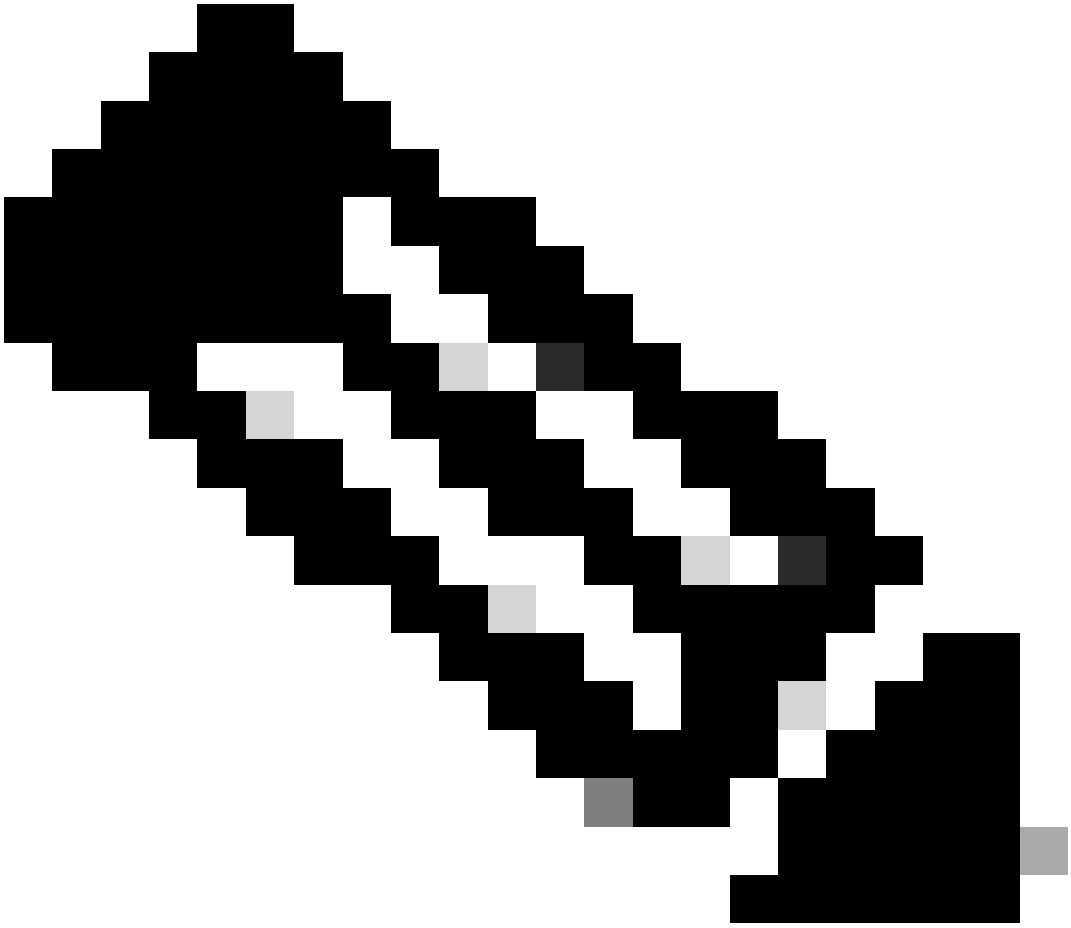
Überlegungen zur Verwendung von Python mit EEM-Skripten

- Guest Shell ist nicht standardmäßig aktiviert, daher muss es aktiviert sein, bevor Python-Skripte ausgeführt werden können.
- Python-Skripts können nicht direkt in der CLI erstellt werden. Sie müssen zunächst in einer Entwicklungsumgebung entwickelt und dann in den Flash-Speicher des Switches kopiert werden.

SELinux in Cisco IOS XE

Beginnend mit Cisco IOS XE 17.8.1 wurde die Unterstützung für Security-Enhanced Linux (SELinux) eingeführt, um Sicherheit durch Richtlinien durchzusetzen, die festlegen, wie Prozesse, Benutzer und Dateien miteinander interagieren. Die SELinux-Richtlinie definiert, welche Aktionen und Ressourcen von einem Prozess oder Benutzer aufgerufen werden dürfen. Eine Verletzung kann auftreten, wenn ein Benutzer oder Prozess versucht, eine Aktion auszuführen, die von der Richtlinie nicht zugelassen ist, z. B. Zugriff auf eine Ressource oder Ausführung eines Befehls. SELinux kann in zwei verschiedenen Modi betrieben werden:

1. Zulässiger Modus: SELinux erzwingt keine Richtlinien. Dennoch werden alle Verstöße so protokolliert, als würden sie abgelehnt.
2. Durchsetzen: SELinux setzt die Sicherheitsrichtlinien im System aktiv durch. Wenn eine Aktion die SELinux-Richtlinie verletzt, wird die Aktion abgelehnt und protokolliert.



