Pythonスクリプトを使用したCatalyst 9000スイッチの自動化

内容

はじめに

このドキュメントでは、Catalyst 9000スイッチでの設定とデータ収集を自動化するためにPythonスクリプトを使用してEEMを拡張する方法について説明します。

前提条件

要件

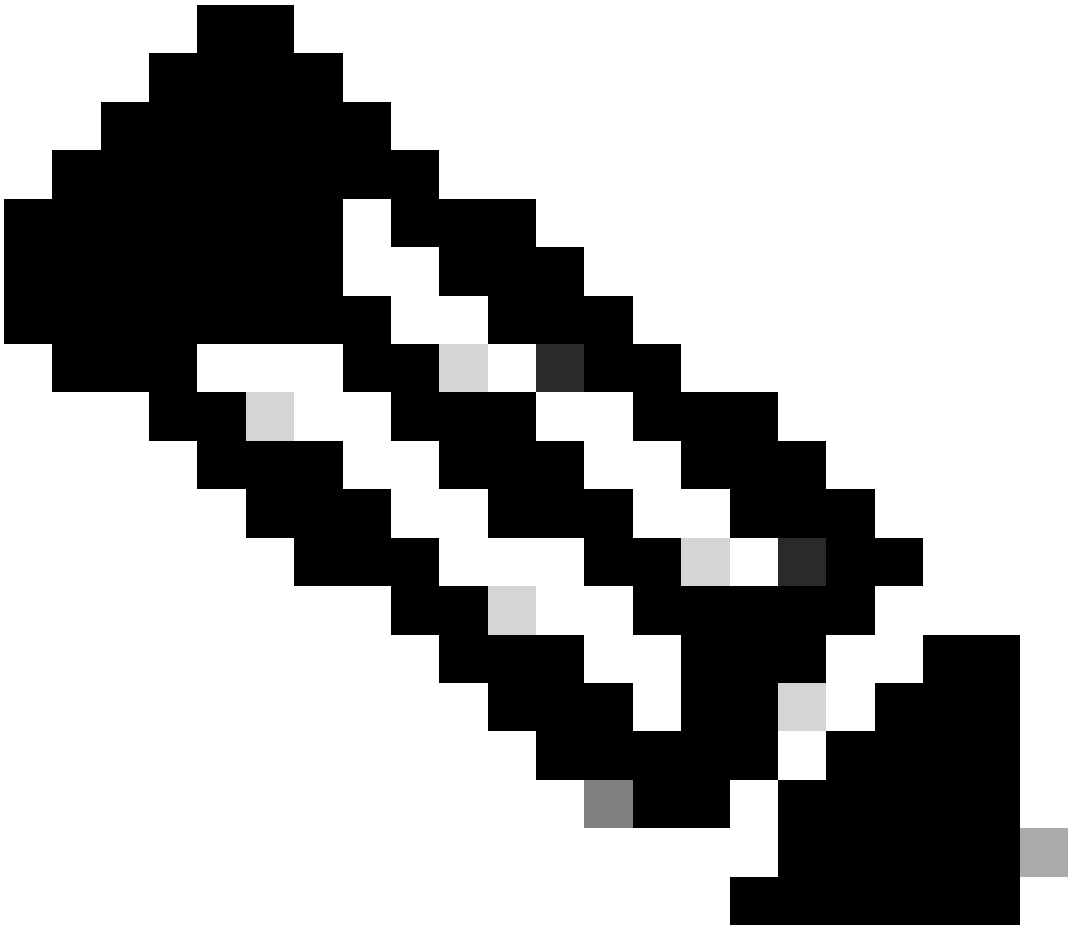
次の項目に関する知識と知識があることが推奨されます。

- Cisco IOS®およびCisco IOS® XE EEM
- アプリケーションホスティングおよびゲストシェル
- Python スクリプティング
- Linuxコマンド