Anmerkung: Der Standardmodus wurde bei der Einführung in Cisco IOS XE 17.8.1 auf "Permissive" gesetzt. Ab Version 17.14.1 ist SELinux jedoch im Erzwingungsmodus aktiviert.

Bei Verwendung der Guest Shell kann der Zugriff auf einige Ressourcen im Durchsetzungsmodus verweigert werden. Wenn bei dem Versuch, eine Aktion mit Guest Shell oder einem Python-Skript auszuführen, ein Fehler aufgrund verweigerter Berechtigungen auftritt, ähnlich wie in diesem Protokoll:

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

Um zu überprüfen, ob ein Skript von SELinux abgelehnt wird, verwenden Sie den Befehl`show platform software audit summary`, um zu überprüfen, ob die Anzahl der Ablehnungen steigt. Zusätzlich wird ein`show platform software audit all`Protokoll der von SELinux blockierten Aktionen angezeigt. Der Access Vector

Cache (AVC) ist der Mechanismus, mit dem Zugriffskontrollentscheidungen in SELinux aufgezeichnet werden. Wenn Sie diesen Befehl verwenden, suchen Sie nach Protokollen, die mit type=AVC beginnen.

Wenn ein Skript abgelehnt und blockiert wird, kann SELinux mit dem Befehl`set platform software selinux permissive`auf den permissiven Modus gesetzt werden. Diese Änderung wird nicht in der aktuellen Konfiguration oder der Startkonfiguration gespeichert, sodass der Modus nach einem Neuladen wieder auf Erzwingung zurückgesetzt wird. Daher muss diese Änderung bei jedem Neuladen des Switches erneut angewendet werden. Die Änderung kann mit`show platform software selinux`verifiziert werden.

Konfigurieren

Um Aufgaben auf dem Switch mithilfe von EEM- und Python-Skripten zu automatisieren, gehen Sie wie folgt vor:

1. Aktivieren Sie die Guest Shell.
2. Kopieren Sie das Python-Skript in das Verzeichnis /flash/guest-share/. Sie können jeden in Cisco IOS XE verfügbaren Kopiermechanismus verwenden, z. B. SCP, FTP oder den Flex Manager in der WebUI. Sobald sich das Python-Skript im Flash-Speicher befindet, können Sie es mit dem Befehl`guestshell run python3 /flash/guest-share/cat9k_script.py`ausführen.
3. Konfigurieren Sie ein EEM-Skript, das das Python-Skript ausführt. Bei dieser Konfiguration kann das Python-Skript mit einer beliebigen Ereigniserkennung ausgelöst werden, die von EEM-Skripts bereitgestellt wird, z. B. eine Syslog-Meldung, ein CLI-Muster und ein Cron Scheduler.

In diesem Abschnitt wird Schritt 1 erläutert. Im nächsten Abschnitt finden Sie Beispiele für die Implementierung der Schritte 2 und 3.

Gastshell mit statischer IP-Adresse aktivieren

Folgen Sie diesem Prozess, um die Guest Shell zu aktivieren:

1. Aktivieren Sie 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. Konfigurieren des Anwendungshostnetzwerks für die Gastshell In diesem Beispiel wird die AppGigabitEthernet-Schnittstelle verwendet, um den Netzwerkzugriff bereitzustellen. Es kann jedoch auch die Management-Schnittstelle (Gi0/0) verwendet werden.

```
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. Aktivieren Sie die Guest Shell.

<#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. Validieren Sie die Guest Shell. In diesem Beispiel wird überprüft, ob die Erreichbarkeit sowohl mit dem Standardgateway als auch mit cisco.com besteht. Überprüfen Sie außerdem, ob Python 3 über die Guest Shell ausgeführt werden kann.

<#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#
```

Aktivieren der Gast-Shell mit einer DHCP-IP-Adresse

In der Regel wird die Guest Shell mit einer statischen IP-Adresse konfiguriert, da der Container der Guest Shell standardmäßig nicht über den DHCP-Client-Dienst verfügt. Wenn die Guest Shell eine IP-Adresse dynamisch anfordern muss, muss der DHCP-Client-Dienst installiert werden. Folgen Sie diesem Prozess:

1. Führen Sie die Schritte aus, um die Guest Shell mit einer statischen IP-Adresse zu aktivieren. Weisen Sie die IP-Adresse in der App-Hosting-Konfiguration in Schritt 2 diesmal jedoch nicht zu. Verwenden Sie stattdessen die folgende Konfiguration:

```
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. Der DHCP-Client kann mithilfe des Dienstprogramms Yum mit dem Befehl`sudo yum install dhcp-client`installiert werden. Repositorys für CentOS Stream 8 wurden jedoch deaktiviert. Um dies zu erreichen, können die DHCP-Client-Pakete manuell heruntergeladen und installiert werden. Laden Sie diese Pakete auf einem PC aus dem CentOS Stream 8-Tresor herunter,

und packen Sie sie in eine TAR-Datei.

- 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. Kopieren Sie die dhcp-client.tar Datei in das Verzeichnis /flash/guest-share/ auf dem Switch.
4. Geben Sie eine Guest Shell-Bash-Sitzung ein, und führen Sie die Linux-Befehle aus, um den DHCP-Client zu installieren und eine IP-Adresse anzufordern.

```
<#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
```

Anwendungsfälle

Anwendungsfall 1: Automatisches Speichern von Konfigurationsänderungen auf einem SCP-Server

In einigen Situationen ist es von Vorteil, Switch-Konfigurationen bei jeder Verwendung des`write memory`Befehls automatisch auf einem Server zu speichern. Auf diese Weise können Änderungen aufgezeichnet und bei Bedarf die Konfiguration zurückgesetzt werden. Bei der Auswahl eines Servers können sowohl TFTP als auch SCP verwendet werden. Ein SCP-Server bietet jedoch eine zusätzliche Sicherheitsebene.

Die Cisco IOS-Archivierungsfunktion stellt diese Funktion bereit. Ein wesentlicher Nachteil ist jedoch, dass SCP-Anmeldeinformationen nicht in der Konfiguration verborgen werden können. Der Serverpfad wird sowohl in der aktuellen als auch in der Startkonfiguration im Nur-Text-Format angezeigt.

```
Switch#show running-config | section archive
archive
```

```
path scp://cisco:Cisco!123@10.31.121.224/
write-memory
```

Mit der Guest Shell und Python können Sie dieselbe Funktionalität erreichen und die Anmeldedaten verborgen lassen. Dies geschieht durch die Nutzung von Umgebungsvariablen in der Guest Shell, um die eigentlichen SCP-Anmeldeinformationen zu speichern. Daher sind die SCP-Serveranmeldeinformationen in der aktuellen Konfiguration nicht sichtbar.

Bei diesem Ansatz zeigt die ausgeführte Konfiguration nur das EEM-Skript an, während die Python-Skripts den`copy running-config scp:`Befehl mit den Anmeldedaten erstellen und an das auszuführende EEM-Skript übergeben.

Führen Sie für dieses Beispiel die folgenden Schritte aus:

1. Kopieren Sie das Python-Skript in das Verzeichnis /flash/guest-share. Dieses Skript liest die Umgebungsvariablen SCP_USER und SCP_PASSWORD und gibt den`copy startup-config scp:`Befehl zurück, sodass das EEM-Skript ihn ausführen kann. Für das Skript ist die IP-Adresse des SCP-Servers als Argument erforderlich. Darüber hinaus führt das Skript ein Protokoll jedes Mal, wenn der Befehl`write memory`in einer persistenten Datei unter /flash/guest-share/TAC-write-memory-log.txt ausgeführt wird. Dies ist das Python-Skript:

```
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 diesem Beispiel wird das Python-Skript mithilfe eines TFTP-Servers auf den Switch kopiert:

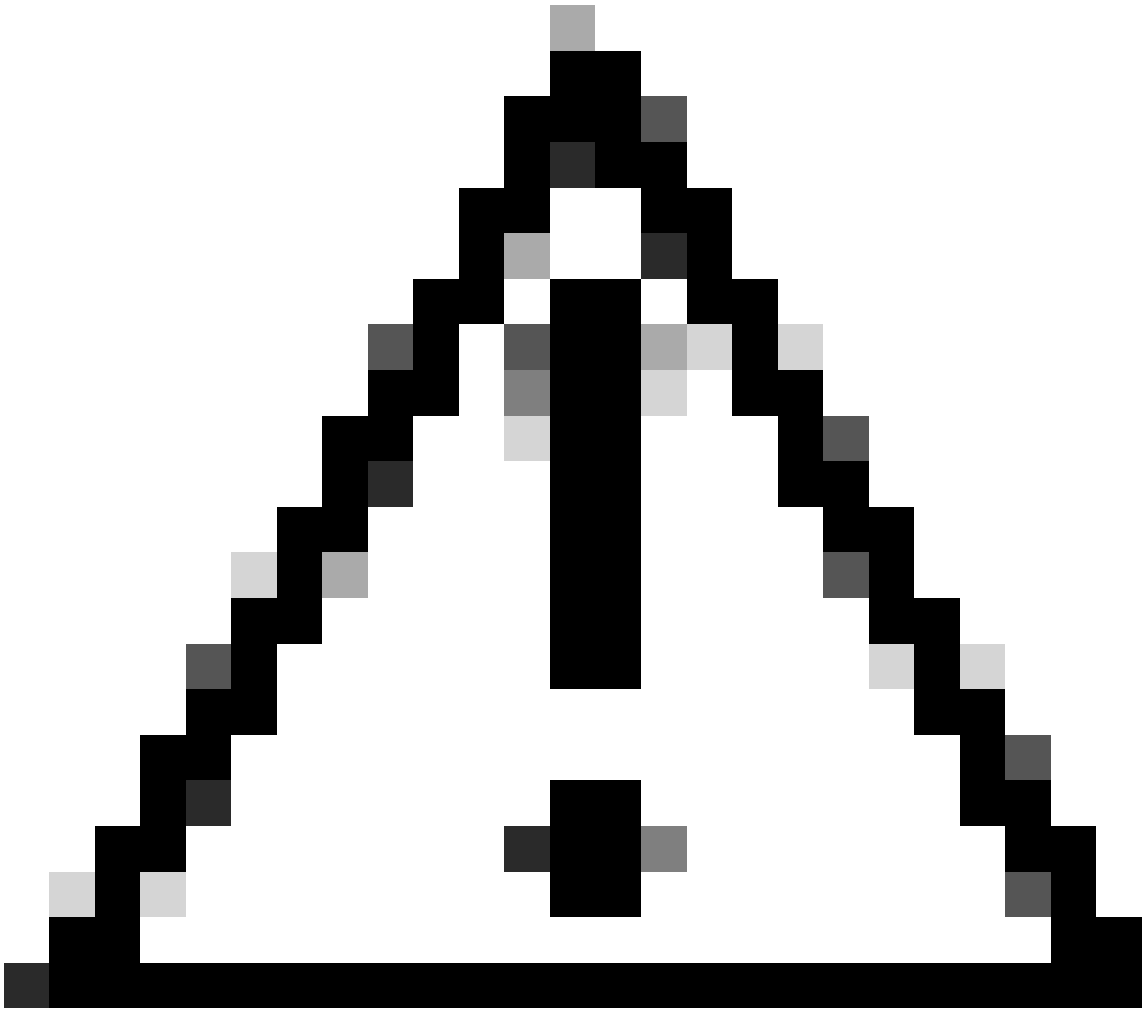
```
<#root>

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. Installieren Sie das EEM-Skript. Dieses Skript ruft das Python-Skript auf. der den`copy startup-config scp:`Befehl zurückgibt, der zum Speichern der Konfiguration auf dem SCP-Server erforderlich ist. Das EEM-Skript führt dann den vom Python-Skript zurückgegebenen Befehl aus.

```
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. Legen Sie die Umgebungsvariablen der Guest Shell fest, indem Sie sie in die`~/.bashrc`Datei einfügen. Dadurch wird sichergestellt, dass die Umgebungsvariablen auch nach dem erneuten Laden des Switches bei jedem Öffnen einer Guest Shell erhalten bleiben. Fügen Sie die folgenden beiden Zeilen hinzu:

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



Vorsicht: Die in diesem Beispiel verwendeten Anmeldeinformationen dienen nur zu Schulungszwecken. Sie sind nicht für den Einsatz in Produktionsumgebungen vorgesehen. Die Benutzer müssen diese Anmeldeinformationen durch ihre eigenen sicheren, umgebungsspezifischen Anmeldeinformationen ersetzen.