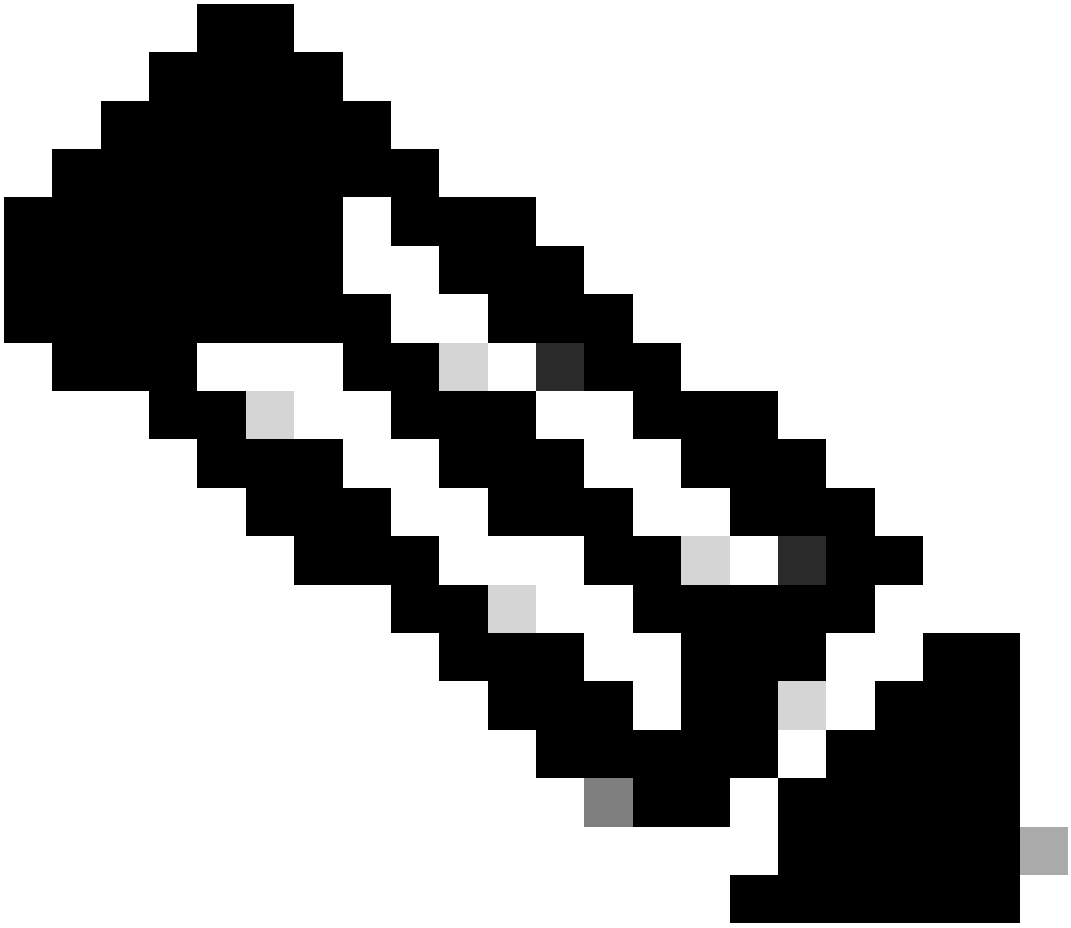
使用するコンポーネント

このドキュメントの情報は、次のソフトウェアとハードウェアのバージョンに基づいています。

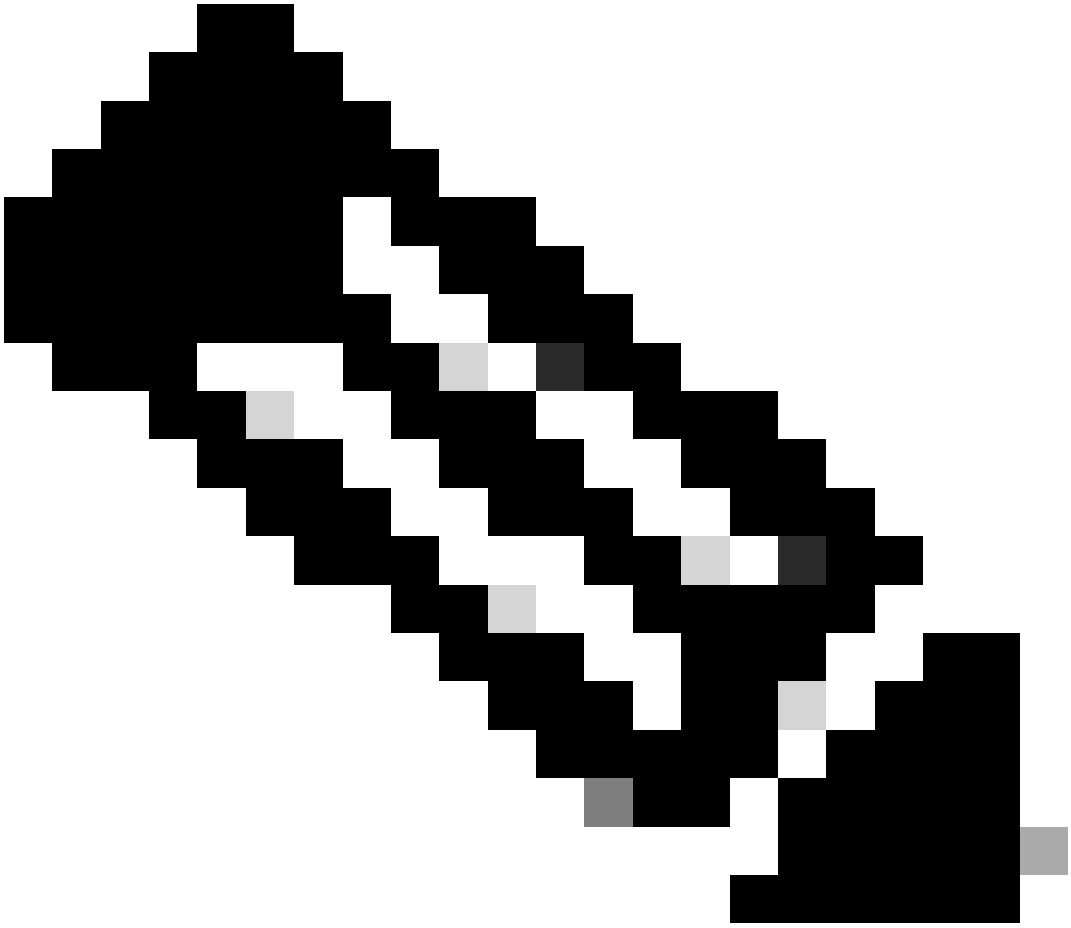
- Catalyst 9200
- Catalyst 9300
- Catalyst 9400
- Catalyst 9500
- Catalyst 9600
- Cisco IOS XE 17.9.1以降のバージョン



注：シスコの他のプラットフォームでこれらの機能を有効にするために使用されるコマンドについては、該当するコンフィギュレーション ガイドを参照してください。



注:Catalyst 9200Lスイッチはゲストシェルをサポートしていません。



注：これらのスクリプトはCisco TACではサポートしておらず、教育のためにも現状のまま提供されています。

このドキュメントの情報は、特定のラボ環境にあるデバイスに基づいて作成されました。このドキュメントで使用するすべてのデバイスは、クリアな（デフォルト）設定で作業を開始しています。本稼働中のネットワークでは、各コマンドによって起こる可能性がある影響を十分確認してください。

表記法

ドキュメント表記の詳細は、『シスコ テクニカル ティップスの表記法』を参照してください。

背景説明

ゲストシェルとPythonスクリプト

Cisco Catalyst 9000ファミリスイッチのアプリケーションホスティングは、ネットワークデバイスをアプリケーションランタイム環境とマージできるため、パートナーや開発者にイノベーションの機会をもたらします。

コンテナ化されたアプリケーションをサポートし、メインオペレーティングシステムおよびCisco IOS XEカーネルから完全に分離されています。この分離により、ホステッドアプリケーションへのリソース割り当てが、コアルーティングおよびスイッチング機能とは別になります。

Cisco IOS XEデバイス向けのアプリケーションホスティングインフラストラクチャはIOx(Cisco IOS + Linux)と呼ばれ、シスコ、パートナー、サードパーティの開発者が開発したアプリケーションやサービスをネットワークデバイス上でホスティングし、多様なハードウェアプラットフォーム間のシームレスな統合を実現します。

特殊なコンテナ展開であるゲストシェルは、システム展開に適したアプリケーションの例です。

ゲストシェルは、Pythonなどのカスタムアプリケーションを実行するように設計された仮想化されたLinuxベースの環境を提供し、シスコデバイスの自動制御と管理を可能にします。ゲストシェルコンテナを使用すると、ユーザはシステム内でスクリプトやアプリケーションを実行できます。特にIntel x86プラットフォームでは、ゲストシェルコンテナはCentOS 8.0最小ルートファイルシステムを備えたLinuxコンテナ(LXC)です。Cisco IOS XE Amsterdam 17.3.1以降のリリースでは、Python V3.6のみがサポートされています。追加のPythonライブラリは、CentOS 8.0のYumユーティリティを使用して実行時にインストールできます。Pythonパッケージは、Pip Install Packages(PIP)を使用してインストールまたは更新することもできます。

ゲストシェルには、Python CLIモジュールを使用してCisco IOS XEコマンドを実行できるPython Application Programming Interface(API)が含まれています。このように、Pythonスクリプトは自動化機能を強化し、ネットワークエンジニアに設定およびデータ収集タスクを自動化するためのスクリプトを開発するための汎用ツールを提供します。これらのスクリプトはCLIから手動で実行できますが、EEMスクリプトと組み合わせて使用し、syslogメッセージ、インターフェイスイベント、コマンド実行などの特定のイベントに応答することもできます。実際には、EEMスクリプトをトリガーできる任意のイベントを使用してPythonスクリプトをトリガーし、Cisco Catalyst 9000スイッチ内の自動化の可能性を拡大することもできます。

ゲストシェルがインストールされると、フラッシュファイルシステムにゲスト共有ディレクトリが自動的に作成されます。これは、Pythonスクリプトおよびゲストシェルからアクセスできるファイルシステムです。スタック構成を使用する際に適切な同期を確保するには、このフォルダを50 MB未満にしてください。

EEMスクリプトでPythonを使用する利点

- Pythonは、複雑なロジック（正規表現、ループ、一致など）をPythonスクリプト内で処理できるようにすることで、EEMスクリプトの自動化機能を拡張します。この機能により、より強力なEEMスクリプトを作成できます。

- Pythonは、Cisco IOS XEデバイスの自動化を希望するネットワークエンジニアにとって、よく知られたプログラミング言語として、入力の障壁を低くします。さらに、メンテナンス性と読みやすさを提供します。