Dies ist der Prozess, um diese Variablen zur~/.bashrcDatei hinzuzufügen:

<#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. Führen Sie das Python-Skript manuell aus, um sicherzustellen, dass der richtige`copy`Befehl zurückgegeben wird, und protokollieren Sie den Vorgang in der persistenten`TAC-write-memory-log.txt`Datei.

```
<#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. Testen des EEM-Skripts. Dieses EEM-Skript sendet auch ein Syslog mit dem Ergebnis des Kopiervorgangs, unabhängig davon, ob dieser erfolgreich war oder fehlgeschlagen ist. Hier ein Beispiel für einen erfolgreichen Lauf:

<#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.
```

Um eine fehlgeschlagene Übertragung zu testen, wird der SCP-Server heruntergefahren.
Dies ist das Ergebnis dieses fehlgeschlagenen Ausführens:

<#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.
```

Anwendungsfall 2: Überwachung von Erhöhungen bei STP-Topologieänderungen

Dieses Beispiel ist nützlich, um Probleme im Zusammenhang mit der Spanning Tree-Instabilität zu überwachen und zu ermitteln, welche Schnittstelle Topology Change Notifications (TCNs) empfängt. Das EEM-Skript wird in einem festgelegten Zeitintervall periodisch ausgeführt und ruft ein Python-Skript auf, das den Befehl "show" ausführt und überprüft, ob die TCNs erhöht wurden.

Das Erstellen dieses Skripts, das nur EEM-Befehle verwendet, würde die Verwendung von für Schleifen und mehrere Regex-Übereinstimmungen erfordern, was umständlich wäre. Daher zeigt dieses Beispiel, wie das EEM-Skript diese komplexe Logik an Python delegiert.

Führen Sie für dieses Beispiel die folgenden Schritte aus:

1. Kopieren Sie das Python-Skript in das Verzeichnis /flash/guest-share/. Dieses Skript führt folgende Aufgaben aus:

    1. Er führt den`show spanning-tree detail`Befehl aus und analysiert die Ausgabe, um TCN-Informationen für jedes VLAN in einem Dictionary zu speichern.

    2. Es vergleicht die analysierten TCN-Informationen mit den Daten in der JSON-Datei aus dem vorherigen Skriptlauf. Wenn die TCNs für jedes VLAN erhöht wurden, wird eine Syslog-Meldung mit ähnlichen Informationen wie in diesem Beispiel gesendet:

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

    3. Es speichert die aktuellen TCN-Informationen in einer JSON-Datei, die beim nächsten Lauf verglichen werden soll. Dies ist das Python-Skript:

```
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 diesem Beispiel wird das Python-Skript mithilfe eines TFTP-Servers auf den Switch kopiert:

```
<#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. Installieren Sie das EEM-Skript. In diesem Beispiel besteht die einzige Aufgabe des EEM-Skripts darin, das Python-Skript auszuführen, das eine Protokollmeldung sendet, wenn ein TCN-Inkrement erkannt wird. In diesem Beispiel wird das EEM-Skript alle 5 Minuten ausgeführt.

```
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. Um die Funktionalität des Skripts zu überprüfen, können Sie das Python-Skript manuell ausführen oder fünf Minuten warten, bis das EEM-Skript es aufruft. In beiden Fällen wird ein Syslog nur dann gesendet, wenn die TCNs für ein VLAN erhöht haben.

```
<#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. Testen Sie das EEM-Skript, indem Sie darauf warten, dass es alle fünf Minuten ausgeführt wird. Wenn die TCNs für ein VLAN erhöht wurden, wird eine Syslog-Meldung gesendet. In diesem speziellen Beispiel ist zu beachten, dass die TCNs im VLAN 30 ständig zunehmen, und die Schnittstelle, die diese konstanten TCNs empfängt, ist 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.
```

Zugehörige Informationen

- Leitfaden zur Konfiguration der Programmierbarkeit, Cisco IOS XE Cupertino 17.9.x (Kapitel: Guest Shell)
- Best Practices und nützliche Skripte für EEM
- Whitepaper "Application Hosting auf Cisco Catalyst Switches der Serie 9000"

Informationen zu dieser Übersetzung

Cisco hat dieses Dokument maschinell übersetzen und von einem menschlichen Übersetzer editieren und korrigieren lassen, um unseren Benutzern auf der ganzen Welt Support-Inhalte in ihrer eigenen Sprache zu bieten. Bitte beachten Sie, dass selbst die beste maschinelle Übersetzung nicht so genau ist wie eine von einem professionellen Übersetzer angefertigte. Cisco Systems, Inc. übernimmt keine Haftung für die Richtigkeit dieser Übersetzungen und empfiehlt, immer das englische Originaldokument (siehe bereitgestellter Link) heranzuziehen.