- Pythonは、エラー処理機能と強力な標準ライブラリも提供します。

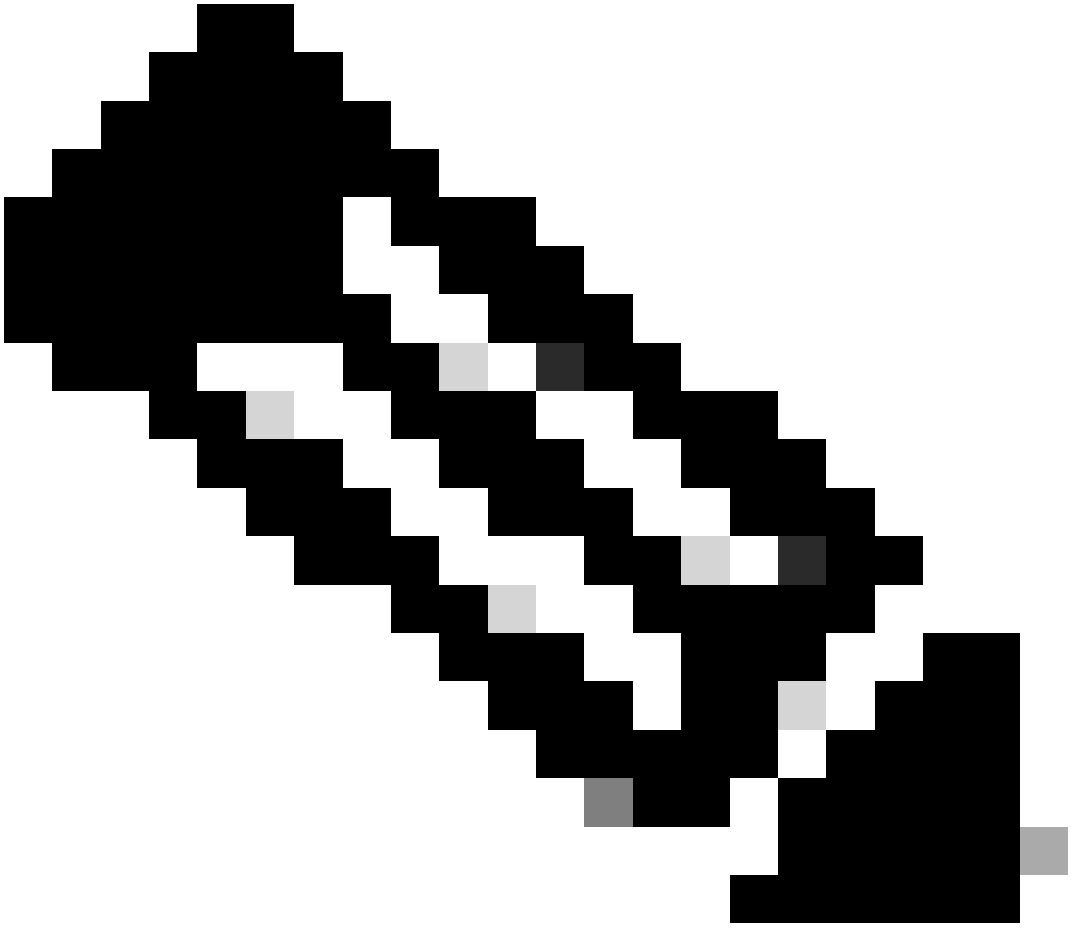
EEMスクリプトでPythonを使用する場合の考慮事項

- ゲストシェルはデフォルトでは有効になっていないため、Pythonスクリプトを実行する前に有効にする必要があります。
- PythonスクリプトはCLIで直接作成できません。最初に開発環境で開発し、次にスイッチのフラッシュメモリにコピーする必要があります。

Cisco IOS XEでのSELinux

Cisco IOS XE 17.8.1以降、プロセス、ユーザ、およびファイルの相互対話方法を制御するポリシーを使用してセキュリティを強化するために、Security-Enhanced Linux(SELinux)のサポートが導入されました。SELinuxポリシーは、プロセスまたはユーザーがアクセスできるアクションとリソースを定義します。違反は、リソースへのアクセスやコマンドの実行など、ポリシーで許可されていないアクションをユーザーまたはプロセスが実行しようとしたときに発生する可能性があります。SELinuxは、2つの異なるモードで動作できます。

1. 許容モード：SELinuxはポリシーを適用しません。ただし、違反が拒否されたかのように口グが残ります。
2. 適用：SELinuxは、システムにセキュリティポリシーを積極的に適用します。アクションがSELinuxポリシーに違反すると、そのアクションは拒否され、ログに記録されます。



注:Cisco IOS XE 17.8.1で導入された時点で、デフォルトモードはpermissiveに設定されていました。ただし、バージョン17.14.1以降では、SELinuxは強制モードで有効になっています。

ゲストシェルを使用する場合、強制モードの使用時に一部のリソースへのアクセスを拒否できます。ゲストシェルまたはPythonスクリプトを使用してアクションを実行しようとすると、次の口グに似た「permission denied」エラーが発生する場合があります。

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

スクリプトがSELinuxによって拒否されているかどうかを確認するには、show platform software audit summaryコマンドを使用して、拒否数が増加しているかどうかを確認します。さらに、show platform software audit allはSELinuxによってブロックされたアクションのログを表示します。Access Vector Cache (AVC)は、SELinuxでアクセス制御の決定を記録するために使用されるメカニズムです。

そのため、このコマンドを使用するときは、type=AVCで始まるログを探してください。

スクリプトが拒否およびブロックされている場合、set platform software selinux permissiveコマンドを使用して、SELinuxをpermissiveモードに設定できます。この変更は実行コンフィギュレーションまたはスタートアップコンフィギュレーションには保存されないため、リロード後にモードがenforcingに戻ります。したがって、スイッチがリロードされるたびに、この変更を再適用する必要があります。変更は、show platform software selinuxを使用して確認できます。

設定

EEMおよびPythonスクリプトを使用してスイッチ上のタスクを自動化するには、次の手順を実行します。

1. ゲストシェルを有効にします。
2. Pythonスクリプトを/flash/guest-share/ディレクトリにコピーします。SCP、FTP、WebUIのFile Managerなど、Cisco IOS XEで使用可能な任意のコピーメカニズムを使用できます。Pythonスクリプトをフラッシュメモリに保存したら、guestshell run python3 /flash/guest-share/cat9k_script.pyコマンドを使用してこのスクリプトを実行できます。
3. Pythonスクリプトを実行するEEMスクリプトを設定します。この設定により、syslogメッセージ、CLIパターン、Cronスケジューラなど、EEMスクリプトが提供する複数のイベントディテクタのいずれかを使用してPythonスクリプトをトリガーできます。

このセクションでは、ステップ1について説明します。次のセクションでは、ステップ2および3の実行方法を示す例を示します。

静的IPアドレスによるゲストシェルの有効化

次のプロセスに従って、ゲストシェルを有効にします。

1. IOxを有効にします。

<#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. ゲストシェル用のアプリケーションホスティングネットワークを設定します。この例では、AppGigabitEthernetインターフェイスを使用してネットワークアクセスを提供していますが、管理インターフェイス(Gi0/0)も使用できます。

```
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. ゲストシェルを有効にします。

<#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. ゲストシェルを検証します。この例では、デフォルトゲートウェイおよびcisco.comとの到達可能性があることを検証します。また、Python 3をゲストシェルから実行できることを検証します。

<#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アドレスによるゲストシェルの有効化

通常、ゲストシェルコンテナにはデフォルトでDHCPクライアントサービスがないため、ゲストシェルはスタティックIPアドレスで設定されます。ゲストシェルがIPアドレスを動的に要求する必要がある場合は、DHCPクライアントサービスをインストールする必要があります。次のプロセスに従います。

1. スタティックIPアドレスを使用してゲストシェルを有効にするには、次の手順を実行します。ただし、今回はステップ2の間にアプリケーションホスティング設定でIPアドレスを割り当てないでください。代わりに、次の設定を使用します。

```
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クライアントをインストールするには、Yumユーティリティでコマンド`sudo yum install dhcp-client`を使用します。ただし、CentOS Stream 8のリポジトリは廃止されました。この問題に対処するには、DHCPクライアントパッケージを手動でダウンロードしてインストールします。PCで、CentOS Stream 8ボールトからこれらのパッケージをダウンロードし、tarファイルにパッケージ化します。

   - 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. スイッチの/flash/guest-share/ディレクトリにdhcp-client.tarファイルをコピーします。
4. ゲストシェルのbashセッションに入り、Linuxコマンドを実行してDHCPクライアントをインストールし、IPアドレスを要求します。

<#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
```

使用例

使用例1:SCPサーバでの設定変更の自動保存

場合によっては、write memoryコマンドを使用するたびに、スイッチの設定をサーバに自動的に保存することが有効です。この方法は、変更の記録を保持するのに役立ち、必要に応じて設定をロールバックできます。サーバを選択する際には、TFTPとSCPの両方を使用できますが、SCPサーバを使用すると、セキュリティをさらに強化できます。

Cisco IOSのアーカイブ機能はこの機能を提供します。ただし、重大な欠点は、SCPクレデンシャルを設定で隠すことができないことです。サーバパスは、実行コンフィギュレーションとスタートアップコンフィギュレーションの両方でプレーンテキストで表示されます。

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

ゲストシェルとPythonを使用することで、クレデンシャルを隠したまま同じ機能を実現できます。これは、実際のSCPクレデンシャルを保存するためにゲストシェル内で環境変数を活用するこ

とで実行されます。その結果、SCPサーバクレデンシャルは実行コンフィギュレーションには表示されません。

この方法では、実行コンフィギュレーションではEEMスクリプトのみが表示されますが、Pythonスクリプトはクレデンシャルを使用して`copy running-config scp:`コマンドを作成し、実行するEEMスクリプトにそのコマンドを渡します。

この例では、次の手順を実行します。

1. Pythonスクリプトを/flash/guest-shareディレクトリにコピーします。このスクリプトは、環境変数SCP_USERおよびSCP_PASSWORDを読み取り、`copy startup-config scp:`コマンドを返します。これにより、EEMスクリプトがそれを実行できるようになります。スクリプトには、引数としてSCPサーバのIPアドレスが必要です。また、このスクリプトは、/flash/guest-share/TAC-write-memory-log.txtにある永続的なファイルでコマンド`write memory`が実行されるたびに、ログを維持します。Pythonスクリプトは次のとおりです。

```
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
```

次の例では、TFTPサーバを使用してPythonスクリプトをスイッチにコピーしています。

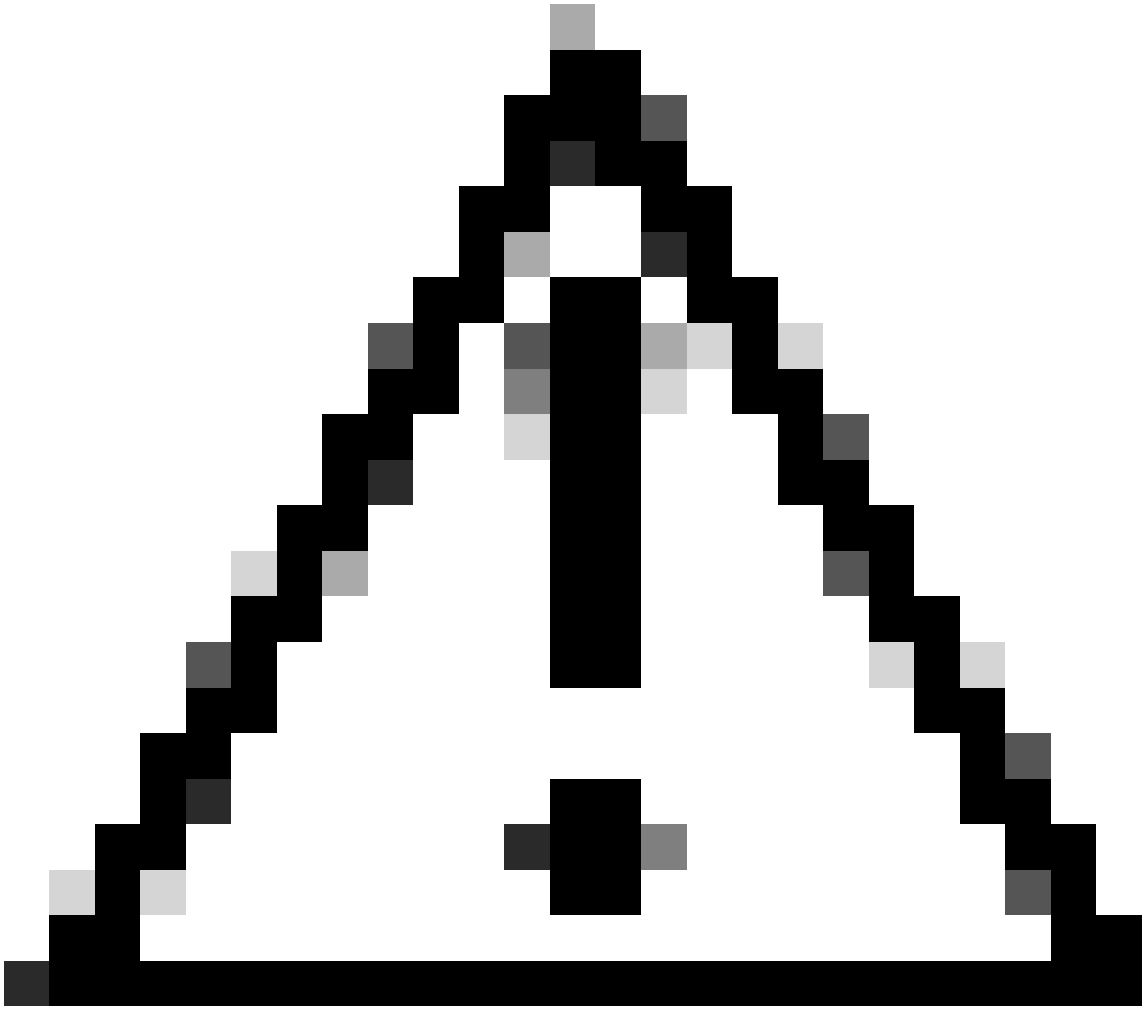
```
<#root>

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スクリプトをインストールします。このスクリプトはPythonスクリプトを呼び出します。このスクリプトは、SCPサーバに設定を保存するために必要なcopy startup-config scp:コマンドを返します。次に、EEMスクリプトはPythonスクリプトから返されたコマンドを実行します。

```
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. ゲストシェルの環境変数を~/.bashrcファイルに挿入して設定します。これにより、スイッチがリロードされた後でも、ゲストシェルが開かれるたびに環境変数が保持されるようになります。次の2行を追加します。

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



注意：この例で使用しているクレデンシャルは、情報提供のみを目的としています。これらは実稼働環境での使用を目的としていません。ユーザは、これらのクレデンシャルを独自のセキュアな環境固有のクレデンシャルに置き換える必要があります。

次に、これらの変数を~/.bashrcファイルに追加するプロセスを示します。

<#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スクリプトを手動で実行して、正しいcopyコマンドが返されることを検証し、操作を永続的TAC-write-memory-log.txtファイルに記録します。

```
<#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スクリプトをテストします。このEEMスクリプトは、成功または失敗に関係なく、コピー操作の結果を含むsyslogも送信します。正常に実行された例を次に示します。

```
<#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.
```

失敗した転送をテストするには、SCPサーバをシャットダウンします。これは、この失敗した実行の結果です。

```
<#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.
```

使用例2:STPトポロジ変更の増加の監視

この例は、スパニングツリーの不安定性に関連する問題を監視し、どのインターフェイスがトポロジ変更通知(TCN)を受信しているかを特定するのに役立ちます。 EEMスクリプトは、指定された時間間隔で定期的に実行され、showコマンドを実行するPythonスクリプトを呼び出して、TCNが増加したかどうかを確認します。

EEMコマンドのみを使用してこのスクリプトを作成するには、forループと複数の正規表現の一致を使用する必要があり、手間がかかります。したがって、この例では、EEMスクリプトがこの複雑なロジックをPythonに委任する方法を示します。

この例では、次の手順を実行します。

1. Pythonスクリプトを/flash/guest-share/ディレクトリにコピーします。このスクリプトは次のタスクを実行します。

    1. show spanning-tree detailコマンドを実行し、出力を解析して、各VLANのTCN情報をディクショナリに保存します。

    2. 解析されたTCN情報と、前のスクリプト実行のJSONファイル内のデータを比較します。各VLANでTCNが増加している場合は、次の例のような情報を含むsyslogメッセージが送信されます。

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

    3. 現在のTCN情報をJSONファイルに保存して、次の実行時に比較できるようにします。Pythonスクリプトは次のとおりです。

```
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()
```

次の例では、TFTPサーバを使用してPythonスクリプトをスイッチにコピーしています。

```
<#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スクリプトをインストールします。この例では、EEMスクリプトの唯一のタスクは、TCNの増加が検出された場合にログメッセージを送信するPythonスクリプトを実行することです。この例では、EEMスクリプトは5分ごとに実行されます。

```
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. スクリプトの機能を検証するには、Pythonスクリプトを手動で実行するか、EEMスクリプトによって呼び出されるまで5分間待機します。いずれの場合も、VLANに対してTCNが増加した場合にだけsyslogが送信されます。

```
<#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. 5分ごとに実行されるのを待って、EEMスクリプトをテストします。いずれかのVLANでTCNが増加している場合は、syslogメッセージが送信されます。この例では、VLAN 30でTCNが継続的に増加しており、これらの一定のTCNを受信するインターフェイスがTwe1/0/3であることに注目してください。

```
<#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.
```

関連情報

- プログラマビリティ設定ガイド、Cisco IOS XE Cuppertino 17.9.x（章：ゲストシェル）
- EEMのベストプラクティスと便利なスクリプトを理解する
- Cisco Catalyst 9000シリーズスイッチでのアプリケーションホスティングに関するホワイトペーパー

翻訳について
シスコは世界中のユーザにそれぞれの言語でサポート コンテンツを提供するために、機械と人による翻訳を組み合わせて、本ドキュメントを翻訳しています。ただし、最高度の機械翻訳であっても、専門家による翻訳のような正確性は確保されません。シスコは、これら翻訳の正確性について法的責任を負いません。原典である英語版（リンクからアクセス可能）もあわせて参照することを推奨します。